Thèse

Présentée au

Centre Universitaire d'Education
et de Formation des Adultes
de GRENOBLE

pour obtenir

le titre d'Ingénieur du Conservatoire National
des Arts et Métiers

par

Jean-Pierre Le Heiget

Généralisation de la notion d'espaces virtuels sous les systèmes CP-67/CMS

Thèse soutenue le 5 Juillet 1972 devant

MM	P. Namian	Président
	L. Bolliet	
	M. Bellot	Examinateurs
	C. Hans	
	P. Arnaud	

Thèse

Présentée au

Centre Universitaire d'Education
et de Formation des Adultes
de GRENOBLE

pour obtenir

le titre d'Ingénieur du Conservatoire National
des Arts et Métiers

par

Jean-Pierre Le Heiget

Généralisation de la notion d'espaces virtuels sous les systèmes CP-67/CMS

Thèse soutenue le 5 Juillet 1972 devant

MM	P. Namian	Président
	L. Bolliet	
	M. Bellot	Examinateurs
	C. Hans	
	P. Arnaud	

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

LISTE DES PROFESSEURS

Président : Monsieur Michel SOUTIF
Vice-Président : Monsieur Gabriel CAU

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Georges	Clinique des maladies infectieuses
	ARNAUD Paul	Chimie
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BENOIT Jean	Radioélectricité
	BESSON Jean	Electrochimie
	BEZES Henri	Chirurgie générale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BONNIER Etienne	Electrochimie Electrométallurgie
	BOUCHERLE André	Chimie et Toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BRAVARD Yves	Géographie
	BRISSONNEAU Pierre	Physique du Solide
	BUYLE-BODIN Maurice	Electronique
	CABANAC Jean	Pathologie chirurgicale
	CABANEL Guy	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et Toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Oto-Rhino-Laryngologie
	CHATEAU Robert	Thérapeutique
	CHENE Marcel	Chimie papetière
	COEUR André	Pharmacie chimique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie Pathologique
	CRAYA Antoine	Mécanique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de Dermatologie et Syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	FELICI Noël	Electrostatique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques Pures
	GALVANI Octave	Mathématiques Pures

MM. GASTINEL Noël	Analyse numérique
GERBER Robert	Mathématiques Pures
GIRAUD Pierre	Géologie
KLEIN Joseph	Mathématiques Pures
Mme KOFLER Lucie	Botanique et Physiologie végétale
MM. KOSZUL Jean-Louis	Mathématiques Pures
KRAVTCHENKO Julien	Mécanique
KUNTZMANN Jean	Mathématiques Appliquées
LACAZE Albert	Thermodynamique
LACHARME Jean	Biologie végétale
LATREILLE René	Chirurgie générale
LATURAZE Jean	Biochimie pharmaceutique
LAURENT Pierre	Mathématiques Appliquées
LEDRU Jean	Clinique médicale B
LLIBOUTRY Louis	Géophysique
LOUP Jean	Géographie
Mle LUTZ Elisabeth	Mathématiques Pures
MALGRANGE Bernard	Mathématiques Pures
MALINAS Yves	Clinique obstétricale
MARTIN-NOEL Pierre	Seméiologie médicale
MASSEPORT Jean	Géographie
MAZARE Yves	Clinique médicale A
MICHEL Robert	Minéralogie et Pétrographie
MOURIQUAND Claude	Histologie
MOUSSA André	Chimie nucléaire
NEEL Louis	Physique du Solide
OZENDA Paul	Botanique
PAUTHENET René	Electrotechnique
PAYAN Jean-Jacques	Mathématiques Pures
PEBAY-PEYROULA Jean-Claude	Physique
PERRET René	Servomécanismes
PILLET Emile	Physique industrielle
RASSAT André	Chimie systématique
RENARD Michel	Thermodynamique
REULOS René	Physique industrielle
RINALDI Renaud	Physique
ROGET Jean	Clinique de pédiatrie et de puériculture
SANTON Lucien	Mécanique
SEIGNEURIN Raymond	Microbiologie et Hygiène
SENGEL Philippe	Zoologie
SILBERT Robert	Mécanique des fluides
SOUTIF Michel	Physique générale
TANCHE Maurice	Physiologie
TRAYNARD Philippe	Chimie générale
VAILLAND François	Zoologie
VAUQUOIS Bernard	Calcul électronique
Mme VERAIN Alice	Pharmacie galénique
M. VERAIN André	Physique
Mme VEYRET Germaine	Géographie
MM. VEYRET Paul	Géographie
VIGNAIS Pierre	Biochimie médicale
YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM. BULLEMER Bernhard	Physique
RADHAKRISHNA Pidatala	Thermodynamique

PROFESSEURS SANS CHAIRE

MM. AUBERT Guy	Physique
BEAUDOING André	Pédiatrie
BERTRANDIAS Jean-Paul	Mathématiques Appliquées
BIARES Jean-Pierre	Mécanique
BONNETAIN Lucien	Chimie minérale
Mme BONNIER Jane	Chimie générale
MM. CARLIER Georges	Biologie végétale
COHEN Joseph	Electrotechnique
COUMES André	Radioélectricité
DEPASSEL Roger	Mécanique des Fluides
DEPORTES Charles	Chimie minérale
DESRE Pierre	Métallurgie
DOLIQUE Jean-Michel	Physique des Plasmas
GAUTHIER Yves	Sciences biologiques
GEINDRE Michel	Electroradiologie
GIDON Paul	Géologie et Minéralogie
GLENAT René	Chimie organique
HACQUES Gérard	Calcul numérique
JANIN Bernard	Géographie
Mme KAHANE Josette	Physique
MM. MULLER Jean-Michel	Thérapeutique
PERRIAUX Jean-Jacques	Géologie et minéralogie
POULOUJADOFF Michel	Electrotechnique
REBECQ Jacques	Biologie (CUS)
REVOL Michel	Urologie
REYMOND Jean-Charles	Chirurgie générale
ROBERT André	Chimie papetière
SARRAZIN Roger	Anatomie et chirurgie
SARROT-REYNAULD Jean	Géologie
SIBILLE Robert	Construction Mécanique
SIROT Louis	Chirurgie générale
Mme SOUTIF Jeanne	Physique générale
M. VALENTIN Jacques	Physique nucléaire

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

Mle AGNIUS-DELORD Claudine	Physique pharmaceutique
ALARY Josette	Chimie analytique
MM. AMBLARD Pierre	Dermatologie
AMBROISE-THOMAS Pierre	Parasitologie
ARMAND Yves	Chimie
BEGUIN Claude	Chimie organique
BELORIZKY Elie	Physique
BENZAKEN Claude	Mathématiques Appliquées
Mme BERTRANDIAS Françoise	Mathématiques Pures
MM. BLIMAN Samuel	Electronique (EIE)
BLOCH Daniel	Electrotechnique
Mme BOUCHE Liane	Mathématiques (CUS)
MM. BOUCHET Yves	Anatomie
BOUSSARD Jean-Claude	Mathématiques Appliquées
BOUVARD Maurice	Mécanique des Fluides
BRIERE Georges	Physique expérimentale
BRODEAU François	Mathématiques (IUT B)
BRUGEL Lucien	Energétique
BUISSON Roger	Physique
BUTEL Jean	Orthopédie
CHAMBAZ Edmond	Biochimie médicale
CHAMPETIER Jean	Anatomie et organogénèse

MM. CHIAVERINA Jean	Biologie appliquée (EFP)
CHIBON Pierre	Biologie animale
COHEN-ADDAD Jean-Pierre	Spectrométrie physique
COLOMB Maurice	Biochimie médicale
CONTE René	Physique
CROUZET Guy	Radiologie
DURAND Francis	Métallurgie
DUSSAUD René	Mathématiques (CUS)
Mme ETERRADOSSI Jacqueline	Physiologie
MM. FAURE Jacques	Médecine légale
GAVEND Michel	Pharmacologie
GENSAC Pierre	Botanique
GERMAIN Jean-Pierre	Mécanique
GIDON Maurice	Géologie
GRIFFITHS Michaël	Mathématiques Appliquées
GROULADE Joseph	Biochimie médicale
HOLLARD Daniel	Hématologie
HUGONOT Robert	Hygiène et Médecine préventive
IDELMAN Simon	Physiologie animale
IVANES Marcel	Electricité
JALBERT Pierre	Histologie
JOLY Jean-René	Mathématiques Pures
JOUBERT Jean-Claude	Physique du Solide
JULLIEN Pierre	Mathématiques Pures
KAHANE André	Physique générale
KUHN Gérard	Physique
Mme LAJZEROWICZ Jeannine	Physique
MM. LAJZEROWICZ Joseph	Physique
LANCIA Roland	Physique atomique
LE JUNTER Noël	Electronique
LEROY Philippe	Mathématiques
LOISEAUX Jean-Marie	Physique Nucléaire
LONGEQUEUE Jean-Pierre	Physique Nucléaire
LUU DUC Cuong	Chimie Organique
MACHE Régis	Physiologie végétale
MAGNIN Robert	Hygiène et Médecine préventive
MARECHAL Jean	Mécanique
MARTIN-BOUYER Michel	Chimie (CUS)
MAYNARD Roger	Physique du Solide
MICOUD Max	Maladies infectieuses
MOREAU René	Hydraulique (INP)
NEGRE Robert	Mécanique
PARAMELLE Bernard	Pneumologie
PECCOUD François	Analyse (IUT B)
PEFFEN René	Métallurgie
PELMONT Jean	Physiologie animale
PERRET Jean	Neurologie
PERRIN Louis	Pathologie expérimentale
PFISTER Jean-Claude	Physique du Solide
PHELIP Xavier	Rhumatologie
Mle PIERY Yvette	Biologie animale
MM. RACHAIL Michel	Médecine interne
RACINET Claude	Gynécologie et obstétrique
RICHARD Lucien	Botanique
Mme RINAUDO Marguerite	Chimie macromoléculaire
MM. ROMIER Guy	Mathématiques (IUT B)
ROUGEMONT (DE) Jacques	Neuro-Chirurgie
STIEGLITZ Paul	Anesthésiologie

MM. STOEBNER Pierre	Anatomie pathologique
VAN CUTSEM Bernard	Mathématiques Appliquées
VEILLON Gérard	Mathématiques Appliquées (INP)
VIALON Pierre	Géologie
VOOG Robert	Médecine interne
VROUSSOS Constantin	Radiologie
ZADWORNY François	Electronique

MAITRES DE CONFERENCES ASSOCIES

MM. BOUDOURIS Georges	Radioélectricité
CHEEKE John	Thermodynamique
GOLDSCHMIDT Hubert	Mathématiques
YACOUD Mahmoud	Médecine légale

CHARGES DE FONCTIONS DE MATIRES DE CONFERENCES

Mme BERIEL Hélène	Physiologie
Mme RENAUDET Jacqueline	Microbiologie

Fait le 8 MARS 1972.

Je tiens à remercier,

Monsieur le Professeur P.NAMIAN qui a bien voulu me faire l'honneur de présider le Jury,

Monsieur le Professeur L.BOLLIET qui a toujours montré la plus grande bienveillance pour mon travail,

Monsieur M.BELLOT qui m'a guidé avec amitié dans mes Etudes Universitaires, et qui n'a pas ménagé ses conseils ni son temps pour me permettre de mener à bien cette Thèse.

Monsieur C.HANS qui, comme instigateur du projet CP+/CMS+, a bien voulu me permettre de réaliser cette Thèse sur ce sujet; et, comme ami, m'a aidé et conseillé tout au long de mes travaux.

Monsieur P.ARNAUD, Directeur du C.U.E.F.A., qui, par son étroite collaboration avec le C.N.A.M. a rendu possible cette Thèse.

Je voudrais aussi remercier,

Tous mes collègues de l'Equipe CP/CMS de l'IMAG et du Centre Scientifique IBM, Madame L.SIRET, Messieurs J.P.DUPUY, J.GUILLOU, P.LEFEBVRE, M.REY et N.T.THI, qui ne m'ont pas ménagé leurs suggestions et leurs critiques, et encore moins leur aide;

Le Service Dactylographique qui a eu la redoutable charge de frapper ce texte.

Je voudrais enfin remercier le Service de Tirage de l'IMAG qui en a assuré l'impression.

CHAPITRE I

INTRODUCTION

Au cours des deux dernières années, un certain nombre de systèmes "Time-Sharing" ont été réalisés par différents constructeurs ou laboratoires universitaires. Ces systèmes sont en fonctionnement dans plusieurs centres privés ou publics et, de par leur construction, s'orientent vers des applications spécifiques ou, au contraire, essaient d'être plus généraux. L'un de ces systèmes : CP-67/CMS, constitue, à lui seul, une classe particulière et est utilisé, depuis 1968, à l'Institut de Mathématiques Appliquées de GRENOBLE (I.M.A.G.). Nos différents travaux ont été dirigés vers la modification de ce système, pour améliorer son fonctionnement dans le contexte particulier de son utilisation universitaire et pluri-disciplinaire.

Le système CP-67/CMS est constitué de deux parties :

- Le programme de contrôle CP-67
- Le système conversationnel CMS.

CP-67 (en abrégé CP), réalise la fonction "partage de temps", qui permet à de nombreux utilisateurs d'accéder simultanément au calculateur.

CMS fournit la fonction "conversation" qui permet, à un seul utilisateur, de contrôler son travail à partir d'un terminal du type "machine

à écrire". En effet, on peut utiliser CMS seul sur une machine de la série IBM 360. Dans ce cas, la totalité du calculateur est réservée à un seul utilisateur qui dialogue avec CMS, à l'aide de la console du pupitre de la machine (cette formule est utilisée, par exemple, au Centre Scientifique IBM de Grenoble sur un 360-40).

Par contre, si CP est utilisé seul, un ou plusieurs autres systèmes doivent être choisis pour assurer la fonction "conversation" ou la fonction "Production". En effet, CP ne fournit que la seule fonction "Partage de temps".

Pour réaliser cet aspect "Multi système", CP crée le concept de "Machine virtuelle". De façon plus générale, on utilise CMS sous CP concurremment avec d'autres associations telles que DOS sous CP, OS sous CP etc.... Grâce à ce concept de "Machine Virtuelle", chacun de ces systèmes est orienté plus particulièrement, soit vers la fonction "conversation", soit vers la fonction "production".

Appelons "élément physique", un constituant de la configuration d'une machine réelle (unité centrale, mémoire ou unités d'entrées-sorties).

Appelons "élément virtuel", la simulation "SOFTWARE" de son équivalent physique (unité centrale, mémoire, unités d'entrées-sorties).

Par définition, une "machine virtuelle" est un ensemble d'éléments virtuels, éléments qui définissent sa configuration.

Le programme de contrôle CP, actif sur une machine IBM 360/67, est un générateur de machines virtuelles. Il doit, pour chacune d'elles, simuler les éléments virtuels qui la composent en partageant les éléments physiques dont il dispose. CP réalise le partage de l'unité centrale, en l'affectant tour à tour à chacune des machines virtuelles, pour un court laps de temps.

Il réalise le partage de la mémoire réelle à l'aide du dispositif de traduction dynamique des adresses propre à l'IBM 360-67. Quant au partage des unités d'entrées-sorties, il est réalisé de trois façons, selon qu'il s'agit d'unités à accès direct (disques), d'unités lentes (lecteurs, perforateurs, imprimantes) et d'unités non partageables simultanément (bandes, écrans cathodiques, etc...)

Les éléments virtuels disques, appelés aussi mini-disques, sont des sous-ensembles d'éléments réels (un disque réel est découpé en plusieurs mini-disques).

Les éléments virtuels lents, sont simulés grâce à un mécanisme d'entrée-sortie différée, plus connu sous le nom de SPOOLING.

Les éléments physiques, non partageables simultanément, sont affectés manuellement, par l'opérateur de CP, aux machines virtuelles qui le demandent. Dans ce cas, il y a identité entre l'élément virtuel et l'élément physique.

Un aspect fondamental du principe des machines virtuelles est le suivant : tout programme écrit sur une machine virtuelle, de configuration donnée, peut aussi fonctionner sans modification sur une machine réelle de même configuration. Cette propriété est capitale lorsque l'on met au point un nouveau système ou lorsque l'on modifie un système déjà réalisé : le système à mettre au point est testé sur une machine virtuelle, ce qui évite d'immobiliser, pendant les tests, le calculateur auquel il est destiné (les ingénieurs et programmeurs système peuvent enfin travailler pendant les heures ouvrables!).

Sauf cas particulier, la mise au point de systèmes, ne justifie pas à elle seule, l'utilisation complète d'un calculateur tel qu'un IBM 360-67. De ce fait, il est nécessaire dans le cadre de CP-67, d'assurer aussi le service d'applications plus classiques (mathématiques, physiques,gestions), et ceci avec un rendement convenable. Il convient, toutefois, de faire quelques remarques de fonctionnement inhérent au principe des machines virtuelles, lesquelles activent

des systèmes particuliers :

Si par exemple, on désire assurer un service à l'aide d'un système tel qu'OS 360 MVT, on assiste à un double effort de multiprogrammation par OS d'une part, et CP d'autre part.

De même, les deux systèmes essaient d'optimiser les entrées-sorties. Ces efforts louables en eux-mêmes s'ils sont pris séparément, deviennent indésirables s'ils sont simultanés. En effet, les deux mécanismes s'ignorent et, de ce fait, deviennent antagonistes. Bien que le fonctionnement d'OS soit convenablement assuré par CP, le rendement général est déplorable. Nous avons pu en faire l'expérience à l'Institut de Mathématiques Appliquées de Grenoble. D'une façon générale il faut admettre que CP, tel qu'il est actuellement, réalise parfaitement sa tâche de simulation de machines réelles. Il suffit, pour s'en convaincre, d'examiner la diversité des systèmes qui ont pu être activés avec succès sur une machine réelle, c'est-à-dire sans défaut de fonctionnement :

TSS 360, MTS, OS(PCP, MVT ...), DOS, APL, CMS ...etc... et CP lui-même.

On peut noter que le rendement est d'autant plus faible que le système, actif sur une machine virtuelle, utilise toutes les possibilités de la machine. En effet, l'effort de simulation que doit fournir CP croit considérablement d'une part avec la complexité des entrées sorties qu'il doit simuler, et d'autre part avec le nombre d'instructions privilégiées émises par la machine virtuelle.

Si l'on admet que le rendement de l'installation est :

$$TUCV \;/\; TUCV + TCP \;,$$

où TUCV est le temps d'unité centrale donné aux différentes machines virtuelles et TCP le temps d'unité centrale défini par CP pour les simuler.

De façon évidente, on voit que plus le temps CP croit, plus le rendement décroit. Nous pouvons donc affirmer que, dans l'état actuel de CP,

plus un système actif sur une machine virtuelle est compliqué, plus le rendement global diminue. Seul CMS est suffisamment simple (ce qui ne veut pas dire simpliste) pour assurer à l'installation un rendement convenable.

CP porte en lui-même la contradiction suivante :

Le concept de machine virtuelle, qualité de base de CP, pour la mise au point ou le développement de système, devient un défaut lorsqu'il s'agit d'assurer un rendement convenable pour une installation.

A l'Institut de Mathématiques Appliquées de Grenoble, nous nous sommes fixés comme objectif d'augmenter le rendement général de l'installation lorsque CP-67 est actif, ceci pour fournir un meilleur service aux utilisateurs que nous avons appelés "classiques". Actuellement ces personnes utilisent le système CMS, lequel possède des qualités intéressantes grâce à ses mécanismes conversationnels. Toutefois, le système CMS, pour la quasi totalité des utilisateurs, est un moyen de réaliser leurs travaux, et en conséquence, le fait que ce système soit actif sur une machine virtuelle n'est pas fondamental.

Nous avons ainsi été amenés à définir une entité, équivalente à une machine virtuelle, qui possède les mécanismes nécessaires au fonctionnement de CMS, tout en imposant le minimum de charge à CP. Cette nouvelle entité est la "machine logique" ou "environnement".

Nous appelons "environnement" une entité ayant fonctionnellement les caractéristiques d'un ordinateur et dont la configuration est formée d'éléments virtuels et logiques. On peut dire aussi qu'un environnement est analogue à une machine virtuelle à laquelle nous ajoutons au moins un élément logique.

Nous appelons "élément logique", un constituant de la configuration d'un environnement qui ne possède pas d'équivalent physique. De ce fait, il introduit une extension des possibilités de CP, qui devient alors CP+. Il

est important d'insister sur le terme d'extension, qui implique que CP+ conserve la possibilité de générer des machines virtuelles, avec en plus la possibilité de générer des environnements.

Ces éléments logiques sont définis de toute pièce, soit pour se substituer à des éléments virtuels, soit pour répondre à de nouveaux besoins fonctionnels. Notre souci majeur a été de minimiser la charge de CP+, ce qui nous oblige à construire ces nouveaux éléments de telle manière, qu'ils utilisent directement les fonctions élémentaires de CP, à savoir : mécanisme de pagination, gestion des entrées-sorties etc...

De plus, ces éléments logiques, vus du côté environnement, sont plus simples à utiliser que les éléments virtuels vus du côté machine virtuelle.

Cette solution nous permet raisonnablement d'affirmer que le temps pris par CP diminuera; en effet, nous avons énoncé que la tâche de CP est de satisfaire les demandes provenant des machines virtuelles. Les formes que prennent ces demandes sont directement liées à la structure de chaque élément virtuel, qui, rappelons-le, est la réplique d'un élément réel. Ceci conduit bien à minimiser le temps passé dans CP dont la tâche principale de simulation de machine virtuelle est de mettre en forme les demandes des machines virtuelles qui ne sont pas directement utilisables par les fonctions élémentaires. Ces demandes ne sont pas directement exécutables par CP, et ne sont pas non plus très favorables à une simulation. En conséquence, CP doit effectuer une transformation, souvent complexe, de chacune d'elles, pour qu'elle puisse être transmise aux fonctions élémentaires. Par opposition, les formes que prennent les demandes liées aux éléments logiques, peuvent s'adapter au mieux à la structure intime de CP, qui ainsi, les utilise directement. De plus, ces éléments logiques, vus du côté d'un environnement sont plus simples à utiliser que les éléments virtuels vus du côté d'une machine virtuelle.

De manière plus globale, la notion d'élément logique généralise et étend la notion d'élément virtuel.

Si nous définissons une mémoire comme étant un ensemble d'éléments ordonnés appelés pages, une fonction (la seule de ce type actuellement dans CP) fait correspondre à chaque élément d'une mémoire virtuelle, un élément de la mémoire réelle.

Si nous considérons, alors, un élément logique comme une suite d'entités élémentaires virtuelles, et si nous réalisons, dans CP+, une fonction qui, à chaque entité élémentaire virtuelle fait correspondre une entité élémentaire de même type sur un support réel, nous avons créé le concept d'espace virtuel ébauché par CP pour la mémoire virtuelle.

Si maintenant nous étudions le système privilégié CMS, nous constatons qu'il ne possède pas tous les mécanismes nécessaires pour tirer profit de la notion d'élément logique, donc d'environnement. Nous le modifierons en conséquence, et en augmentant ses possibilités, il deviendra CMS+.

Nos différents travaux nous ont conduits à faire un certain nombre d'expériences, et à réaliser un certain nombre d'applications sur les systèmes CP et CMS. Ces diverses activités ont pour but de justifier les modifications que nous proposons, et de quantifier les avantages que nous pouvons attendre de la mise en oeuvre des systèmes CP+ et CMS+.

Dans la suite de la thèse, nous ne donnerons pas la description externe des systèmes CP et CMS; nous renvoyons pour cela le lecteur à la référence C5. Par contre, les mécanismes fondamentaux, soit de l'IBM 360/67, soit de CP, soit de CMS nécessaires à une bonne compréhension, seront détaillés dans le chapitre II. Nous présenterons également la complexité des problèmes posés à CP par la simulation des machines virtuelles. Nous étudierons sur un exemple (compilation des programmes canaux) tout l'intérêt qu'il y a à supprimer cette simulation chaque fois qu'elle n'est pas indispensable. Nous examinerons la gestion de fichier de CMS, qui est la clé de ce système; nous en donnerons un aperçu pour pouvoir, par la suite, expliquer les modifications qu'elle a subies.

Dans le Chapitre III, nous exposerons les divers essais de réalisations patiques que nous avons pu faire sur la transformation de CMS en environnement, et les améliorations ainsi apportées au système en exploitation permanente à l'IMAG.

Dans le Chapitre IV, nous présenterons les systèmes CP+ et CMS+ dans l'état actuel de leur définition; en insistant sur les nouveautés qu'ils peuvent comporter et qui, rappelons-le, ont pour but d'améliorer l'efficacité globale de l'installation.

CHAPITRE 2

CERTAINS MECANISMES INTERNES DE CP ET DE CMS

2.1. MEMOIRES VIRTUELLES [Ref D2]

2.1.1. Définition. Transformation des adresses.

Les applications sur ordinateur, donc les programmes qui les traitent, sont devenues de plus en plus complexes, ce qui implique en particulier, l'utilisation de vastes mémoires. Toutefois,la mémoire rapide reste chère; aussi pour essayer d'en minimiser le coût, une solution possible consiste à utiliser des mémoires à plusieurs niveaux de hiérarchie. Dans la plupart des cas, on se limitera à deux niveaux :

- La mémoire principale, directement accessible par l'unité centrale.
- La mémoire secondaire, située sur une unité périphérique rapide (en général tambour ou disques) dont le contenu doit être d'abord amené en mémoire principale, avant d'être utilisable par l'unité de traitement.

Le problème de la gestion de la hiérarchie mémoire est de

définir à tout instant, comment l'information (instructions et données), doit être répartie entre les différents niveaux :

La solution classique consiste à laisser à l'utilisateur le soin de résoudre ce problème de nature système. Pour ce faire, le système lui attribue quelques centaines de Kilo-Octets de mémoire principale, pour la durée de l'exécution de son travail.

Si cette zone se révèle insuffisante, il peut d'une part, créer des fichiers intermédiaires pour y placer ses données et d'autre part amener son programme par morceaux successifs en mémoire principale depuis un support externe. Pour cela, l'utilisateur peut employer soit des techniques de recouvrement statiques (OVERLAY), soit une gestion dynamique telle qu'il en existe, par exemple dans l'operating system OS/360 (Macro-instructions du type LOAD, DELETE, LINK, XCTL, etc...). Rien ne prouve d'ailleurs que, confronté à ces techniques, l'utilisateur emploie au mieux la zone mémoire qui lui a été impartie.

La solution automatique de gestion de la hiérarchie mémoire, repose sur la constatation suivante :

- le concept d'adresse est ambigu

  D'une part les adresses servent à repérer les mots (au sens de "la plus petite unité adressable") qui sont utilisés dans un programme; Les adresses sont donc des NOMS (l'adressage symbolique le prouve bien). L'espace ainsi défini est "L'espace des Noms" ou "Espace utilisateur"; il est lié à la logique du programme.

  D'autre part, les adresses indiquent l'emplacement physique où résident, dans la machine, les mots utilisés. C'est "L'espace mémoire" ou "espace réel"; il est lié à la logique du calculateur.

Dans une machine à adressage classique, cette ambiguïté subsiste : En effet on peut remarquer que l'implantation d'un programme dans l'espace mémoire établit un lien rigide avec l'espace des noms;un programme reste physiquement au même emplacement pendant toute son exécution.

Dans une machine à adressage automatique, l'espace des noms est aussi appelé "MEMOIRE VIRTUELLE"; par opposition à l'espace mémoire (réel ou physique). En conséquence, nous avons distingué la mémoire réelle et la mémoire virtuelle.

De ce fait, rien ne s'oppose à ce qu'elles aient des caractéristiques différentes : entre autres, la mémoire virtuelle peut avoir :

- une taille inférieure, égale ou supérieure à celle de la mémoire réelle.

- un lieu de résidence, distinct de la mémoire principale (ou réelle), qui englobe tous les niveaux de la hiérarchie mémoire, que ce soit la mémoire principale (réelle) ou les mémoires auxiliaires ou les deux à la fois.

Ces différentes propriétés impliquent toutefois qu'il existe une fonction de transformation qui applique l'espace virtuel dans l'espace réel et qui, s'effectue dynamiquement pendant l'exécution du programme.

Ce mécanisme présente un certain nombre d'avantages, en particulier :

a) Puisqu'il n'y a pas de lien entre l'espace utilisateur et l'espace réel, toute modification de la taille de la mémoire réelle ou de la mémoire virtuelle n'implique aucun effort supplémentaire de programmation. Ainsi, en cas de panne d'une partie de la mémoire réelle, tous les travaux peuvent s'exécuter de la même manière avec comme seule conséquence une perte

de performance.

b) Un programme peut être déplacé à l'intérieur des différents niveaux de la hiérarchie, sans avoir à se préoccuper de modifier les valeurs des constantes de type "Adresse".

c) Un programme peut facilement croître en taille au cours de son élaboration, sans se heurter aux limites imposées par la dimension de la mémoire réelle affectée.

d) Le système d'exploitation, plus précisément l'algorithme qui optimise l'utilisation de la mémoire réelle, effectue un contrôle dynamique permanent pour déterminer la taille optimale de la zone réelle à attribuer à un programme quelconque.

2.1.2. Transformation des adresses.

2.1.2.1. Représentations de la transformation

Chaque fois que l'unité centrale décode une instruction située dans la mémoire virtuelle, la ou les adresses qu'elle calcule, sont dans l'espace des noms. Or, il est évident que l'unité de traitement, ne peut agir que dans l'espace réel et qu'en conséquence, elle doit associer à toute adresse virtuelle v, son image r = f(v) en supposant que l'élément placé en v de l'espace virtuel V réside en r de l'espace réel R(Voir fig.2.1.2(a)).

Si cette dernière hypothèse n'est pas remplie, c'est-à-dire si v'' n'a pas de transformé dans R, le mécanisme est mis en défaut; un signal d'"élément manquant" est généré et le traitement du programme en cours est alors interrompu le temps nécessaire au transfert de l'élément v" en mémoire réelle. Dans ce cas, il se peut qu'il soit nécessaire, au préalable, de chasser l'élément r'= f(v') pour procurer une place à celui d'adresse v".

L'instantané de la figure 2.1.2(b) est pris alors qu'un programme résidant dans V est actif. Pour plus de clarté l'espace V est limité à 8 éléments et l'espace R à 4 éléments. La figure montre que l'emplacement virtuel 0 est rangé à l'emplacement réel 3 (r_3) de même v_2 est en r_1 et v_6 en r_2. Par contre les emplacements virtuels 1,3,4,5,7, ne sont pas en mémoire réelle R, au moment de l'instantané. Notons que l'emplacement r_o n'est le transformé d'aucun élément de V.

L'application de V dans R peut être défini par une table (ensemble de mots réels), dont on présente deux variantes suivant l'espace qu'elles représentent : V ou R.

Table de type I : Représentation de V

Elle possède un nombre d'entrées proportionnel à la taille de la mémoire virtuelle (fig.2.1.2.(c)). La k-ième entrée indique si la k-ième adresse de l'espace virtuel à un transformé dans l'espace R : Si cela est, elle fournit la valeur de ce transformé sous forme du numéro de l'élément r associé. Ainsi à V_o est associé r_3, à v_2 est associé r_1 etc... Par contre, si le transformé n'existe pas dans R, l'entrée correspondante est marquée invalide (* sur la figure).

En fonctionnement normal, l'unité centrale possède une adresse virtuelle v et souhaite connaître le transformé r = f(v). La table fournit le résultat en une seule lecture; par contre, les mémoires virtuelles pouvant être de très grande taille, le nombre d'éléments de la table peut être aussi très important et soulever un sérieux problème d'occupation de place en mémoire réelle.

Table de type II,Représentation de R.

Elle possède un nombre d'entrées proportionnel à la taille de

la mémoire réelle R : *fig.2.1.2.(d)*. A chaque adresse r de R, elle associe une valeur v, adresse dans V, telle que r = f(v), si le transformé existe.

Cette table est, en général, moins encombrante que la précédente. Par contre, la recherche du transformé est nettement plus compliquée. En effet, pour cette table, r est implicite (numéro de l'entrée), mais la seule valeur donnée est v. Il faut donc rechercher séquentiellement les différentes entrées de la table pour trouver v et en déduire r = f(v). Si la recherche n'a pas donné satisfaction, c'est que v n'a pas de transformé dans R et la seule table de type II ne permet pas de lui en trouver un.

Pour que la recherche dans une table de ce type, soit faite avec des performances satisfaisantes, il est nécessaire que toutes ses entrées puissent être examinées simultanément.

Ceci est réalisé en pratique par une mémoire associative. Les couples (v,r) sont rangés dans les éléments de la mémoire associative qui, de par son principe, fournit la valeur r cherchée lorsqu'on lui donne la clé v comme paramètre, ou à défaut un signal d'élément "non-trouvé".

2.1.2.2. Nécessité du découpage de V et R en blocs.

Les deux méthodes de représentation de la transformation f que l'on vient de présenter, ne sont pas directement applicables dans la réalité. En effet, ces tables contiennent autant d'entrées qu'il existe d'adresses dans l'espace qu'elles représentent. On est donc amené, pour réduire leur taille, à grouper l'information par blocs d'adresses.

Chaque bloc est défini comme une suite d'adresses contigües quel que soit l'espace en cause. Les entrées dans les tables (quel qu'en soit

le type), contiennent maintenant les numéros de blocs. Ce qui diminue considérablement la dimension des tables. Deux cas se présentent selon que la taille des blocs est variable ou fixe :

Blocs de taille variable : segmentation.

Dans cette méthode de gestion de la mémoire virtuelle, on organise l'espace des noms en blocs de tailles arbitraires, appelés "segments".

L'espace virtuel ainsi structuré se présente comme une collection de segments repérés par leurs noms. Dans un tel espace, le programmeur adresse une information par un couple v = (S,d) où S est le nom du segment et d le déplacement de la référence à l'intérieur du segment.

Citons quelques avantages de cette méthode :

- La taille d'un segment (contenant par exemple des données), peut varier au cours de son utilisation.
- Chaque segment peut représenter une entité logique et comme telle :
  - il est possible de lui associer un privilège d'accès (lecture seulement, lecture écriture, etc...), ce qui accroit la protection.
  - Il peut être partagé, c'est-à-dire connu sous des noms distincts dans différents espaces disjoints.

Un tel mécanisme peut se réaliser de la façon suivante : chaque segment est chargé entièrement dans une zone contigüe de la mémoire réelle. L'adresse réelle de début du segment, constitue son adresse de base. Cette dernière est contenue dans l'entrée correspondante de la table des segments

(Table de type I); elle contient également sa longueur, ainsi que les privilèges d'accès (*fig.2.1.2.(e)*). Chaque segment ayant une longueur L qui lui est propre, celle-ci doit être conservée dans la Table, afin de pouvoir vérifier que d ≤ L. Si d > L il y a violation des règles de segmentation; une interruption pour faute d'adressage est générée. L'absence ou la présence d'une valeur de la base B indique, suivant le cas, l'absence ou la présence du segment en mémoire rapide. (Dans le cas où B est absent, une interruption pour segment manquant est générée). L'un des problèmes fondamentaux posé par la segmentation vient de la nécessité de trouver, en mémoire réelle, des régions, d'un seul tenant, de tailles différentes les unes des autres. Ceci contribue à donner rapidement un aspect de "Gruyère" à la mémoire centrale. En effet les zones utilisées peuvent être séparées les unes des autres par des zones inemployables, parce que trop petites. Ce problème est identique à celui rencontré dans les systèmes classiques (qui n'utilisent pas la mémoire virtuelle) qui acceptent un nombre dynamiquement variable de tâches (OS/MVT par exemple), il conduit à un gaspillage de mémoire réelle qui atteint fréquemment 25 %.

Blocs de longueur fixe : Pagination.

Les espaces réels et virtuels sont découpés en blocs de taille fixe, appelés "Pages". Une page est alors la plus petite unité d'information qui puisse être déplacée d'un niveau de la hiérarchie mémoire à un autre, contrairement à la segmentation. La gestion de la mémoire ainsi découpée en blocs homogènes, ne pose pas de problèmes particuliers. Il suffit en effet de trouver un bloc libre, et non pas: "Une zone contigüe suffisamment vaste pour contenir le segment". Par contre la pagination qui est un découpage arbitraire d'un espace logique, ne peut en altérer la linéarité. Les avantages liés à la structure logique des segments, tels que :

- Modules de données variant dynamiquement en taille.
- Privilèges d'accès attachés à un segment (Entité logique).

- Partage des segments entre différents utilisateurs,

ne sont plus aussi évidents à réaliser.

L'application pratique de la transformation r = f(v) pour un tel découpage, peut être la suivante : (*fig.2.1.2.(f)*).

Soit "t" la taille d'une page et "v" une adresse virtuelle; on calcule

$$p = [\frac{v}{t}] \quad \text{(division entière)}$$

et

$$d = v - pt \quad \text{(reste de la division)}$$

"p" est le numéro de page virtuelle et "d" le déplacement à l'intérieur de celle-ci. La transformé p' = f(p) est alors obtenu grâce à une table de Type I et

$$r = p' * t + d$$

Dans les machines à arithmétique binaire, le calcul de l'adresse réelle r à partir de v est simplifié lorsque la taille t d'une page est une puissance de 2. En effet évaluer r = p't + d revient en fait à concaténer les deux chaines de bits (numéro de page, déplacement dans la page).

Combinaison des deux méthodes : Segmentation et Pagination

Chacune des méthodes précédentes a ses propriétés qui, par certains côtés, sont complémentaires. Tout naturellement, les deux méthodes : Pagination et Segmentation ont été combinées pour donner un seul mécanisme d'adressage. Cette technique nécessite la présence des deux tables précédemment définies (Table de segments et Table de pages).

La partie de l'entrée de la table des segments donnant la base a changé de signification et par un mécanisme analogue à l'adressage indirect, donne l'adresse d'une table de page qui décrit le segment découpé en pages.

La partie donnant la longueur contient le nombre de pages composant le segment. La longueur d'un segment doit donc être un multiple de la taille d'une page. *(Fig.2.1.3.(h))*.

Cette organisation présente de nombreux avantages par rapport à chaque méthode prise séparément.

Si la partie B d'une entrée de la table des segments est invalide, cela signifie que la table des pages associée n'est pas, en mémoire réelle et une interruption de type adéquat est générée.

Cette facilité permet de réduire la place occupée par les tables de pages qui peuvent ainsi n'être amenées depuis un support externe en mémoire réelle que lorsqu'elles sont indispensables.

Si la partie p' d'une entrée de la table des pages est invalide, cela signifie que la page associée n'est pas en mémoire réelle et une interruption pour faute de page est générée.

Il n'est plus nécessaire qu'un segment soit entièrement en mémoire à un instant donné.

Grâce à l'existence de la table des pages associée au segment, celui-ci n'est plus obligé d'occuper une zone contigüe de mémoire réelle.

Cette organisation constitue, en particulier, la base de la traduction dynamique d'adresse de l'IBM 360-67, machine que nous allons étudier.

2.1.3. L'IBM 360/67. [Ref.I3]

Le calculateur 360/67, reste compatible avec la série 360, mais possède certains dispositifs spéciaux particuliers additionnels :

- 16 registres de contrôle supplémentaires (notés CR_o à CR_{15}).
- une traduction dynamique des adresses (DAT).
- un mot d'état programme étendu (PSW étendu).
- des clés de protection mémoires étendues.
- des instructions supplémentaires qui permettent de gérer ces dispositifs.

Les autres caractéristiques de ce calculateur sont analogues à un IBM 360 standard.

2.1.3.1. Etude de la traduction des adresses.

La technique employée pour résoudre la transformation V → R est, ainsi qu'il l'a été dit en 2.1.2., celle de la segmentation et de la pagination. La taille d'une page est fixée à 4096 Octets (2^{12}).

L'adresse virtuelle donnée au dispositif de conversion possède 24 bits significatifs. L'espace V est donc limité à 16×10^6 octets.

Si le 360/67 est muni d'un dispositif spécial supplémentaire, cette adresse est portée à 32 bits significatifs (4.10^9 octets).

a) Le registre de contrôle 0. (segment table register)

Fig.2.1.3.(a).

Il contient l'adresse de la table des segments active ainsi que la longueur de cette table (sous une forme précisée à l'alinéa b).

b) Table des segments.

Il en existe une par mémoire virtuelle et elle réside en mémoire à une adresse multiple de 64 et elle est divisée en blocs de 16 entrées.

- Dans le cas de l'adressage virtuel à 24 bits, il n'existe qu'un seul bloc. (16 entrées).

- Dans le cas de l'adressage virtuel à 32 bits, les bits 0 à 7 du registre de contrôle 0 donnent le nombre de bloc - 1 (c'est-à-dire qu'au maximum nous aurons 256 blocs de 16 entrées).

Chaque entrée a 4 octets de long et contient l'adresse et le nombre d'entrées moins un de la table des pages associée :(*fig.2.1.3.(b)*). On peut noter que l'on a affecté une signification particulière au bit 31 de l'adresse de la table des pages, parce que celle-ci est alignée sur un demi mot (adresse paire).

c) Table des pages (*fig.2.1.3.(c)*)

Il en existe autant que d'entrées dans la table des segments, soit :

- 16 au maximum pour un adressage virtuel à 24 bits.

- 4096 au maximum pour un adressage virtuel à 32 bits.

Une table de page comprend un nombre variable d'entrées (au maximum 256). Ce nombre est indiqué dans la table des segments. Chaque entrée, longue d'un demi mot, donne le numéro de la page en mémoire réelle et la présence ou l'absence de la page en mémoire réelle est indiquée par un bit particulier de ce 1/2 mot.

d) Calcul effectif d'une adresse (cas de l'adressage virtuel à 24 bits) *fig.2.1.3.(d).*

Les 24 bits de l'adresse virtuelle sont découpés en trois groupes :

. Les 4 bits de poids fort (8-11) constituent le numéro de segment.

. Les 8 bits suivants (12-19)constituent le numéro de page.

. Les 12 bits de poids faible (20-31) donnent le déplacement à l'intérieur de la page.

Le numéro de segment est multiplié par quatre et ajouté au contenu du registre de contrôle 0. On obtient ainsi l'adresse d'une entrée dans la table des segments.

Cette entrée donne, alors, l'adresse de la table des pages associée à laquelle il suffit d'ajouter le numéro de page (après multiplication par deux) pour repérer une entrée dans la table des pages. Cette dernière donne enfin le numéro de la page en mémoire réelle; par concaténation de ce numéro avec le déplacement, on obtient l'adresse réelle recherchée.

Si l'une des opérations précédentes ne peut se réaliser

- soit, lorsque le bit 31 de l'entrée dans la table des segments vaut 1 (cas 1),

- soit, lorsque le bit 12 de l'entrée dans la table des pages vaut 1 (cas 2); une interruption programme se produit avec :

. le code 16 pour le cas 1
. le code 17 pour le cas 2

On peut donc remarquer que le mécanisme est tel que la table des segments doit toujours être en mémoire, mais que les tables des pages peuvent résider, si besoin est, sur un support externe.

e) Registres associatifs *(fig.2.1.3 (e))*

Ils sont au nombre de huit et contiennent, au fur et à mesure des besoins et de l'évolution des adresses intéressées, le doublet "adresse virtuelle"-"n° de page réelle" des 8 pages les plus récemment utilisées. L'adresse virtuelle est composée du couple "numéro de segment" et "numéro de pages"virtuels *(fig.2.1.3.(f))*. Ces registres spéciaux ont une très grande vitesse de résolution et sont les premiers utilisés à chaque nouvelle adresse virtuelle référencée. L'argument de recherche associative est alors numéro de segment et numéro de page virtuels et il est comparé en parallèle avec le contenu des 8 registres.

Si l'un d'entre eux contient l'équivalent de l'argument, le résultat de la comparaison donne le numéro de page réelle concernée.

Si par contre, aucun registre n'a un contenu satisfaisant, le calcul général de l'adresse énoncé au paragraphe précédent a lieu et de ce fait, l'un des registres (le moins récemment utilisé) est initialisé avec le nouveau doublet déterminé.

Nous ne détaillerons pas l'ensemble des mises à jour des différentes tables de segments et de pages.

2.1.3.2. Les deux mots d'état programme (PSW) *Fig.2.1.3.(d)*

Il existe deux types de PSW sur un 360/67 :

- Le PSW standard (celui des autres 360) dont nous ne parlerons pas
- le PSW étendu.

Ce registre de 64 bits contient un certain nombre de paramètres liés au fonctionnement en mode 67. Ces paramètres sont différents des paramètres d'un PSW standard; il convient donc de connaître le type de PSW actif à un instant donné. Pour cela, le bit 8 du registre de contrôle CR6 est mis à 1 quand le fonctionnement est en mode 67.

On notera que ce bit est automatiquement mis à 0.

Lors de la séquence IPL (par construction) et lors d'une interruption de type "machine-check" (de cette façon tout chargement initial utilise le mode de fonctionnement standard), on notera aussi que ce bit est modifiable par programme, à l'aide d'une instruction spéciale.

Description du PSW étendu

Sa longueur est de un double-mot, soit 64 bits numérotés de 0 à 63 de gauche à droite :

- bits 0 à 3 : inutilisés mais doivent être nuls
  sinon une interruption du type spécification sera générée.

- bit 4 : Si ce bit = 0, il indique l'adressage à 24 bits, si ce bit = 1, l'adressage est à 32 bits; mais la valeur 1 suppose que le hardware spécial pour adressage 32 bits est disponible, ce qui n'est pas le cas sur le 67 de Grenoble, et en conséquence le bit doit être nul.

- bit 5 : si ce bit = 0, il indique un fonctionnement sans conversion des adresses virtuelles en adresses réelles.
Si ce bit = 1, le fonctionnement est en mode 67 et la conversion d'adresse doit être appliquée.

- bit 6 : C'est un masque général pour les interruptions de type Entrée/Sortie; Si ce bit = 0, aucune interruption n'est acceptée.
Un masque contenant un bit par canal apparaît dans le registre de contrôle CR4; lorsqu'une interruption est générée par le canal 1, celle-ci n'est acceptée que si le bit ad-hoc de CR4 et le bit 6 du PSW ont la valeur 1.

- bit 7 : C'est un masque général pour les interruptions de type externe (c'est-à-dire dans le cas du 67 de Grenoble, les interruptions dûes à l'horloge ou les interruptions provoquées par le bouton 'externe' du pupitre). A ce masque sont associés 2 bits du registre de contrôle CR6 : le bit 24 correspond à l'horloge, le 25 au bouton "externe" du pupitre.
A titre d'exemple, une interruption 'horloge' n'est acceptable que si le bit 24 du CR6 et le bit 7 du PSW ont la valeur 1.

- bits 8 à 11 : Ils contiennent la clé de protection mémoire.

- bits 12 à 15 : Ces bits AMWP signifient :
  - A : la codification utilisée est l'ASCII.
  - M : c'est le bit masque de "machine check".
  - W : ce bit indique que la machine est en attente.
  - P : ce bit indique que la machine est en état problème.

- bits 16 à 17 : Ils définissent la longueur de l'instruction en cours.

- bits 18 à 19 : ces bits indiquent le code condition.

- bits 20 à 23 : ce sont les bits de masque du programme. Ils définissent les types d'interruption programme qui sont acceptés.

- bits 24 à 31 : ils sont inutilisés mais doivent être nuls.

- bits 32 à 63 : c'est l'adresse d'instruction; en fait, avec l'adresse à 24 bits (cas de Grenoble), les 8 bits de poids les plus forts sont inutilisés (bits 32 à 39).

Nous pouvons remarquer que le PSW étendu ne contient aucune information sur les codes des interruptions (contrairement au PSW standard). Ceux-ci sont rangés en mémoire et pour chaque type d'interruption on dispose de 2 octets qui lui sont propres : Les adresses sont les suivantes :

Adresse hexadécimale	Type de l'interruption
E - F	externe
10 - 11	SVC
12 - 13	programme
14 - 15	machine-check
16 - 17	entrée-sortie

2.1.3.3. Clés de protection mémoires étendues : *(fig.2.1.3 (g))*

Chaque clé de protection mémoire qui correspond à un bloc de 2K (il y en a donc deux par page) comprend sept bits :

0 à 3 : ils sont identiques à ceux du 360 standard; ils interviennent dans le PSW et le CAW pour la protection mémoire en écriture.

4 : si ce bit = 1, il indique la protection mémoire en lecture et écriture.

5 : dit de référence, prend la valeur 1 chaque fois que le bloc correspondant est référencé (en lecture ou écriture) par l'unité centrale ou un canal.

6 : dit de modification prend la valeur 1 chaque fois qu'une valeur est rangée (par l'unité centrale ou le canal) dans le bloc correspondant.

Rappelons que : Pour écrire dans un bloc, la clé (du PSW ou du CAW) doit être nulle ou identique à celle du bloc. Les instructions ISK et SSK sont modifiées pour tenir compte de ces chargements.

REMARQUE Les clés de protection mémoire sont utilisables à l'intérieur de chaque mémoire virtuelle où elles servent à préserver certaines zones.

2.1.3.4. Conclusion

Les dispositifs spéciaux du 360/67 donnent à cette machine un ensemble de fonctions et un mode de fonctionnement particuliers.

Quelques systèmes de programmation ont été développés pour cette machine. L'un d'entre eux, CP67, de conception originale et qui fût le premier à proposer un contexte de machine virtuelle, utilise les différents mécanismes du 360/67. Nous allons examiner les parties de ce système (encore appelé hyperviseur CP67) qui mettent en oeuvre ces mécanismes.

2.1.4. Mise en oeuvre des mémoires virtuelles par C.P. [Ref.C4]

2.1.4.1. Etude des fonctions nécessaires à la définition des mémoires virtuelles.

La hiérarchie de mémoire utilisée par C.P. comporte deux niveaux:

- La mémoire centrale rapide (mémoire réelle)
- Les unités périphériques rapides ou mémoire secondaire (tambours et disques) qui sont les supports externes des espaces virtuels.

Examinons donc séparément les espaces réels et les espaces virtuels.

a) Espaces réels :

Soit $M = \bigcup_j m_j \qquad j = 0,\ldots,n-1$, l'ensemble des pages de la mémoire réelle où m_j est la page de rang j, et n le nombre de taille en pages de la mémoire réelle.

Par construction, C.P. est résident en mémoire réelle, et occupe avec sa mémoire libre l'ensemble des pages définies par :

$$CP = \bigcup_j r_j \qquad j = 0,\ldots,p-1;$$

Les autres pages affectées font partie des lieux de résidence temporaires pour les espaces virtuels.

C'est l'ensemble :

$$R = \bigcup_j r_j \qquad j = p,\ldots,n-1.$$

On a évidemment :

$$R \cup CP = M$$

L'ensemble R représente l'ensemble des pages disponibles pour la pagination (cf.§ b). C'est le "R-espace" ou "espace réel" défini précédemment(2.1.1.). Plus précisément, une page r de R possède deux états possibles :

- soit occupé par une page d'une mémoire virtuelle.
- soit libre et ne contenant rien de significatif.

L'ensemble R peut donc être découpé en deux sous-ensembles RO et RL avec la relation :

$$R = RO \cup RL$$

où RO est l'ensemble des pages occupées et RL l'ensemble des pages libres.

Soit maintenant :

$$E = \bigcup_{j = 0,q-1} ej$$

l'ensemble des pages constituant la mémoire secondaire (tambours et disques). On peut écrire :

$$E = EO \cup EL;$$

où EO est l'ensemble des pages occupées sur les supports externes, et EL l'ensemble des pages libres sur ces mêmes supports.

b) Espaces virtuels

Soit maintenant une mémoire virtuelle définie ainsi :

$$V(i) = \bigcup_{k = 0,t(i)-1} vk$$

où v_k est la $k^{ième}$ page de V(i), et i le numéro de la mémoire virtuelle;

c'est-à-dire que la valeur donnée à i identifie une mémoire virtuelle et une seule.

- t(i) est la taille, en pages, de la mémoire virtuelle V(i).

On peut alors décrire l'ensemble des mémoires virtuelles par la relation :

$$W = \bigcup_{i=1,N} V(i)$$

où N est le nombre de mémoires virtuelles actives à cet instant.

Les définitions précédentes montrent que les pages des mémoires virtuelles doivent avoir un support physique; De ce fait on est amené à définir des transformations qui projettent W dans R ∪ E. Ces transformations ou fonction d'application sont appelées : la pagination.

c) Applications de W dans R ∪ E

Nous pouvons remarquer tout naturellement qu'il y a autant d'applications que de mémoires virtuelles V(i) distinctes. Nous raisonnerons sur une seule mémoire virtuelle V(i) :

Soit une page vk ∈ V(i); à un instant donné. Elle peut avoir un transformé, soit dans E, soit dans R; soit les deux (nous envisageons le cas le plus général) *(fig.2.1.4.(a))*.

Ceci introduit deux transformations f(i) et g(i) telles que :

$$V(i) \xrightarrow{f_i} R$$

$$V(i) \xrightarrow{g_i} E$$

On doit remarquer que f_i n'est pas définie pour tout v ∈ V(i)

puisque V(i) peut être très supérieure à R. Par contre, pour éviter toute impossibilité de fonctionnement, g_i est définie pour tout

$$v \in V(i)$$

Dans le cas où il existe pour $v \in V(i)$ deux transformés $f_i(v)$ et $g_i(v)$, une relation de précédence entre f_i et g_i résoud "l'ambiguité". En effet

$$f_i(v) \in R$$

est seul valable, car il est le seul transformé qui a pu être le plus récemment modifié par le programme résident dans v(i).

En résumé nous pouvons énoncer la règle suivante :

Pour tout vk $\in$ V(i), on applique d'abord la transformation f_i, Si $f_i(vk)$ existe, c'est le transformé cherché. Sinon, on applique la transformation g_i, qui, par définition, est telle que $g_i(vk)$ existe toujours.

d) La page identiquement nulle.

Appelons C(v), le contenu de la page $v \in V(i)$. A l'initialisation d'une mémoire virtuelle, pour tout $v \in V(i)$, C(v) n'est pas défini à priori.

Il est intéressant de fixer ce contenu comme étant identiquement nul. C'est-à-dire, que tous les bits représentant C(v) sont à zéro. Dans ce cas on notera :

$$C(v) = Z \quad \text{avec} \quad Z \in E;$$

où plus précisément Z qui possède la propriété de ne jamais être modifié (le

contenu de Z est zéro et doit le rester) est une extension de EO. De cette manière, l'initialisation de la mémoire virtuelle se traduit par :

$$C(v) = Z \qquad \forall v \in V(i).$$

V(i) est donc, quelle que soit sa taille, entièrement connue et définie par le contenu de la seule page Z.

On conçoit aisément le gain de place procuré par cette solution. En effet, dans un but de généralisation, la totalité des mémoires virtuelles aura comme valeur initiale Z ou encore :

$$C(v) = Z \qquad \forall v \in V(i)$$
$$\text{et} \quad \forall i \qquad i = 0,1,...N$$

2.1.4.2. Description des tables utilisées par CP.

Comme nous l'avons montré au paragraphe 2.1.3., un certain nombre de tables sont indispensables pour appliquer f_i et g_i.

Voyons le cas de CP :

a) M espace et CORTABLE.

Rappelons que :

$$M = CP \cup RO \cup RL = \bigcup_{j=1,n} m_j$$

La description de l'ensemble M de pages est faite dans CP par une table :

"La CORTABLE".

Pour j = 0,n-1, il existe une entrée dans la CORTABLE qui décrit m_j. Cette entrée indique si :

$m_j \in CP$ ou $m_j \in RO$ ou $m_j \in RL$.

(fig.2.1.4.(b))

Si une page appartient à RO, une entrée de la CORTABLE doit définir à quelle V(i) elle appartient. La solution adoptée est de repérer la suite de mots qui décrit le propriétaire de V(i), en conséquence l'entrée contient l'adresse de la UTABLE (voir alinéa e).

Deux renseignements supplémentaires dans chaque entrée indiquent s'il existe une copie de la page sur support externe. Pour cela, une adresse repère une entrée de la table d'occupation des supports externes appelée SWAPTABLE (voir alinéa suivant). La seconde information est le paramètre LOCK; il précise si cette page doit être conservée en mémoire réelle (espace M) en permanence et aussi ne jamais changer de hiérarchie. Plus précisément LOCK est un sémaphore que l'on incrémente lorsqu'on désire garder une page en mémoire réelle, et que l'on décrémente lorsque cette condition n'est plus nécessaire. Seule la condition : LOCK = 0, permet à la page de quitter la mémoire réelle.

Pour les deux ensembles CP et RL, la solution est plus simple. En effet, lorsque le pointeur sur la UTABLE contient les caractères * CP *, cette page appartient à CP. Par contre lorsque le pointeur contient la valeur hexadécimale "FFFFFFFF", alors la page appartient à RL.

b) E-espace et indicateurs ALLOC

Rappelons que :

$$E = EO \cup EL = \bigcup_{j=0,q-1} e_j$$

Chaque e_j a deux états, il est donc possible de représenter les états EO et EL par 1 et 0 respectivement. Ainsi l'espace externe E est géré à l'aide d'une suite d'indicateurs (table de bits) appelée :

ALLOC

La page e_j est identifiée par le bit de j et son état, par la valeur de ce bit :

si $e_j \in EO$; ALLOC(j) = 1

si $e_j \in EL$; ALLOC(j) = 0

c) SEGTABLE et PAGTABLE - L'application f_i

Rappelons que :

$$V(i) \xrightarrow{f_i} R$$

f_i applique l'espace des noms V(i) dans l'espace réel R. Sur la machine utilisée par CP (le 360/67), cette application a la forme imposée par la technologie, c'est-à-dire : table de segment et table de page, SEGTABLE et PAGTABLE respectivement (voir § 2.1.3.).

d) SWPTABLE - L'application g_i

Rappelons que :

$$V(i) \xrightarrow{g_i} E$$

Nous avons vu précédemment que, pour tout :

$$v \in V(i) \qquad g_i(v) \in E$$

Là encore, l'application est définie à l'aide d'une table : la SWPTABLE *(fig.2.1.4.(c))*.

Elle contient autant d'entrées que de pages $v \in V(i)$. Chaque entrée de la SWPTABLE contient :

- une suite d'indicateurs que nous ne détaillerons pas.
- le numéro de page virtuelle qui figure le rang de l'entrée dans cette table.
- une zone appelée clé qui contient les valeurs de la clé de protection mémoire associée à la page. Ceci est rendu nécessaire par le fait que les clés n'appartiennent pas au contenu de la page; ceci veut dire que si la page, $v \in V(i)$, réside sur un support externe, la valeur des clés associée à cette page est rangée dans la zone CLE.
- la zone, "adresse sur support externe", qui définit l'emplace-

ment de la page dans l'espace EO, c'est-à-dire dans la pratique sur les unités périphériques à accès direct.

e) Relation entre les tables

Les tables qui définissent f_i et g_i sont fonction de i, numéro de la mémoire virtuelle considérée. Comme il n'y a qu'une mémoire virtuelle par machine virtuelle générée par CP, i est représenté dans CP par la table UTABLE *(fig.2.1.4.(d))*. On peut la considérer comme le catalogue général qui identifie de manière complète la machine virtuelle concernée. Le couplage UTABLE, SEGTABLE, PAGTABLE, SWPTABLE, CORTABLE, se fait suivant la *figure 2.1.4.(e)*.

Le lien (1) identifie l'utilisateur courant, appelé RUNUSER, ce qui revient à fixer i.

Le lien (2) est imposé par la technologie et par là-même il est possible de charger le registre de contrôle CRO (§ 2.1.3.).

Le lien (3) est imposé par la technologie.

Le lien (4) lie la fonction f_i à la fonction g_i, lorsque pour un

$$v \in V(i) \quad \not\exists \; f_i(v).$$

Les liens (5) et (8) permettent de savoir à qui appartient une page de la mémoire réelle (lien 8) et où est sa copie sur support externe (lien 5) à l'aide de l'entrée correspondante dans la SWPTABLE.

Les liens 6 et 7 sont formés par les numéros de pages virtuelles (7) et réelles (6), qui permettent de calculer, par indexation, l'adresse cherchée soit dans la PAGTABLE (7), soit dans la CORTABLE(6).

2.1.4.3. Gestion effective.

Nous allons examiner maintenant la façon dont CP utilise les diverses tables que nous avons détaillées et ceci dans l'ordre chronologique

de leur utilisation.

a) Initialisation de CP.

Après son installation en mémoire, CP initialise la CORTABLE de telle façon que :

$$M = CP \cup RL \cup RO \quad \text{avec } RO = \emptyset$$

De même, la table ALLOC est initialisée de telle façon que :

$$E = EO \cup EL \qquad \text{avec } EO = \emptyset$$

b) Entrée d'un utilisateur dans le système.

Lorsqu'un utilisateur propriétaire d'une machine virtuelle, joint le système en nommant au terminal sa machine virtuelle et en donnant son mot de passe, CP crée entre autre chose les tables suivantes :

- la UTABLE
- la SEGTABLE avec les liens vers la (ou les) PAGTABLE.
- la (ou les) PAGTABLE de sorte que f_i soit telle que : $\forall v \in V(i)$, il n'y ait pas de transformé (aucune page de la mémoire virtuelle en mémoire réelle).
- la (ou les) SWPTABLE liée à la PAGTABLE, de sorte que la zone "adresse sur support externe" contienne l'adresse de Z, c'est à dire :

$$\forall v \in V(i) \qquad g_i(v) = Z.$$

La page Z est une constante du système, elle peut donc être définie de deux façons :

- Soit par une page physiquement pleine de zéros sur un support externe.

- Soit par une séquence d'instructions qui remet à zéro une page donnée de RL.

Cette dernière solution permet un gain de temps important, car le nombre d'instructions nécessaire pour lancer une lecture de page est plus grand que le nombre d'instructions nécessaires pour remettre à zéro une page par programme.

c) Initialisation de la mémoire virtuelle.

Une machine virtuelle n'a d'utilité que lorsque sa mémoire contient un programme (en général d'ailleurs un système de programmation) capable de gérer les ressources physiques qui lui sont données. Remarquons que l'installation de ce programme sur une machine réelle se fait à l'aide d'une fonction câblée. C'est, dans le cas du 360, la fonction IPL.

CP peut réaliser cette opération de deux manières différentes; chacune d'elles étant choisie par l'utilisateur :

- IPL par adresse

  L'utilisateur précise l'adresse virtuelle du dispositif sur lequel se trouve le programme à charger.
  CP simule alors la fonction IPL classique du 360; et c'est la machine virtuelle qui, à partir de cet instant, de par sa propre autonomie, charge le programme en mémoire.

- IPL par nom.

  L'utilisateur précise le nom symbolique du système qu'il veut amener dans la mémoire virtuelle. Pour que ceci soit possible, il faut que ce système soit connu de CP et qu'il ait subi un traitement préalable que nous allons examiner à titre d'exemple avec le système privilégié CMS :

A partir d'une unité d'entrée sortie quelconque (bande, lecteur, etc...) le noyau de CMS est installé en mémoire et il exécute ses propres instructions initiales. Le pré-traitement peut se faire indifféremment sur une machine réelle, ou sur une machine virtuelle. On active alors un programme utilitaire qui recopie, page par page, l'image mémoire du noyau, sur disque *(fig.2.1.4.(f))*. L'espace disque, ainsi créé, est une extension ESYS de EO qui possède, comme Z, la propriété de n'être jamais modifié c'est-à-dire encore de n'être accessible qu'en lecture.

Revenons à CP; Il connait de façon statique aussi bien l'emplacement de ESYS sur disque que la taille de ESYS en nombre de pages; il est donc capable de construire une SWPTABLE réduite associée à ESYS seul.

Il attribue, à cette partie de la SWPTABLE, le nom symbolique donné au système, dans notre exemple: "CMS".

Lorsque l'utilisateur demande l'IPL par nom de CMS (IPL CMS), CP recopie donc les adresses correspondantes à ESYS à l'emplacement voulu dans la SWPTABLE de la mémoire virtuelle considérée *(fig.2.1.4.(g) et (h))*. On a donc à cet instant :

$$V(i) = ESYS \cup Z$$

et

$$V(i) \cap R = \emptyset$$

Si une autre machine V(k) est, au même moment, dans le même cas, (ce qui est possible), on aurait :

$$V(k) = ESYS \cup Z$$

et

$$V(k) \cap R = \emptyset$$

Plus généralement, on peut avoir plusieurs machines virtuelles dans le même contexte au même instant, ce qui explique que ESYS et Z ne doivent pas être

modifiés, sous peine de voir intéragir plusieurs mémoires virtuelles qui, par définition, doivent rester logiquement disjointes. Il y a seulement partage entre les mémoires virtuelles d'un sous-ensemble particulier de E :

$$ESYS \cup Z$$

Cette technique présente un aspect très intéressant et nous verrons, par la suite, (§ 3.5) les différents avantages que l'on peut en tirer.

d) Fonctionnement normal

Nous venons de voir qu'à l'initialisation d'un système, CMS par exemple, l'ensemble de la mémoire virtuelle V(i) est représenté par : ESYS $\cup$ Z; et $f_i(v)$ n'est pas défini $\forall v \in V(i)$.

Dès que la mémoire virtuelle V(i) est active, c'est-à-dire que CRO contient l'adresse de la SEGTABLE associée, la première référence à la mémoire virtuelle provoque une interruption pour page manquante puisque $f_i(v)$ n'est pas défini. CP détermine la première page , $vk \in RL$, à l'aide de la CORTABLE. Si une telle page existe ($RL \neq \emptyset$), il note le numéro correspondant dans la PAGTABLE, définissant ainsi f_i pour cette page virtuelle v. Les adresses de la UTABLE et de la SWPTABLE sont ensuite écrites à leur emplacement dans la CORTABLE, et LOCK est incrémenté de 1(il était à 0).

A partir de l'adresse sur support externe, contenue dans l'entrée adéquate de la SWPTABLE associée à V(i), CP initialise une opération de transfert qui amène la page v de l'espace E dans la page rk choisie. En attendant la fin de cette opération, CP désactive la machine virtuelle.

Lorsque l'entrée/sortie est terminée, CP décrémente le LOCK de 1 dans la CORTABLE, et place la machine virtuelle dans la liste des machines candidates à être activées.

Dans le cas où $RL = \emptyset$, la recherche dans la CORTABLE n'a pas

fourni de page rk $\in$ RL, une seconde recherche dans la CORTABLE permet de sélecter une page $r_\ell \in$ RO tel que LOCK(r_ℓ) = 0 avec r_ℓ solution de $f_j(v)$ $v \in V(j)$.

C'est ici qu'interviennent les clés de protection mémoire étendues. En effet, une page r_ℓ a été choisie de façon à donner au moins un élément à RL, mais cette page redeviendra élément de E (c'est-à-dire réécrite sur un support externe) si et seulement si les clés de protection associées à cette page ont le bit : "page modifiée".

. si ce bit est à 1, cela signifie que :

$$C(f_i(v)) \neq C(g_i(v))$$

ce qui nécessite une opération d'entrée-sortie pour réattribuer la page à E.

. Par contre, si ce bit est à 0, on sait que :

$$C(f_i(v)) = C(g_i(v)),$$

et que l'opération de réécriture est inutile.

En cas de réécriture de la page, (page modifiée), il est nécessaire de savoir si l'adresse dans la SWPTABLE doit être utilisée pour renvoyer la page dans E. Il ne faut pas que cette adresse corresponde à une page de ESYS ou Z.

D'une façon générale, CP doit connaître si l'adresse contenue dans l'entrée de la SWPTABLE est valide en écriture ou non. Cette information est précisée par le bit : RECOMPUTE qui appartient à la zone des indicateurs de l'entrée j de la SWPTABLE de V(j). Avant toute reécriture de page, si le bit RECOMPUTE est à 1, CP examine la table ALLOC pour trouver une page e $\in$ EL. Il marque cette page e comme appartenant à EO, et met son adresse dans la SWPTABLE de V(j). CP remet ensuite le bit recompute à 0. Ce bit aurait été mis à 1 à la lecture d'une page située dans ESYS. Il est destiné à éviter de modifier les zones lecture seulement de l'espace externe.

Revenons à la page r_ℓ qui est à reécrire; elle était solution de $f_j(v)$ pour la mémoire virtuelle V(j) (en général $j \neq i$). Puisque la page r_ℓ est renvoyée dans E, elle n'est plus solution de $f_j(v)$. Il faut donc invalider l'entrée correspondante de la PAGTABLE de V(j). L'opération d'écriture est initialisée par CP ce qui rend la machine V(j) inactivable. A la fin de l'entrée-sortie, la page r_ℓ est devenue libre donc élément de RL et nous sommes ramenés au cas où l'on avait trouvé dans la CORTABLE une page $r \in RL$.

Les transferts de pages entre les différents espaces, ainsi que l'ordre dans lequel ils se font, sont représentés par la *figure 2.1.4.(i)*.

REMARQUE : Le facteur de répétition relatif aux transferts 1 et 3 est dû au fait qu'une page peut éventuellement être lue un très grand nombre de fois avant d'être réécrite puisque l'opération de réécriture n'est nécessaire que si la page a été modifiée.

e) Sortie du système d'un utilisateur (LOGOUT).

Lorsqu'un utilisateur quitte le système, d'une part l'ensemble des tables fonctions de i (UTABLE, SEGTABLE, PAGTABLE, SWPTABLE) est rendu à la mémoire libre de CP et d'autre part, l'emplacement occupé par V(i) autant dans RO que dans EO est, lui aussi, rendu respectivement à RL et EL. Ceci consiste à mettre à jour les entrées intéressées de la CORTABLE et d'ALLOC.

REMARQUE : Nous avons montré comment CP fait passer un élément de RO dans RL (écriture d'une page) mais rien n'a été dit pour le transfert de EO dans EL. De façon très restrictive, LOGOUT est le seul moyen prévu initialement dans CP pour faire passer un élément de EO dans EL. Nous verrons au § 3.1. le moyen que nous allons utiliser pour remédier à cet état de fait.

Nous venons de voir comment CP met en oeuvre le concept de Mémoire virtuelle, voyons maintenant comment CMS l'utilise.

2.1.5. Emploi de la mémoire virtuelle par CMS [Ref.C2]

Le système conversationnel CMS, de par sa conception et son état de fonctionnement actuel est prévu pour utiliser une machine IBM 360 standard qu'elle soit réelle ou virtuelle. De ce fait, il n'y a pas de structures particulières qui pourraient lui donner des privilèges trop importants mais seulement quelques mécanismes qui le lient à CP.

Puisque CMS est opérationnel sur une machine réelle type 360 standard, il utilise donc, pour son fonctionnement, les techniques classiques de programmation de cette machine, en particulier pour les entrées sorties. Quant à la gestion et à l'utilisation de la mémoire, le fonctionnement est très conventionnel; CMS détermine la taille de mémoire dont il dispose en faisant référence page à page, à l'ensemble de la mémoire, jusqu'à ce qu'il recoive une interruption programme pour adressage invalide. La taille, ainsi calculée, est définitivement fixée et n'est pas recalculée au cours de la session. Dans le cas où CMS est actif sous CP, il procède de la même manière, ce qui signifie que s'il existait un moyen d'augmenter dynamiquement la taille de la mémoire, CMS ne l'utiliserait pas. Les deux attaches qui lient CMS à CP dans le cadre de la gestion de la mémoire, sont les suivantes :

1/ Lien dû à l'architecture du noyau.

Un certain nombre de modules composant le noyau de CMS sont réentrants. Cette partie caractéristique du noyau peut donc être partagée par toutes les machines virtuelles qui utilisent le système CMS. En conséquence, un mécanisme (§ *2.1.4.2 (a)*) propre à CP verrouille l'ensemble des pages réentrantes en mémoire rapide.

2/ Lien dû à l' IPL par nom.

Pour faciliter la recopie du noyau de CMS dans ESYS, l'initialisation de CMS se déroule logiquement en deux étapes. La première partie de cette

opération est l'initialisation principale de CMS. Elle se termine par une demande explicite de CMS qui a besoin de connaître s'il faut recopier le noyau dans ESYS. Deux cas sont à considérer.

a) Dans le cas d'une réponse positive, CMS demande alors de placer un arrêt adresse en un point bien défini de sa mémoire (adresse hexadécimale FO).

Cet arrêt permet de charger un programme utilitaire qui recopie l'image mémoire du noyau dans ESYS. C'est ce programme qui établit le lien entre CP et CMS.

La partie de CMS ainsi recopiée, représente le système presque totalement initialisé. En conséquence, chaque opération future dûe à un IPL par nom se traduit uniquement par l'initialisation secondaire puisque l'initialisation principale n'est pas à refaire.

L'initialisation secondaire consiste seulement à calculer la taille de la mémoire (V(i). Nous n'avons pas besoin d'insister sur le gain de temps assez considérable d'une telle méthode).

Toutefois, un inconvénient de cette solution pour CMS (moindre que ses avantages), est que l'initialisation principale consiste, en particulier à construire en mémoire, dans la partie réservée au noyau, le catalogue des fichiers résidant sur le disque système.

Toute modification du catalogue de ce disque système rend donc obligatoire la recopie d'un nouveau noyau dans ESYS.

b) Dans le cas d'une réponse négative, l'initialisation secondaire a lieu immédiatement sans sauvegarde de ce noyau.

L'organisation générale du noyau de CMS est indiquée par la *figure 2.1.5.(a)*.

2.2. GESTION DES DISQUES APPARTENANT AUX MACHINES VIRTUELLES.

2.2.1. MINI-Disques. [Ref.C3, C4]

a) Rappels : structure d'un disque [Ref.I1]

Nous ne voulons pas donner ici le mode d'emploi des unités de disques mais définir la structuration de l'espace qu'il représente car cette structuration est imposée par la logique des unités de contrôle qui gèrent ce type des périphériques.

Nous examinerons le cas des unités de disque IBM 2314, car ce sont celles utilisées en général avec un 360/67, donc par CP.

Un 2314 comporte 203 cylindres numérotés de 0 à 202. Chaque cylindre est constitué de 20 pistes numérotées de 0 à 19.

La structure d'une piste n'est pas imposée par le constructeur. La seule restriction est qu'il doit exister un repère sur la piste, sous forme de numéro de cylindre, numéro de piste, écrit en tête de celle-ci, à un emplacement imposé par la technologie. Ce repère appelé HOME-ADRESS, est indispensable à l'unité de contrôle, pour vérifier que le déplacement du bras du disque et que la sélection de tête, ont été faits correctement après une opération de SEEK.

Quant aux informations écrites sur la piste, elles ont un format libre. Ce sont les enregistrements.

Chaque enregistrement peut être de longueur quelconque. Il est précédé de son identification qui est soit une clé de recherche, soit une identification reprenant les données de la HOME-ADRESS, complétée par le numéro de l'enregistrement.*(Fig.2.2.1.(a))*.

C'est cette dernière solution qui a été adoptée par CP pour "pré-formater" ses disques. Les enregistrements sont séparés les uns des autres par des espaces appelés entre-enregistrements. Ainsi, la recherche d'une information sur une piste, se fait à l'aide de la commande SEARCH, qui a comme argument de recherche, l'identification de l'enregistrement.

Cette organisation est représentée sur la *figure 2.2.1.(b)*. Cette figure appelle deux remarques :

REMARQUE 1 : Les numéros d'enregistrements sont croissants sur la même piste, mais ne commencent pas forcément par le numéro 1.

REMARQUE 2 : Un enregistrement peut se trouver à cheval sur deux pistes. L'unité de disque est alors capable de lire ou d'écrire les deux parties en une seule opération.

b) Définition d'un MINI-Disque.

On appelle MINI-disque, au sens CP, une collection de cylindres consécutifs appartenant à un même disque physique. Le premier cylindre du MINI-disque sera numéroté 0, le second 1, le 3e 2, etc...

- un MINI-disque ne peut avoir une taille inférieure à un cylindre.
- un MINI-disque peut, au plus, avoir la taille d'un disque complet.

Il convient de justifier cette définition; elle est rendue nécessaire par le fait que la plus grande partie des machines virtuelles, qui utilisent CP, ont besoin d'espace disque pour recueillir de façon permanente leurs informations. Le nombre des machines actives à un instant donné est très souvent supérieur au nombre d'unités physiques que possède la configuration réelle, il

n'est donc pas possible d'affecter une unité physique à chaque machine active. Un autre aspect, tout aussi important, et d'une part l'ensemble des opérations fréquentes de démontage et remontage de cartouches de disques, à chaque fois qu'une machine quitte ou joint le système, et d'autre part le cas des machines qui ont besoin de plusieurs unités de disque pour leur activité.

La notion de MINI-disque présente donc l'avantage de donner, à chaque machine virtuelle, l'espace disque nécessaire à son fonctionnement de telle sorte qu'il n'y ait pas de restriction notable sur le nombre d'utilisateurs.

Un MINI-disque se comporte donc, par l'intermédiaire de CP, comme un disque complet, sa taille mise à part, et devient une unité à laquelle CP attribue certaines particularités d'accès qui augmentent sa souplesse d'emploi; Ce sont essentiellement des facilités de partage :

- Partage,en lecture seulement, d'un MINI-disque entre plusieurs utilisateurs.

  Cette facilité permet l'accès simultané de plusieurs machines virtuelles à un seul MINI-disque, en lecture seulement. C'est le cas, par exemple, du mini-disque système CMS. (On affecte à ce type de mini-disque l'attribut RDONLY).

- Autorisation de lecture d'un MINI-disque.

  Le propriétaire d'un MINI-disque a attaché son VERROU à son unité disque; il peut en autoriser l'accès, en lecture seulement, à toutes les machines virtuelles qui connaissent le MOT-CLE d'accès à ce disque.

  Cette facilité n'est autorisée par CP qu'à la condition que le propriétaire du MINI-disque ne l'utilise pas en écriture. Si cette condition est réalisée tous les utilisateurs qui possèdent le MOT-CLE ont accès simultanément à ce MINI-disque (on affecte à ce type de mini-disque l'attribut RDSHAR).

- Autorisation d'écriture d'un MINI-disque

La propriétaire d'un MINI-disque peut de la même façon que précédemment donner accès en écriture à son disque. Dans ce cas cependant, un seul utilisateur a accès à ce disque, à un instant donné.(On affecte à ce type de mini-disque l'attribut WRSHAR).

c) Principe de mise en oeuvre et de fonctionnement des MINI disques réalisés par CP.

Le fait de simuler la présence de plusieurs disques, sur une seule unité, entraine la mise en oeuvre par CP d'un certain nombre de mécanismes de simulation, que nous allons examiner un par un :

c-1) Conversion des adresses d'unité.

Lorsqu'une machine virtuelle commande une opération d'entrée-sortie, elle indique l'unité sur laquelle cette opération doit s'effectuer, grâce à un numéro encore appelé: adresse virtuelle de l'unité. Cette adresse dépend de la configuration de la machine et comme-telle,doit être liée au monde réel. Il est donc nécessaire pour CP, de faire correspondre à cette adresse virtuelle une adresse d'unité qui existe réellement, et sur laquelle est défini le MINI-disque en cause.

c-2) Facteur de translation de l'argument de SEEK.

Lors d'une opération d'entrée-sortie sur disque (sauf cas particulier) il est nécessaire de sélecter, d'une part le cylindre, d'autre part la piste sur laquelle doit se faire l'opération. Ceci est réalisé par la commande SEEK qui admet un argument analogue au contenu de la "HOME ADRESS". Cet argument est de la forme : OO CC HH avec les conventions de la *figure 2.2.1.(a)*.

Le numéro de cylindre CC, donné par la machine virtuelle, est celui de son MINI-disque, et non pas le numéro de cylindre réel où réside l'information. CP doit donc appliquer au numéro de cylindre donné par la machine virtuelle, le facteur de translation qui correspond au numéro réel du 1er Cylindre appartenant au MINI-disque, sur l'unité réelle *(fig.2.2.1.(c))*. Pour le numéro de tête HH(c'est-à-dire le numéro de piste) aucune modification n'est nécessaire; en effet un MINI-disque est une collection de cylindres, donc le numéro de piste HH n'est pas affecté.

c-3) Problèmes liés aux dépassements des limites d'un MINI-disque.

Dans le cas où l'argument de SEEK, donné par la machine virtuelle, dépasse les limites de l'espace disque qui lui est affecté, il est indispensable de renvoyer à celle-ci une condition : "fin de disque", telle qu'elle existe sur une unité réelle. Seul un contrôle "Software" permet de s'assurer qu'une opération d'entrée-sortie s'exécute dans les limites imposées et permet d'empêcher qu'une machine virtuelle détruise le contenu d'un MINI-disque qui ne lui appartient pas.

Cette méthode de contrôle est la seule valable. En effet, une machine virtuelle doit pouvoir, au cours d'une opération d'entrée-sortie, utiliser toutes les commandes d'accès aux disques, en particulier celles qui servent à protéger les espaces disques par un mécanisme de clé. On notera que cette condition "fin de disque" est utilisée par CMS pour calculer dynamiquement la taille des MINI-disques qu'il utilise.

c-4) Translation des données de WHA.

Nous avons vu que la HOME-ADRESS est utilisée par l'unité de contrôle des disques pour vérifier que l'opération de SEEK s'est déroulée correctement.

Il existe une commande spécifique des unités à accès direct, qui est le "WRITE HOME-ADRESS", qui permet de réécrire cet emplacement.

Dans l'exemple de la *figure 2.2.1.(c)*, si la machine virtuelle désire réécrire la HOME-ADRESS du cylindre 03 piste 04, il faut que CP translate l'argument de cette opération, sinon la HOME-ADRESS du disque réel ne serait plus en accord avec la numérotation physique. Pour cela CP réalise sur l'argument de l'opération "WHA", la même translation que dans le cas de SEEK.

c-5) Gestion des privilèges d'accès.

Pour résoudre les problèmes de partage entre plusieurs machines virtuelles d'un même MINI-disque, CP doit s'assurer que les accès concordent avec les privilèges. Dans le cas d'un accès en lecture seulement, CP doit vérifier que le MOT CLE correspond au VERROU et que les commandes d'entrées sorties n'altèrent pas le contenu du disque.

Dans le cas d'un accès en écriture, CP vérifie seulement la correspondance entre le MOT CLE et le VERROU et n'impose aucune contrainte sur l'utilisation des commandes d'entrée-sortie.

Les opérations que nous avons détaillées sont relatives à la réalisation du mécanisme des MINI-disques. Il est cependant une opération plus complexe rendue nécessaire par le fait que les données des opérations d'entrée sortie résident en mémoire virtuelle, alors que l'ensemble des commandes et la technologie sont définies pour une mémoire réelle. Cette opération particulière est appelée : la compilation des programmes canaux.

2.2.2. Compilation des programmes canaux

Un programme canal, au sens de la série des machines 360, est une suite de commandes qui spécifient les opérations à effectuer sur une unité d'entrée-sortie.

La suite de commandes est un programme destiné au "calculateur canal", et la structure de chacune d'elles est montrée par la *figure 2.2.2.(a)*.

L'ordre normal d'exécution des commandes est la séquence. Il existe cependant des commandes assimilables à des opérations de "rupture de séquence conditionnelle" (par exemple SEARCH sur une unité à accès direct); ainsi que des commandes de "rupture de séquence inconditionnelle", (TIC).

Lorsqu'une opération d'entrée sortie est initialisée par une machine virtuelle, elle fournit normalement, au canal, le programme qui lui est destiné.

Dans le cas d'une machine virtuelle, le lancement de cette opération doit être intercepté par CP, car le programme canal, construit par la machine virtuelle, ne peut pas être exécuté tel qu'il est par le canal réel. En effet, le programme canal est écrit dans l'espace virtuel pour l'espace virtuel.

Appelons programme canal source, le programme canal construit par la machine virtuelle; et programme canal objet, le programme réellement utilisé pour effectuer l'opération d'entrée sortie, telle qu'elle a été voulue par la machine virtuelle.

Cette traduction "programme canal source", "programme canal objet", est une véritable compilation puisqu'il convient de définir une fonction calculable qui applique l'espace virtuel dans l'espace réel. Ceci est dû aux restrictions suivantes apportées par la technologie :

1) <u>Restrictions sur les conditions de déroulement du programme canal</u>.

Un canal est assimilable à un calculateur périphérique qui a sa propre autonomie. Il existe donc des problèmes de synchronisation canal-unité centrale. De plus, le programme canal qui s'exécute, est étroitement dépendant

du temps car il accède, en général, à des unités comportant des éléments mécaniques possédant leur propre inertie. En conséquence, un programme canal peut donc être assimilé à une tâche temps réel.

Voyons ce qui peut se passer dans l'espace virtuel :

Les adresses placées dans les commandes du programme canal source, par la machine virtuelle, sont des adresses dans la mémoire virtuelle. De ce fait, les arguments des commandes peuvent ne pas être présents en mémoire réelle, au moment où l'opération d'entrée-sortie est lancée, et par là-même on pourrait admettre que le canal génère une interruption, pour page manquante, au cours de l'exécution du programme canal.

Ceci est technologiquement impossible: on ne peut pas par exemple, arrêter l'écriture sur une bande, le temps d'arrêter en mémoire la suite des informations puisque l'arrêt d'une bande sépare deux enregistrements. Il faut donc, avant d'initialiser une opération d'entrée-sortie, examiner toutes les commandes constituant le programme canal source, pour amener en mémoire réelle tous les arguments intéressés et les verrouiller jusqu'à l'achèvement de l'opération c'est-à-dire que la totalité des informations intéressées est bloquée en mémoire réelle.

UNE REMARQUE montre la complexité d'un tel problème :
Les commandes, ont en général, un format bien établi, *(fig.2.2.2.(a))*. Malheureusement, seul le code opération de la commande est examiné de façon systématique par le canal, car, pour certains codes opérations, l'action à effectuer est suffisamment définie par le code lui-même, et le canal ne tient pas compte des autres paramètres de la commande qui peuvent donc contenir n'importe quoi. Ainsi pour éviter d'amener et de bloquer en mémoire trop (beaucoup) de pages, CP doit examiner toutes les opérations pour déterminer les commandes qui n'ont pas de paramètres. Cet aspect est très important car ces commandes sont différentes suivant le type d'unité sélectée.

Nous illustrons ce qui précède par un exemple sur une unité de disques 2314. Choisissons le code particulier RECALIBRATE (positionner le bras du disque sur le cylindre numéro 0). On conçoit que pour ce cas, seul le code opération est nécessaire et une commande écrite, comme sur la *figure 2.2.2.(b)* est donc acceptable par le canal.

Si CP ne traite pas ce cas particulier, il doit amener huit pages de la mémoire virtuelle situées à partir de l'adresse virtuelle hexadécimale 40 000. Si cette adresse est valide, (inférieure à la taille de la mémoire) huit pages de la mémoire réelle sont immobilisées inutilement et un certain nombre d'entrées-sorties inutiles ont été nécessaires pour les y amener. Si cette adresse est invalide, (extérieure à la mémoire virtuelle) l'opération est déclarée invalide alors qu'elle aurait dû se dérouler normalement.

On voit donc que cette seule partie de la compilation (tester les codes opérations pour définir le type d'opération et amener si nécessaire les pages référencées en mémoire réelle) n'est qu'une succession de cas particuliers.

Cet aspect rend la compilation particulièrement longue et délicate.

Ce n'est pourtant pas la partie la plus complexe du problème.

2) Nécessité de traduction des adresses - génération de commandes supplémentaires.

Une commande destinée à être exécutée par un canal, contient comme nous l'avons vu, une adresse mémoire ainsi qu'une longueur. Ces deux valeurs définissent les paramètres de la commande, ainsi lorsqu'une commande est construite par une machine virtuelle, dans sa mémoire, l'adresse et la longueur définissent une zone contiguë dans V(i).

Soit A cette adresse, et ℓ la longueur de la zone, *(fig.2.2.2.(c))* Les canaux du 360-67 n'ont pas accès à la traduction dynamique des adresses. Il est donc nécessaire de leur fournir des adresses réelles. Si l'opérande de la commande virtuelle" est entièrement contenu dans une page, l'opérande de la commande réelle est, lui aussi, entièrement contenu dans une page et la traduction de celle-ci en celle-là, consiste simplement à recopier la "commande virtuelle" en modifiant toutefois l'adresse A en son transformé:

$$f_i(A) = B$$

Dans le cas où l'opérande, défini par A et ℓ est placé de telle façon qu'il occupe plus d'une page de la mémoire virtuelle, *(fig.2.2.2.(d))*, la transformation est plus complexe. Si vk et vk + 1 sont les pages mises en cause par l'opérande en mémoire virtuelle, leurs transformés sont :

$$r_n = f_i(vk) \text{ et } r_p = f_i(vk + 1)$$

La probabilité p = n + 1 est presque nulle. L'argument de la commande n'est donc plus représenté en mémoire réelle par une zone contigüe.

Ceci impose, dans le cas de l'exemple, de construire deux mots de commande pour le canal afin de prendre en compte les deux éléments de l'argument.

De façon générale, la longueur maximale d'un argument est de 32 765 octets. Pour satisfaire ce cas limite, il faut prévoir qu'un seul mot de "commande virtuelle" puisse se traduire par 9 mots de "commande réelle".

En résumé, pour un élément du programme canal source, il peut être indispensable de générer plusieurs éléments dans le programme canal objet. Ceci justifie le terme de compilation.

3) Cas des commandes de saut

Nous avons signalé qu'il existe des commandes assimilables à

des opérations de rupture inconditionnelle: TIC. D'après ce qui a été dit au paragraphe précédent, la longueur du programme canal objet est supérieure à la longueur du programme canal source; pour ajuster en mémoire réelle les adresses mises en jeu par les commandes TIC, une compilation en deux passages sur le programme source s'avère nécessaire.

. Le premier passage calcule la longueur du programme objet, détermine les adresses des commandes de rupture de séquence et acquiert dynamiquement la zone mémoire nécessaire où sera généré le programme canal objet.

. Le deuxième passage constitue la construction du programme canal objet.

4) Cas des programmes canaux auto-modifiables.

Il n'est pas interdit, par la technologie, de construire des programmes canaux "Auto-modifiables"; c'est-à-dire que les données de la première partie du programme canal constituent les commandes de la deuxième partie de ce programme; on conçoit mieux encore la complexité du programme de CP, qui réalise la compilation.

Cet aspect d'auto-modification introduit d'ailleurs pour CP, les limites de la compilation des programmes canaux. En effet, les programmes canaux auto-modifiables qui tiennent compte des particularités des unités, de telle sorte que leur bon fonctionnement dépende du temps, conduisent à une compilation incorrecte. Ceci entraîne donc une certaine limitation dans le cadre de CP; mais ce cas est heureusement très rare et ne constitue pas à proprement parler un handicap.

2.2.3. Gestion de fichier de CMS.

Nous savons (cf § 2.2.1.) qu'une machine virtuelle possède des mini-disques. Etudions le cas où, sur cette machine, est actif le système CMS. L'un des rôles principaux de ce système, est de gérer l'espace disque appartenant à cette machine virtuelle ou à son utilisateur (nous considèrerons ces

deux termes comme équivalents): c'est la gestion de fichier.

La gestion de fichier de CMS, considère un mini-disque comme un tout, indépendant, et le gère comme tel. Nous avons énoncé que CMS peut être activé sur ne machine réelle, auquel cas les principes de la gestion de fichiers sont applicables à l'espace défini par une cartouche complète de disques. Nous regrouperons ces deux éventualités et nous ne parlerons que de "disque", sauf s'il est nécessaire de lever une ambiguïté.

2.2.3.1. Structuration physique et gestion d'un espace disque.

Lorsqu'un système utilise pour la première fois un disque, il ne peut pas préjuger de son contenu, ni de la structure des enregistrements. CMS ne fait pas exception à cette règle et de ce fait, il est nécessaire avant toute utilisation de structurer le disque sous la forme voulue. Pour réaliser cette opération, un programme utilitaire transforme l'espace disponible en N blocs de 800 octets chacun, numérotés de 1 à N (N est fonction de la taille du disque).

De cette manière, il existe une correspondance bi-univoque simple entre le numéro d'un bloc et son adresse physique sur le disque. Tous les composants du système CMS utilisent donc le bloc comme une adresse logique sur le disque.

Le programme utilitaire mentionné précédemment écrit sur le disque des blocs initialisés à la valeur zéro. Ceci constitue, en plus, une analyse de surface.

Pour gérer l'ensemble des N blocs représentant un disque, CMS construit une table contenant N bits, appelée "table d'allocation". A tout bit de rang i(i = 1 à N) de cette table, est associé le bloc de rang i(i = 1 à N) sur le disque. La convention est la suivante : Si un bloc est disponible (non encore utilisé), le bit associé est à 0.

Si un bloc est utilisé, le bit associé est à 1. Cette technique est très comparable à celle utilisée par CP pour gérer l'espace E.

La table d'allocation est une caractéristique intrinsèque du disque et de son contenu. En tant que telle, elle doit donc être conservée sur ce disque. Elle est donc rangée sur le disque, à l'intérieur du bloc numéro 4. (Ce numéro 4 est une constante du système CMS).

Les blocs numéros 1 et 2 sont réservés à la séquence d'initialisation, pour réaliser l'opération que nous avons appelée IPL par numéro d'unité. Le bloc 3 est réservé à l'identification symbolique du disque (Label). Ainsi, un disque, nouvellement initialisé, a la structure représentée par la *fig.2.2.3.(a)*.

La gestion de la table d'allocation est réalisée par un seul programme de CMS, auquel, s'adressent tous les autres composants qui ont besoin, soit d'acquérir des blocs libres, soit de rendre des blocs non utilisés.

2.2.3.2. Structuration logique en fichiers et gestion de ces fichiers.

Un fichier peut être considéré comme une suite ordonnée d'éléments (ou enregistrements logiques).

Dans le cas de CMS, à cette structure logique est superposée un découpage arbitraire en blocs de 800 octets qui ne sont pas nécessairement contigus, et la description d'un fichier est constituée de deux tables : *fig.2.2.3.(b)*.

a) Une table donnant les caractéristiques du fichier (nom, nombre d'enregistrements logiques, longueur des enregistrements, date de création, etc..) c'est le descripteur.

b) Une table d'index contenant les numéros des blocs constituant le fichier considéré.

L'ensemble des descripteurs des fichiers sont regroupés dans un même ensemble qui constitue le catalogue. Celui-ci est rangé sur le disque dans des blocs dont les numéros sont conservés dans une Table (ou Index) qui fait partie, comme la Table d'Allocation, du bloc n°4. La caractéristique principale de CMS pour la gestion de ce catalogue est qu'il est totalement installé mémoire virtuelle lorsque le disque est actif (LOGIN). Cette gestion en mémoire, rend

très rapide la recherche d'un fichier, mais impose une reécriture périodique du catalogue sur disque; en particulier lorsqu'il a été modifié.

La structure indexée des fichiers CMS est une donnée interne du système. La plupart des composants qui utilisent les fichiers sur disque, ignorent tout de cette structure et à, fortiori, les utilisateurs.

Pour une description détaillée de la gestion de fichiers de CMS, le lecteur intéressé, pourra se reporter aux références C5, C2, L3. Ce principe de fonctionnement donne une grande souplesse d'utilisation, car pour accéder à un enregistrement logique d'un fichier, il suffit de préciser son numéro. La gestion de fichiers de CMS, exécute un algorithme de recherche pour restituer l'enregistrement demandé, ou pour inclure l'enregistrement précisé. Cette caractéristique est à rapprocher des techniques de paginations de CP décrites au § 2.1.2.

Dans le cas où CMS est actif sur une machine virtuelle, c'est-à-dire mise en activité par CP, on voit donc se superposer deux techniques de gestion de l'espace disque, identiques quant au fond, mais différentes quant à la forme, et qui s'ignorent totalement.

Il est tentant de les harmoniser et, par là-même, de profiter au maximum, sous CMS, des facilités procurées par CP, à travers l'architecture du 360-67.

C'est ce qui a été tenté dans la conception du système CMS+ que nous allons décrire au Chapitre 4.

CHAPITRE 3

EXPERIENCES PRELIMINAIRES

Dans le cadre de l'élaboration d'un nouveau système CP+/CMS+ conservant la structure programme de contrôle type CP, sur la machine réelle, et superviseur type CMS, sur les machines virtuelles; nous avons été amenés à réaliser cinq expériences préliminaires qui visent essentiellement deux buts

1/ Vérifier le bien fondé de certaines hypothèses de base du nouveau système CP+/CMS+, dont nous avons déjà parlé dans l'introduction.

2/ Améliorer le fonctionnement du système actuel.

3.1. PARTICIPATION DE CMS A LA GESTION DE L'ESPACE DE PAGINATION DE CP :

3.1.1. Préliminaire.

Tout système quel qu'il soit possède une gestion de mémoire libre. Gérer une mémoire revient, en fait, à distinguer deux classes d'emplacement mémoire :

1/ La classe des emplacements dont le contenu est significatif.

2/ La classe des emplacements dont le contenu n'est pas significatif (mémoire libre).

Dire que le contenu d'un emplacement mémoire est significatif, veut dire que, si l'on change par des moyens externes (clés du pupitre par exemple), le contenu d'un tel emplacement, cela revient à modifier le fonctionnement du programme actif.

Inversement, si une telle modification a lieu sur un emplacement mémoire non significatif, le comportement du programme actif n'est pas altéré. Chaque emplacement mémoire appartient à l'une ou l'autre de ces deux classes; connaître l'ensemble des emplacements dont le contenu n'est pas significatif, revient à connaître, du même coup, l'ensemble des emplacements dont le contenu est significatif.

En conclusion, on peut donc changer la totalité du contenu de la mémoire libre sans perturber le fonctionnement du système.

Considérons une machine virtuelle MV(i) :

Lorsque CP l'initialise, toute la mémoire virtuelle est remise à zéro c'est-à-dire est représentée par l'unique page Z (§ 2.1.4.).

Au cours du fonctionnement de MV(i), les pages de sa mémoire vont être modifiées et leur contenu ne sera plus celui de Z. Etudions l'évolution de la mémoire V(i) :

Considérons la *figure 3.1.(a)* qui représente les emplacements successifs d'une page v de la mémoire virtuelle V(i) :

- "1" représente la lecture initiale de la page v

  $v \equiv Z$

- "2" représente la première écriture de v dans E quand le contenu de v a été modifié. Cette opération a nécessité la recherche d'un emplacement ek dans E.

- "3" représente une des lectures ultérieures de la page v.

- "4" une reécriture ultérieure à la suite d'une nouvelle modification de v. Dans ce cas, il n'est pas utile de chercher un emplacement dans E puisqu'il est déjà acquit dans EO.

On voit que l'ordre des évènements peut être représenté par la chaîne suivante qui caractérise la "vie" d'une page :

$$1^{*}\ 2(3^{*}4)^{*}$$

Toute page v ∈ V(i), qui a subit l'évènement "2", occupe une place dans EO, et cette place lui est propre pendant la durée de vie de la mémoire virtuelle (la durée de vie est le temps qui sépare deux "IPL" successifs). Nous dirons que l'évènement "2" est cumulatif puisqu'il est unique dans la chaîne. La mémoire V(i) qui, avant toute activité, n'occupe qu'une page (Z) sur support externe va donc occuper au cours du temps, n emplacements dans E. (n est égal au nombre de fois où l'évènement "2" s'est produit avec n ≤ nombre de page de V(i)).

Voyons la réalisation pratique de ces diverses opérations :

Dans un souci de performance, la partie EO de E doit se trouver sur les périphériques les plus rapides, c'est-à-dire les tambours du type IBM 2301 ou 2303. Un tambour utilisé dans le cadre de CP a une capacité équivalente à 900 pages; si nous considérons alors des mémoires virtuelles de 256 K nous pouvons ranger 14 mémoires virtuelles complètes.

L'effet cumulatif montré précédemment, permet donc de prévoir une saturation de l'espace tambour dès la prise en compte de la 15ième mémoire virtuelle (si l'on ne dispose que d'une unité tambour) et par là-même, dégrader les performances du système de façon importante. En effet, dès que l'espace tambour est saturé, l'algorithme de pagination prévoit une extension sur l'espace disque disponible; La différence de vitesse de transmission et de temps moyen

d'accès est telle qu'il apparait plus souhaitable de limiter le nombre des machines virtuelles actives, que de paginer sur disque.

Nos mesures ont montré que le seuil de saturation d'un tambour est atteint pour un nombre moyen de machines virtuelles égal à 20. Ceci correspond à la présence sur tambour de 75 % des pages des mémoires virtuelles, pour une taille moyenne de mémoire virtuelle de 256 K. Cette différence s'explique par le fait qu'une machine virtuelle utilise rarement la totalité de sa mémoire. Notre première expérience a été de repousser le seuil de saturation du tambour en réalisant une communication entre CMS et CP.

3.1.2. Collaboration entre CMS et CP pour l'espace de pagination.

CMS connait, à tout instant, l'ensemble de sa mémoire libre. Il peut donc communiquer à CP la liste des pages appartenant à cette partie libre de mémoire. Cette information autorise CP à réinitialiser le contenu de ces pages à la valeur zéro. Cette opération conduit à libérer l'emplacement occupé par ces pages dans EO, et à utiliser, à nouveau, l'élément unique "Z" pour représenter toutes ces pages. D'une façon plus générale, ce que nous venons d'énoncer s'applique à la mémoire libre de toutes les machines virtuelles qui utilisent CMS.

En pratique, CMS indique à CP l'ensemble des pages constituant la mémoire libre d'une mémoire virtuelle, une seule fois au début de l'exécution de chaque programme. Au cours du fonctionnement, CMS communique à CP les pages qui sont rendues à la mémoire libre.

De son côté, CP utilise ces renseignements pour libérer les emplacements correspondants occupés dans EO, mais aussi dans R et modifie les tables (CORTABLE, PAGTABLE, SWAPTABLE) décrivant la mémoire virtuelle, pour que les pages indiquées par CMS soient représentées par Z.

Considérons la *figure 3.1.(b)* :

Elle est identique à la *figure 3.1.(a)*, mais comporte en plus l'évènement "5" qui est défini de la façon suivante :

L'évènement "5" est l'évènement qui traduit qu'une page appartient à la mémoire libre de MV(i).

L'ordre des évènements qui trace la "vie" d'une page est représenté maintenant par la chaîne :

$$(1^* \; 2(3^* \; 4)^* \; 5)^*.$$

L'évènement "2" n'est plus irréversible pour l'ensemble des pages, car il est annulé par "5".

La sous-chaîne $(3^* \; 4)^*$ constitue la partie active de la "vie" de la page, où son contenu est significatif. Les nouvelles mesures que nous avons effectuées après la mise en place de ce mécanisme, ont montré que la saturation d'un tambour intervient, maintenant, avec 25 machines virtuelles, au lieu de 20 cités auparavant. Nous avons ainsi mis en place un mécanisme qui apporte un gain de 25 % pour l'occupation de l'espace E.

Cette expérience nous a montré que, pour ce problème précis, l'interaction entre les machines virtuelles et le programme de contrôle est bénéfique. C'est une première ébauche d'environnement.

3.2. ESPACE DE COPIE

Nous avons énoncé dans l'introduction, qu'une machine virtuelle, telle qu'elle est définie dans CP, ne possède que des éléments virtuels qui, comme nous l'avons dit, sont la simulation "Software" de leurs équivalents physiques. Si l'on considère maintenant un environnement, on peut envisager l'adjonction de nouveaux éléments, appelés éléments logiques, qui ne sont pas directement issus de composants réels. De plus, ces éléments délivrés des

contraintes technologiques sont dérivés de la structure même de l'environnement et dans notre cas, adaptés au mécanisme de pagination de CP. C'est le cas de l'espace de copie.

3.2.1. Définition

On appelle espace de copie C, un ensemble de pages tel que :

$$C(i) = \bigcup_{j=1,n} cj \qquad i = 1,...m$$

A chaque environnement i actif, est attaché un espace C(i) qui lui est propre; à l'initialisation de l'environnement $C(i) = \emptyset$.

3.2.2. Fonctions d'utilisation de C_i

Pour utiliser cet espace C(i), l'environnement dispose de plusieurs fonctions :

a) V WRITE (A,N)

Par cette fonction, l'environnement i demande à CP de copier dans l'espace C(i) la page de sa mémoire virtuelle, d'adresse A. En réponse, CP lui fournit le numéro N de la page de C(i), où a été effectuée la copie.

b) VREAD (A, N)

Par cette fonction, l'environnement i demande à CP de transférer dans sa mémoire virtuelle, à l'adresse A, la page de numéro N de l'espace de copie C(i).

c) VDELETE (N)

Par cette fonction, l'environnement i demande à CP de supprimer la page de numéro N de C(i).

d) VQUIT

Cette fonction permet à l'environnement de signaler à CP qu'il ne désire plus conserver de pages dans C(i). Ceci revient à dire qu'après l'application de cette fonction C(i) = ∅ ;

NOTA :Cette fonction est réalisée automatiquement par CP, lorsqu'un environnement quitte le système (LOGOUT).

3.2.3. Fonctionnement.

3.2.3.1. Tables utilisées pour décrire C(i): CTABLES

Comme la mémoire virtuelle, l'espace de copie prend place dans l'espace de pagination E de CP. Cependant, il n'est rangé que dans la partie ED de E qui réside sur les unités périphériques disque.

En effet, il est nécessaire de faire ce choix pour éviter de saturer l'espace tambour avec des informations moins prioritaires que celles représentant les mémoires virtuelles proprement dites. On est donc amené à définir la transformation :

$$C(i) \xrightarrow{h_i} ED$$

L'espace C, unique pour chaque environnement, est défini à partir de la UTABLE. Un mot de cette table contient l'adresse de la description de la transformation h_i.

Contrairement à la mémoire virtuelle, de taille fixe, l'espace C est de taille variable au cours de l'activité d'un environnement. La fonction h_i ne peut donc pas être représentée par une simple table, comme l'est, par exemple, la fonction g_i (cf.2.1.4.) qui joue le même rôle vis à vis de la mémoire virtuelle :

$$(V(i) \xrightarrow{g_i} E)$$

La solution adoptée consiste à représenter h_i par une liste de tables. Chacune d'elles appelée CTABLE est de longueur fixe : *fig.3.2.(a)*. Chaque CTABLE est composée de 34 entrées de un mot de long, c'est-à-dire 136 octets. La structure d'une CTABLE est la suivante :

- 1er mot : Il contient l'adresse de la première entrée libre dans la table. Une entrée libre, correspond à une page de C(i) qui n'a pas de transformé dans ED, s'il n'y a plus d'entrées libres dans la table, ce mot contient 0.

- les mots 2 à 33 peuvent avoir deux significations :
  1) si l'indicateur de l'entrée (bit de gauche du mot) est à 1, cette entrée est libre, et elle contient l'adresse de l'entrée libres suivante dans la même table. Si il n'existe pas d'entrée libre suivante, la partie adresse du mot (3 octets de droite) est nulle.

  2) si l'indicateur de l'entrée est à 0, cette entrée contient un repère sur le transformé de la page de C dans ED, c'est-à-dire l'adresse disque de cette page.

- le mot 34 contient, soit l'adresse de la table suivante, soit 0, si une telle table n'existe pas.

3.2.3.2. Etude du fonctionnement à l'aide des CTABLES

Nous allons examiner le fonctionnement, fonction par fonction.

a) l'initialisation :

L'espace de copie C(i) est vide; ceci se matérialise par l'absence

de CTABLE. Le pointeur sur la 1ère CTABLE, dans la UTABLE, est donc nul.

b) 1er WRITE (A,N). La 1ère CTABLE est absente. Un espace de 34 mots est demandé à la mémoire libre de CP; et l'adresse de cette zone est placée dans la UTABLE, définissant ainsi la 1ère CTABLE.

Cette CTABLE ne comporte que des entrées libres, elles sont donc chainées les unes aux autres, et la dernière entrée est initialisée à 0. Cette première opération constitue l'initialisation d'une CTABLE. La suite de l'algorithme est décrite dans le cas général "WRITE(A,N)".

c) VWRITE(A,N) Autre que le 1er

Le premier mot de chaque table est examiné, jusqu'à trouver une table dont le premier mot soit $\neq 0$. Par déduction, ceci consiste à repérer le premier élément libre, si il existe. S'il n'existe pas nous sommes ramenés au cas du 1er WRITE, c'est-à-dire qu'il convient d'effectuer l'initialisation d'une CTABLE et de placer l'adresse de cette nouvelle CTABLE dans le dernier mot de la CTABLE précédente.

Dans tous les cas, après avoir repéré un élément libre dans une CTABLE, une demande est faite au composant de CP qui gère l'espace de pagination afin d'obtenir un emplacement dans ED pour ranger la nouvelle page.

Ce composant donne, en retour, l'adresse sur disque. Cette adresse est ensuite insérée dans l'élément intéressé de la CTABLE. A la suite de cette opération préliminaire, la page de la mémoire virtuelle, d'adresse A, est amenée en mémoire réelle par un appel au module de pagination. Par un autre appel, au module de pagination, on force sa reécriture dans C(i) à l'adresse voulue. Ces opérations terminées, CP indique à l'environnement le numéro de page N de C(i) où a été copiée la page d'adresse A.

REMARQUE : Un calcul simple permet de déterminer N, à partir de son adresse relative dans la table, et le numéro de la table.

d) VREAD (A,N)

La valeur N permet de calculer l'adresse de l'entrée intéressée dans les CTABLE. Si l'entrée est libre (bit de gauche à 1), la commande est refusée. Sinon, l'adresse de la page appartenant à C(i) est placée dans la SWPTABLE, à l'emplacement correspondant à l'adresse A.

La prochaine référence à cette page provoque une interruption pour page manquante et par là-même sa lecture automatique, grâce au mécanisme de pagination de CP.

e) VDELETE(N)

L'adresse de l'entrée de la CTABLE est calculée à partir de N; si cette entrée est libre, la commande est refusée, sinon la page correspondante de C(i) est rendue à l'espace libre de pagination, et l'entrée est chainée aux autres entrées libres dans la CTABLE.

f)VQUIT

L'ensemble de l'espace C(i) occupé est rendu à l'espace libre de pagination et les CTABLE qui décrivent C(i) sont rendues à la mémoire libre de CP.

3.2.4. Utilisation

Comme nous l'avons signalé, la création de l'espace de copie fait partie d'une série d'expériences. En conséquence, tout utilisateur de cet espace doit faire intervenir les fonctions de base décrites précédemment.

En particulier, s'il désire découper l'espace C(i) qui lui est donné en sous-ensembles logiques distincts, il doit garder en mémoire virtuelle les suites de numéros de pages, composants ces sous-ensembles.

Un autre aspect de l'utilisation de l'espace de copie est que les fonctions de base permettent des lectures successives d'une même page, en accès direct, par numéro. Par contre, aucune reécriture d'une page précédemment écrite n'est possible sans utiliser préalablement VDELETE car CP est seul maître

du numéro de page en écriture. Cette restriction est imposée pour minimiser l'encombrement des listes de CTABLE, dans la mémoire libre de CP. En effet l'algorithme construit dans CP ne permet de créér une nouvelle CTABLE que lorsqu'il n'existe plus une seule entrée libre dans les CTABLE déjà acquises.

Malgrè ces imperfections actuelles, la création de l'espace de copie nous a permis de recenser les difficultés que présentent d'une part l'utilisation de l'espace de pagination, à d'autres fins que celles prévues initialement, et, d'autre part l'utilisation des modules de pagination pour réaliser des entrées sorties ne mettant en jeu que des pages et ceci, sur l'ordre explicite d'un environnement.

3.3. REALISATION D'UNE FONCTION D'ENTREE SORTIE DISQUE.

Nous avons vu précédemment (§ 2.2.) la complexité du module qui com compile les programmes canaux générés par les machines virtuelles. Pour réaliser un environnement, il est nécessaire de concevoir des mécanismes d'entrée-sortie moins complexes et, par là-même, moins couteux en temps d'unité centrale.

3.3.1. Mise en oeuvre de la fonction dans l'environnement CMS

Lorsque CMS commande une entrée-sortie disque, il donne le contrôle à un composant particulier qui réalise cette entrée-sortie (DISKIO). Comme nous l'avons vu précédemment (§ 2.2.3.) toute la gestion de fichier de CMS utilise une unité d'enregistrement physique de 800 octets repérée par son numéro.

Pour réaliser une entrée-sortie disque, le module DISKIO doit donc connaître le numéro de bloc de 800 octets ainsi que l'adresse mémoire. Dans le cas où l'entrée-sortie porte sur plus de 800 octets, la liste de paramètres de DISKIO comporte une table de numéros de blocs, une adresse de zone d'entrée sortie, et le nombre total d'octets à transférer: *fig.3.3.(a)*.

Une telle opération impose à CMS de construire un programme canal disque dont l'emplacement et la structure, sont fixes si l'entrée-sortie porte sur un seul bloc de 800 octets. C'est le programme canal standard *(fig.3.3.(a)* dont nous examinerons la structure au §.3.3.2. Si l'entrée-sortie porte sur un nombre d'octets plus important, le programme canal est plus complexe et doit être construit dans la mémoire libre de CMS. Nous ne dirons rien de plus sur ce programme canal qui n'a pas d'intérêt pour la suite de l'exposé.

Dans tous les cas DISKIO génère ces programmes canaux en convertissant les numéros de blocs en adresse disque (CCHHR). De plus, dans un souci de simplicité, DISKIO, commande l'opération d'entrée-sortie et attend qu'elle se termine avant de rendre le contrôle au programme qui l'avait appelé. Cette solution d'attente peut paraître simpliste pour un système qui peut utiliser la simultanéité. Cependant, il faut remarquer que CMS fonctionne sur une machine virtuelle, et comme tel , n'est pas maître de ses entrées-sorties. En effet, c'est CP qui utilise la simultanéité et le temps pendant lequel une machine virtuelle est en attente, est récupéré par les autres machines virtuelles qui peuvent être activées.

Nous avons donc redéfini le module DISKIO et nous l'avons re-structuré pour créer une fonction d'entrée-sortie très simplifiée, qui est réalisée par CP. Les contraintes de cette fonction ne permettent que d'utiliser le programme canal standard *(fig.3.3.(b))*.

Les formes externes de cette fonction sont les suivantes :

LOGSIO (DISK, CAW)

où DISK est l'adresse virtuelle du mini-disque mis en jeu par l'entrée-sortie et CAW est l'adresse du programme canal standard. Cette fonction dure le temps de l'opération d'entrée-sortie complète. Pour une entrée-sortie qui met en jeu un bloc de 800 octets, une seule fonction LOGSIO est nécessaire. Pour une entrée-sortie qui met en jeu plus d'un bloc de 800 octets, autant de fonctions LOGSIO sont émises par le module DISKIO qu'il existe de blocs dans la liste des paramètres.

3.3.2. Réalisation de la fonction LOGSIO par CP

Un certain nombre de contraintes ont été imposées dans la conception de cette fonction, pour répondre à deux objectifs importants :

- d'une part, réduire la complexité de la compilation des programmes canaux, ce qui nous a imposé de ne conserver qu'un format unique pour ces programmes : *fig.3.3.(b)*.

- d'autre part, éviter une modification trop importante de CMS, ce qui nous a imposé de garder les informations concernant l'opération d'entrée-sortie, sous forme de programme canal.

En conséquence, la fonction LOGSIO est conçue comme suit :

Après vérification de la validité des paramètres (l'adresse donnée doit être l'adresse d'un disque appartenant à la configuration virtuelle et le CAW doit être aligné sur une frontière de double mot), la fonction construit le programme canal destiné à l'unité réelle associée. Pour cela elle amène en mémoire centrale, soit la page virtuelle concernée par l'opération d'entrée-sortie, soit au plus les deux pages virtuelles si la zone d'entrée sortie chevauche une frontière de page. Cette implantation en mémoire permet de calculer l'adresse réelle à placer dans les commandes du programme canal; de plus, le programme canal virtuel est connu à l'avance donc la taille du programme canal réel est elle aussi connue, ainsi que sa structure: *fig.3.3.(c)*. Etudions en détail chacun des ordres de ce programme :

1/ La première commande (SEEK) a, comme argument (2) où :

- C_1C_1 est le numéro de cylindre réel mis en cause par l'entrée sortie. Il est calculé en additionnant le numéro réel du 1er cylindre correspondant au mini disque, et le numéro de cylindre relatif donné par le programme canal standard CMS de la *fig. 3.3.(b)*).

- HH est le numéro de tête; c'est la copie de HH de la *fig.3.3.(b)*

Cette opération réalise la translation du cylindre propre au mini disque.

2/ La deuxième commande (SEARCH) a pour argument (3) qui est identique à l'argument initial (CCHHR de la *fig.3.3.(b)*). En effet le mini-disque a été structuré par CMS et doit être adressé par ses propres adresses.

3/ La troisième commande (TIC) est la commande standard de transfert dans un programme canal; elle est ici couplée au SEARCH.

4/ La cinquième commande (READ ou WRITE) est identique à celle du programme canal standard avec l'adresse de l'argument (ici la zone d'entrée-sortie A1) en mémoire réelle.

5/ La cinquième commande est utilisée si la zone de mémoire virtuelle met en jeux deux pages qui ne sont pas contiguës en mémoire réelle. Dans ce cas, la quatrième commande est alors affectée à la première partie de la zone (adresse A1) qui réside dans une page, et la cinquième commande est affectée à la deuxième partie de la zone qui réside dans l'autre page (adresse B1).

Le programme canal ainsi construit, est exécuté par le canal réel. Pendant toute la durée de transmission, l'environnement concerné est désactivé, et il n'est réactivé qu'à la fin de l'opération d'entrée-sortie.

Ces adjonctions à CP et à CMS, en vue de transformer une machine virtuelle en environnement a sensiblement amélioré les performances du système. Les mesures que nous avons effectuées portent essentiellement sur deux paramètres

fondamentaux :

1- Le temps CP : temps nécessaire à CP pour simuler la machine virtuelle.

2- Le temps problème : temps pendant lequel la machine est active, c'est-à-dire temps d'utilisation de l'unité centrale pour cette machine.

Dans le cas d'un assemblage :

La diminution d'un temps CP est en moyenne de 23 %.
La diminution du temps problème est très faible (2 %).
Ces pourcentages s'expliquent très simplement; en effet,

- Les 23 % ont été gagnés uniquement en supprimant le temps de compilation des programmes canaux.

- Les 2 % de diminution du temps problème sont dûs au fait que la fonction LOGSIO est plus simple à programmer qu'une fonction d'entrée-sortie classique.

Dans le cas d'une tâche ne faisant que des entrées-sorties sur disque (duplicata de fichiers disques) :

Les mesures ont montré une amélioration de 44 % du temps CP, et une amélioration de 5 % du temps problème.

Ces deux derniers chiffres confirment les affirmations que l'on a formulées auparavant.

3.4. T ESPACE OU DISQUE LOGIQUE TEMPORAIRE

Nous venons de définir une nouvelle fonction générale d'entrée-sortie adaptée à un environnement, et nous avons étudié son application pour un système particulier : le système CMS.

Nous devons remarquer auparavant que tout système demande un ou plusieurs espaces disques de grande taille pour une activité limitée dans le temps. Dans l'état actuel de CP, pour satisfaire ces demandes épisodiques, il est nécessaire d'affecter en permanence l'espace disque voulu. On déduit aisément que le coefficient moyen de remplissage de cet espace est quasiment nul. Ceci veut dire encore qu'une zone importante de l'ensemble disque disponible est mobilisée sans être utilisée.

Nous avons étudié et réalisé un mécanisme qui permet de n'effectuer aucune réservation de disque à priori, et qui, par déduction, attribue dynamiquement l'espace disque temporaire nécessaire : c'est le principe du T espace.

3.4.1. Définitions

On appelle T espace ou disque logique temporaire, un ensemble T(i,a(i)) d'éléments t tel que s'il existe i et a(i) l'on ait

$$T(i,a(i)) = \bigcup_{j = 0,p(a)} t_j$$

avec :

- t : élément de T est un cylindre de disque appartenant à l'espace disque de CP. Un T espace possède donc une structure analogue à la structure physique d'un disque, ce qui justifie le terme disque logique.

- i : identifie la machine virtuelle concernée. i n'a de valeur que si la machine est activable par le système, c'est-à-dire entre LOGIN et LOGOUT, ce qui justifie le terme temporaire.

- a(i) : est l'adresse du disque logique temporaire. Elle a la forme d'une adresse de disque ordinaire (mini disques) pour s'intégrer facilement dans la description d'une configuration virtuelle (*fig.3.4.(a)*). Une machine virtuelle peut donc avoir plusieurs T espaces.

- p(a) : représente la taille du T espace considéré. Elle est fonction de a; c'est-à-dire que chaque disque logique temporaire possède une taille propre, pré-définie.

Les T espaces n'ont d'existence que pour la durée d'une session; les différents cylindres t sont pris exclusivement dans l'espace disque de pagination de CP. C'est un sous-ensemble de E appelé ET.

Il faut donc pour chaque T espace, définir l'application ℓ de T(i,a(i)) dans ET

$$T(i,a(i)) \xrightarrow{\ell} ET$$

3.4.2. Définition de l'application ℓ .

Considérons i et a(i); cela revient à choisir un VDEVBLOK dans la configuration virtuelle associée à i.

La transformation ℓ est représentée par une table (TMPTBL) possédant p+1 entrées, décrite par la *figure 3.4.(b)*. Elle est liée au VDEVBLOK comme le montre la *figure 3.4.(c)*.

3.4.3. Mise en oeuvre par CP

La mise en oeuvre du mécanisme de T espace peut se découper en trois parties :

1/ Création du T espace
2/ Gestion et fonctionnement
3/ Suppression du T espace

3.4.3.1. Création du T espace

La création du T espace se fait lorsque une machine virtuelle se présente au système (LOGIN).

Au cours de la procédure de LOGIN, CP consulte le fichier général des utilisateurs nommé DIRECTORY, qui contient toutes les indications utiles sur la définition de la machine virtuelle qui vient de se présenter (nom, mot de passe, taille mémoire, définition de ses mini-disques etc...) Ce fichier contient aussi la définition des T espaces sous la forme suivante :

Adresse du disque logique : a(i).
Nombre de cylindres de ce disque : P(a).

Ces informations permettent à CP de créer le VDEVBLOCK et la TMPTBL associée. Le contenu de la TMPTBL est initialisé à 0. Sa longueur est (P(a)+2) entrées (*figure 3.4.(b)*).

3.4.3.2. Gestion et fonctionnement

Pour accéder à son T espace, l'environnement dispose de trois fonctions :

TSTATE (a,p)
LOGSIO (a,CAW)
TQUIT (a)

- TSTATE (a,p)

Cette fonction permet à un environnement de connaître la taille "p" du T espace dont il donne l'adresse "a".

Pour réaliser cette fonction CP sélectionne le VDEVBLOCK en fonction du paramètre "a". Si le VDEVBLOCK trouvé n'est pas un VDEVBLOCK de T espace, la fonction donne en retour un code erreur à l'environnement.

De la même façon, si "a" ne correspond pas à un VDEVBLOCK de la configuration virtuelle, la fonction donne également un code erreur.

Pour le cas où le VDEVBLOCK trouvé est bien associé à un T espace, la fonction indique à l'environnement la valeur du paramètre p (c'est la taille en cylindre du T espace).

- LOGSIO (a,CAW).

Cette fonction possède les mêmes caractéristiques que la fonction du même nom décrite au paragraphe 3.3.1. avec cependant quelques différences :

La transformation de l'argument de SEEK, placé en tête du programme canal par l'environnement, se fait en liaison avec la TMPTBL associée au VDEVBLOCK.

Le numéro de cylindre virtuel sert d'index dans la TMPTBL; si l'entrée sélectée est égale à 0, c'est que le cylindre correspondant n'a pas encore été affecté au T espace.

Dans ce cas, la fonction demande à CP un cylindre libre de l'espace disque ET. Le module de gestion de l'espace disque fournit à la fonction le numéro de cylindre alloué ainsi que l'adresse du bloc de contrôle qui décrit l'unité de disque réelle associée (RDEVBLOK).

Ces deux informations sont rangées dans la TMPTBL à l'endroit voulu. Nous sommes alors dans le cas où l'entrée sélectée n'est pas nulle (le cylindre ayant déjà été alloué au T espace).

Dans ce cas, le numéro de cylindre (1er octet de l'entrée de la table) est placé dans l'argument du SEEK, et le programme canal est initialisé pour le disque réel; en accord avec l'information ADREV(j) placée dans la deuxième partie de l'entrée de la TMPTBL.

- TQUIT (a)

Cette fonction permet à un environnement de libérer les cylindres occupés dans l'espace ET, par le T espace d'adresse "a".

Pour cela CP sélecte séquentiellement chaque entrée de la TMPTBL rend à l'espace disque de CP tous les cylindres occupés (entrées ≠ 0) et, les entrées de la TMPTBL sont remises à 0 au fur et à mesure. A la suite de l'application de cette fonction, le T espace est revenu à l'état initial.

3.4.3.3. Suppression du T espace.

Le T espace est détruit de façon systématique lorsque l'environnement quitte le système (LOGOUT). Cette destruction se traduit par l'application de la fonction TQUIT suivie d'une destruction des tables VDEVBLOK et TMPTBL.

3.4.4. Utilisation du T espace par le système CMS.

Le T espace est utilisé par CMS de la même manière que les disques ordinaires (mini disques).

Cependant nous avons ajouté à CMS quelques mécanismes particuliers qui tiennent compte de la qualité de disque temporaire.

a) Initialisation du disque logique temporaire.

A l'IPL de CMS, la procédure d'initialisation de ce système, s'assure qu'il existe un disque logique temporaire d'adresse 192. Pour cela, CMS demande par TSTATE(192) si l'unité virtuelle 192 existe ou non, ou, si l'unité virtuelle 192 n'est pas un T espace, l'initialisation du système se poursuit

sans changement par rapport à ce qu'elle est normalement. Si l'unité 192 est un T espace, CMS utilise le nombre de cylindres donné par TSTATE pour créer, en mémoire virtuelle, toutes les tables nécessaires à la gestion d'un disque classique. En particulier la table d'indicateurs qui reproduit l'image des blocs de 800 octets libres ou occupés. A cet instant, l'ensemble de ces tables est initialisé pour indiquer que le disque est vierge de toute information.

b) Fonctionnement normal en cours de travail

Comme nous l'avons souligné précédemment, le disque logique temporaire est géré comme un disque ordinaire, puisque CMS a été adapté pour utiliser la gestion LOGSIO (a,CAW), (§ 3.3.1.).

Cependant, chaque fois que le système CMS va se mettre en attente de commande (soit sur l'achèvement de l'activité dû à une commande précédente, soit à la fin du déroulement d'un programme utilisateur), il active la fonction TQUIT (192) et réinitialise le disque Logique Temporaire. Cette façon de procéder a été ajoutée dans le but d'éviter une saturation de l'espace disque de CP.

Nous avons, de plus, rendu automatique l'utilisation du T espace comme support des fichiers utilitaires des compilateurs et assembleurs du système, dans le cas où l'environnement possède un T espace.

3.4.5. Remarques

Le mécanisme de T espace donne actuellement un gain d'espace disque appréciable. En effet les mini-disques, préalablement définis pour chaque machine virtuelle, devaient posséder une taille suffisante pour ranger tous les fichiers permanents de l'utilisateur. Mais aussi, les fichiers intermédiaires (utilitaires) des compilateurs et assembleurs. La place, ainsi perdue, n'est pas négligeable. Nos mesures ont montré que pour l'assemblage d'un fichier de 3000 cartes, de complexité moyenne pour l'assembleur (peu de macro-instructions), la taille occupée par les fichiers utilitaires au cours de l'assemblage est de 9 cylindres!

On peut donc estimer à près de 400, le nombre de cylindres ainsi "gaspillés" dans le cadre du système CP-CMS à l'IMAG. Nous sommes actuellement sur le point de mettre en application l'utilisation systématique du T espace, et de restructurer l'espace disque affecté aux utilisateurs. La récupération de l'espace ainsi perdu, nous permet d'envisager raisonnablement les chiffres suivants :

-180 cylindres, réservés aux T espaces, permettront de faire face aux fortes demandes dans le cas le plus défavorable.

- 220 cylindres, récupérés, permettront de ranger les fichiers permanents de 30 nouveaux utilisateurs en moyenne.

3.5. F ESPACE

Les expériences précédentes nous ont conduit tout naturellement à étudier un nouveau principe général pour la gestion des fichiers des utilisateurs. Nous nous sommes interrogés pour savoir s'il est envisageable de considérer une unité d'enregistrements pour tous les fichiers de telle sorte qu'il soit possible de généraliser à tout niveau un programme type d'entrée-sortie. Un autre aspect, non moins intéressant, nous a permis de définir les règles et les principes du partage des informations entre plusieurs utilisateurs. Ainsi nous sommes en train de préciser la structure générale des fichiers et les différentes possibilités d'accès en cherchant à utiliser le mécanisme de la pagination de CP.

Cette application est donc à l'étude; certaines parties non encore figées sont donc volontairement omises dans la description, mais sont exposées au chapitre suivant.

3.5.1. Définitions

On appelle F espace, un ensemble F(i,a(i)) de pages f tel que :

$$F(i,a(i)) = \bigcup_{j = 1,p(a)} f_j$$

où a(i) est l'adresse de l'unité F espace représentée avec la convention classique : CUU (Canal Unité de Contrôle Unité), et p(a) est la taille du T espace d'adresse "a" de l'utilisateur "i".

Les pages f des F espaces se trouvent sur supports externes dans l'espace adressable par CP. Cet espace de CP comprend l'espace E destiné à la pagination, et l'espace EU, ensemble de l'espace attribué aux utilisateurs pour ranger leurs fichiers permanents.

Un F espace est découpé en blocs de pages contiguës appelés F segments. Un F segment réside entièrement sur une même unité physique. Le nombre de pages d'un F segment est fixe au cours du temps, mais peut être variable d'un F segment à l'autre.

Ce découpage de F espace est arbitraire, et n'est pas connu de l'environnement qui l'utilise; il permet de réaliser la transformation m telle que :

$$F(i,a(i)) \xrightarrow{m} E \cup EU$$

3.5.2. Réalisation de la transformation - m

La transformation "m" est matérialisée par une table des F segments ou FTABLE. En gardant les mêmes notations que dans les paragraphes précédents, on peut dire que fixer "i" et a(i) revient à sélectionner un VDEVBLOK (table qui définit une unité virtuelle).

La description et les liens d'un VDEVBLOK et d'un FTABLE sont donnés par la *figure 3.5.(a)*.

La transformation a pour argument le numéro de page x

du F espace dont on veut chercher le transformé dans E ∪ EU ($0 \leq x < p(a)$). Le transformé doit définir l'unité réelle où réside physiquement la page; ainsi que son adresse sur cette unité.

Une exploration séquentielle de la FTABLE permet de trouver le F segment Fy tel que :

$$\sum_{k = 1,y-1} \text{NBPSEG}(k) \quad < x \leq \quad \sum_{k = 1,y} \text{NBPSEG}(k)$$

Le F segment Fy définit directement l'unité réelle associée par RDEVADDR(y).

L'adresse z de la page sur cette unité physique se calcule facilement par :

$$z = \text{NOPSEG}(y) + (x - \sum_{k = 1,y-1} \text{NBPSEG}(k))$$

3.5.3. Partage des F espaces.

Les FTABLE qui définissent les différents F espaces initialisés, sont chainées les unes aux autres, et le début de chaine est repéré par un pointeur situé à une adresse fixe dans CP. Les différents VDEVBLOK, qui peuvent se partager un F espace, sont tous reliés à la FTABLE correspondante par l'intermédiaire de chaque VPNTREAL. Le compteur SHARECNT contient le nombre de VDEVBLOK qui partagent la FTABLE (*figure 3.5.(b)*).

Actuellement, le partage d'une FTABLE n'est autorisé que pour des F-espaces qui sont accédés en Lecture seulement.

3.5.4. Accès au F-Espace :

Un environnement peut utiliser un F-Espace désiré grâce à trois fonctions élémentaires que nous avons ajoutées à CP.

1 - FSTATE (a,p)

La fonction donne au paramètre "p" la valeur de la taille en pages du F-Espace d'adresse "a".

2 - FWRITE (a,v,f,p)

Cette fonction copie un nombre "p" de pages situées en mémoire virtuelle à partir de la page d'adresse "v" sur le F-Espace d'adresse "a", à partir de l'adresse de la page "f" du F-Espace.

3 - FREAD (a,v,f,p)

Cette fonction possède des paramètres qui ont les mêmes définitions que FWRITE, mais seul le sens de transfert est modifié.

3.5.5. Une Application du F-Espace au système CMS : Le SESPACE.

Le principe des F-Espaces est en cours d'application pour le système CMS. Nous étudions la possibilité d'utiliser le F-Espace pour servir de support aux composants non résidents en mémoire virtuelle.

Actuellement le mécanisme de CMS qui gère les composants, utilise des fichiers d'un type particulier, appelés modules et ces fichiers sont placés sur le mini-disque spécial qui retient le système CMS. Un module est une image-mémoire du composant considéré (ou d'une partie du composant dans le cas où il possède une structure d'"overlay"). La partie de mémoire virtuelle ainsi recopiée sur disque est liée rigidement à son emplacement initial et elle n'est pas translatable; ceci est dû au fait que toutes les adresses sont résolues.

On appelle S-Espace un F-Espace qui contient des images-mémoire des composants de CMS. Ces Images-mémoire placées sur le S-Espace sont appelées SDULES.

Actuellement dans CMS, le changement d'un fichier de type module, se fait par l'intermédiaire de la gestion de fichiers standard de CMS. Dans le cas de SDULES, le changement est réalisé par la fonction FREAD appliquée au S-Espace.

3.5.6. Création du S-Espace.

Le S-Espace doit donc contenir des images de mémoire-virtuelle. Pour le créer, un programme utilitaire de CMS, recopie dans un ordre pré-établi (ordre décroissant de fréquence d'utilisation), les fichiers de type module résidents sur le disque système, dans le S-Espace grâce à la fonction FWRITE *(fig.3.5.(c))*. Il est important de noter que le S-Espace ainsi structuré n'est pas modifiable; en conséquence si l'on désire changer un SDULE du S-Espace, il faut recréer l'ensemble des SDULES sur le S-Espace.

En parallèle avec cette recopie le programme utilitaire dresse la table des SDULES. Cette table est copiée sur le disque système, sous forme d'un fichier de noms INDEX, et de type SDULES. Ce fichier constitue le catalogue du S-Espace. Un enregistrement du catalogue décrit un SDULE. Il comprend :

- Le nom du SDULE
- Le numéro de la 1ère page du SDULE dans le S-Espace (adresse du SDULE dans le S-Espace).
- La longueur du SDULE
- L'adresse du point d'entrée (adresse mémoire).

3.5.7. Utilisation du S-Espace

A l'initialisation du système CMS, le catalogue du S-Espace est installé en mémoire virtuelle. En cours de fonctionnement, chaque fois qu'il faut charger un composant,CMS examine la Table des SDULES pour savoir si le composant cherché est dans le S-Espace. Dans l'affirmative, le SDULE est chargé

par une seule fonction VREAD. Notons que cette fonction n'effectue pas d'entrée-sortie directement; elle modifie les tables (SWPTABLE, PAGTABLE) représentant la mémoire virtuelle, pour qu'à la première référence, chaque page intéressée soit lue en mémoire par le mécanisme de pagination de CP (cf.2.1.4.). Dans la négative, le composant mis en cause est chargé à partir du disque système selon la procédure standard de CMS.

En pratique, pour accélérer le chargement des SDULES, une partie du S-Espace est recopiée automatiquement sur Tambour à l'initialisation de CP. La taille de la partie ainsi recopiée sur tambour est variable en fonction des besoins. Elle est fixée par l'administration du système. Du fait de l'ordre dans lequel sont rangés les SDULES sur le S-Espace, les composants de CMS les plus utilisés sont résidents sur tambour. Nous avons ainsi créé un F-Espace à supports physiques hétérogènes. Cette structure hétérogène du support du S-Espace, est inconnue de CMS.

Cette dernière expérience est une synthèse des Expériences C-Espace et T-Espace. Elle devrait amener des améliorations de performance assez importantes.

Cette affirmation est purement subjective car nous n'avons pas encore mesuré les conséquences d'une telle application. Toutefois, les déductions que nous pouvons faire à la suite des expériences C-Espace et T-Espace, nous permettent d'être raisonnablement optimistes.

Cependant nous ne pouvons ignorer les interférences de cette réalisation avec les autres parties du système, ce qui pourrait faire apparaître un phénomène de saturation de certains organes physiques du système (canaux d'entrées-sorties surtout) conduisant par là-même à un déséquilibre du système.

CHAPITRE 4

LES SYSTEMES CP+ ET CMS+

Nous avons énoncé dans l'introduction,ce que nous comptions réaliser grâce aux systèmes CP+ et CMS+. Nous avons vu dans le chapitre précédent les essais préliminaires que nous avons pu faire ainsi que les conclusions que nous en avons tiré. Dans ce chapitre, nous ne prétendons pas donner une description complète des systèmes CP+ et CMS+, mais nous en exposerons les principes de base, d'une part sur l'utilisation de la segmentation et d'autre part, sur la structure de l'espace disque. Nous présenterons aussi l'architecture de la gestion de fichiers de CMS+.

4.1. M-ESPACE , MEMOIRES VIRTUELLES DISCONTINUES.

4.1.1. Définitions

Une mémoire virtuelle générée par CP est, comme nous l'avons vu (§ 2.1.4.), un espace continu qui est en fait une suite continue d'adresses virtuelles. Si la taille d'une mémoire virtuelle est de N pages, l'adressage par octet est possible de 0 à N x 4096 - 1. Avec une telle continuité dans l'adressage CP n'utilise pas explicitement la notion de segment telle qu'elle

existe sur le 360-67 (§ 2.1.3.).

Considérons une suite de pages réparties en nombres variables à l'intérieur de plusieurs segments; par déduction la taille des tables de pages associées est variable et par conséquent une adresse ne peut plus être quelconque entre 0 et la plus grande adresse dans le dernier segment. Nous avons fait apparaître une discontinuité dans l'adressage directement dérivée de la structure ainsi définie. Cette structure est appelée M-Espace et facilite le partage de mémoire virtuelle entre plusieurs environnements.

Nous pouvons alors donner la définition :

Un M-Espace est une suite de segments S_j *(du type de ceux définis sur le 36-67) avec* $0 \leq J \leq$ Smax *où* Smax *est le nombre maximum de segments autorisé par construction (sur le 360-67* Smax = 15*). Par construction aussi, il est nécessaire d'attacher à chaque segment, une table de pages (PAGTABLE). Chaque table de pages* P_j *associée au segment* S_j *a une taille* t_j *avec* $0 \leq t_j \leq$ tmax *où tmax est la taille maximale d'une table de page (255 dans le cas du 360-67).*

4.1.2. Gestion et partage du M-Espace

4.1.2.1. Gestion physique par CP+

Lorsqu'un environnement, possédant une mémoire virtuelle du type M-Espace, joint le système CP+, ce dernier doit construire les tables associées à la structure de M-Espace; il est donc nécessaire de connaître la taille de chaque segment composant sa mémoire. L'option qui consiste à limiter chaque segment à une taille maximale statique (modifiable par l'administration du système) a pour but d'éviter qu'un environnement, à la suite d'un fonctionnement erratique, ne sature la totalité de l'espace, à savoir 16 Méga-Octets sur le 360-67. Pour ce faire, nous avons décidé de reproduire la structure générale des tables actuelles de CP c'est-à-dire :

La Table des segments (SEGTABLE) et le couple Table de pages, Table de SWAP (PAGTABLE, SWPTABLE) associé à chaque segment *(fig.4.1.(a))*.

Rappelons brièvement ce qui se produit, lorsqu'une machine virtuelle fait référence à une adresse quelconque (cf.2.1.4.). Deux cas seulement sont à envisager :

1°/ L'adresse virtuelle a un transformé dans RO et la traduction dynamique des adresses résoud les problèmes d'adressage sans que CP intervienne.

2°/ L'adresse virtuelle n'a pas de transformé dans RO auquel cas il se produit une interruption de type "Page Manquante" et deux éventualités doivent être examinées par CP :

a) L'adresse est supérieure à l'adresse maximale de la mémoire virtuelle; CP doit donc simuler pour la machine virtuelle en cause, une interruption de type "ADRESSING".

b) L'adresse est dans les limites de la mémoire virtuelle; elle a obligatoirement un transformé dans E (adressage continu) et CP doit appliquer la transformation g.

Etudions alors ce qui se produit avec un M-Espace. Les deux éventualités précédentes sont identiques, et le premier cas est traité de la même manière. Par contre, le deuxième cas est traité différemment. En effet,CP+ doit simuler pour l'environnement une interruption de type "ADRESSING".

a) Si le segment S_j associé à l'adresse générée est manquant, c'est-à-dire qu'il n'existe pas de Table de pages associées ou la taille du segment est nulle (segment invalide *fig. 4.1.(a)*).

b) l'adresse générée fait référence à une page du segment S_j, extérieure aux limites maximales fixées à l'initialisation (adressage d'un "trou" du M-Espace).

Dans les autres cas, on est certain qu'il existe un transformé de l'adresse dans E et CP+ applique la fonction g.

4.1.2.2. Partage.

Etudions d'abord le partage de parties de mémoires virtuelles tel qu'il est réalisé dans CP.

Soit un ensemble de "p" pages contiguës dans une mémoire virtuelle, à partager entre "q" machines différentes.

La solution actuelle de CP *(figure 4.1.(b))* consiste à verrouiller les "p" pages en mémoire réelle, et la même suite de "p" pointeurs est alors dupliquée dans les "q" tables de pages correspondantes. La partie de SWPTABLE associée est, elle aussi, dupliquée q fois. Si, avec une telle organisation, l'on ne souhaite pas verrouiller les pages partagées, il faut maintenir une liste de pointeurs indiquant toutes les entrées des PAGTABLE à valider ou invalider, lorsqu'un mouvement de page partagée est réalisé.

Cette liste permet aussi de mettre à jour les parties de SWPTABLE dupliquées, afin de conserver l'homogénéité des adresses sur support externe, de toutes les pages partagées.

Supposons maintenant que, dans le cadre de M espaces, les "p" pages à partager aient été isolées dans un segment particulier des M-Espaces concernés. Le partage devient alors particulièrement simple. En effet :

Le couple, PGTABLE - SWPTABLE, décrivant le segment partageable, est unique pour tous les M-Espaces, et l'entrée correspondante, de chacune des "q" tables de segments des M-Espaces, associée au segment partageable, repère la même table de pages (*figure 4.1.(c)*).Le segment partageable peut donc être facilement paginé, et cette nouvelle possibilité permet de généraliser l'emploi de ce type de partage, sans pour cela encombrer la mémoire réelle. Comme nous allons le voir, cette particularité est utilisée au maximum par CMS+.

4.1.3. Emploi du M-Espace par CMS+

Lorsqu'un utilisateur joint CMS+, CP+ recherche , dans un fichier (DIRECTORY Ref.C_3), la composition initiale de l'environnement à créer. Pour chaque utilisateur, une entrée appelée "CORE" précise la quantité des mémoires potentiellement utilisable à l'intérieur des différents segments.

Le M-Espace ainsi créé, est ensuite pré-initialisé; ceci consiste à donner une valeur initiale à chaque page. Lorsqu'une page est privée (c'est à-dire appartient à un seul utilisateur), elle est initialisée à l'aide de la page identiquement nulle (Z) - voir 2.1.4. Lorsqu'une page est partagée, elle est obligatoirement élément d'un segment qui est lui-même partagé : CP+ recherche alors la table de pages déjà existante qui décrit ce segment et l'attache au M-Espace en cours de création.

Dans la structure ainsi définie, CMS+ affecte à chaque segment un rôle particulier :

a) segment 0 : C'est la zone qui contient la mémoire libre et le code non-réentrant de CMS+.

Le segment 0 du M-Espace est un segment privilégié imposé par la constitution de la machine. En effet, il contient la page 0 qui, elle-même contient tous les mots d'état programme (PSW) nécessaires à la gestion des interruptions. De plus, cette page est la seule qui puisse être adressée sans registre de base. Ces contraintes imposent donc de ne placer dans le segment 0, que des informations non-partageables entre plusieurs environnements.

Le segment 0 comprend aussi un emplacement pour les programmes utilitaires du système qui ne sont pas résidents.

b) Segment 1 : Cette zone contient le code réentrant et partageable de CMS+. Le segment 1 est réservé à la partie de CMS+ qui est réentrante. Ce segment est partagé par tous les utilisateurs de CMS+, ce qui veut dire que les tables qui décrivent ce segment, PAGTABLE, SWPTABLE, sont communes à tous les M-Espaces qui utilisent CMS+.

Ce code de CMS+ qui se trouve dans ce segment, acquiert la mémoire libre nécessaire à sa réentrance dans le segment 0.

c) Segment 2 : C'est la zone de mémoire libre privée attachée au segment 3.

Tout comme la suite segment 0, segment 1 représente la mémoire libre, suivie du code réentrant qui l'utilise, CMS+ utilise le segment 2 comme mémoire libre (donc non partageable), associée au segment 3.

d) Segment 3 :c'est une zone partagée par un sous-ensemble d'utilisateurs. Ce segment joue, pour un sous-ensemble, le même rôle que le segment 1 pour la totalité des utilisateurs de CMS+.

Ce segment contient donc des programmes écrits par un utilisateur, pour être partagés par plusieurs utilisateurs; ce peut être, par exemple, un sous-système qui utilise les facilités de CMS+ (banque de donnée, enseignement programmé etc...).

e) segment 4 et 5 : Ce sont les zones privées. Chacun de ces deux segments est une zone de travail de l'utilisateur du M-Espace. Chacun d'eux contient sa propre mémoire libre.

Cette duplication de deux entités équivalentes (segments contenant sa propre mémoire libre), permet de faire cohabiter, au sein d'un même espace virtuel, deux programmes incompatibles dans un même segment; par exemple, un programme utilisateur chargé dans le segment 4, pourra appeler un compilateur, chargé dans le segment 5. Les mémoires libres des deux segments étant distinctes, il n'y a pas possibilité d'interférence néfaste entre les deux programmes.

f) segments 6 à 14 : l'ensemble de ces segments est réservé à la gestion de fichiers de CMS+, pour y ranger les descripteurs de fichiers.

Nous y reviendrons, plus en détail, dans la partie 4.2.

g) Segment 15 : C'est une zone privilégiée; elle est attribuée au système de mise au point SPY développé à l'IMAG (Ref.L2). Ce segment est entièrement réservé à ce système.

REMARQUE : Le partage de programmes entre différents utilisateurs (segment 1 et 3 par exemple) se fait toujours en utilisant la même adresse dans les différents M-Espaces. Cette méthode nous semble suffisamment riche et très simple à mettre en oeuvre par rapport au partage avec adresses distinctes qui implique, en particulier, une importante fragmentation. En effet, il convient d'aligner chaque procédure partagée sur une frontière de page et il est indispensable d'utiliser un mécanisme de liaison spécifique analogue aux "PSECT" de TSS (Ref.T1).

Les programmes non partagés,qui résident dans un F-Espace et qui sont utilisés fréquemment, sont conservés sous forme d'images de la mémoire. De cette façon leur changement dans le M-Espace est immédiat puisqu'il suffit, en effet, de modifier seulement certaines adresses dans les SWPTABLE.

4.1.4. Mécanismes de protection dans CMS+

4.1.4.1. Utilisation du mode de fonctionnement du PSW

Les mécanismes de protection disponibles par construction, sur la machine, vont être utilisés par CMS+, ce qui n'était pas le cas de CMS. En effet nous voulons protéger certaines zones de mémoire contre une destruction possible et même, dans certains cas, contre la lecture de leur contenu, par l'utilisateur. Ceci impose non seulement d'employer la protection mémoire, mais encore d'empêcher l'utilisateur de changer les clés du bloc mémoire, du PSW ou du CAW. En conséquence, un programme utilisateur, ne pourra pas fonctionner

en mode superviseur.

Détaillons quelque peu les différentes catégories de programmes qui se partagent l'unité centrale. Nous trouvons :

- CP+ qui actif dans un M-Espace, doit être par définition en mode superviseur réel puisque telle est sa fonction.

- CMS+ qui actif dans un M-Espace, doit être par définition, en mode programme réel.

Or CMS+ se compose de deux parties : Le superviseur que nous appelons SCMS+; et la partie utilisateur que nous nommons UCMS+; SCMS+ doit donc avoir un mode dit "Pseudo-Superviseur" pour lequel CP+ simule en particulier toutes les instructions privilégiées. Quant à UCMS+, il ne peut fonctionner que dans le mode dit "Pseudo-Programme" où toute instruction privilégiée provenant de UCMS+ conduit CP + à réfléchir à SCMS+ une interruption-Programme (*fig.4.1.(d)*).

4.1.4.2. Utilisation des clés de protection mémoire.

- L'espace d'adressage de CMS+ est disjoint de celui de CP+ c'est à-dire que CMS+ dispose, pour lui seul, d'un M-Espace, et ainsi, la protection de CP+ est totale.

- La partie SCMS+ réside dans une zone affectée de la clé de protection 0. L'environnement CMS+ fonctionne donc dans cette partie avec la clé 0 dans le PSW.

- La partie UCMS+ utilise d'une manière analogue, la clé de protection 1.

De plus, SCMS+ sera muni du "Fetch-protect" qui est un dispositif de protection mémoire que possède le 360-67 et qui implique que le contenu d'une zone

ainsi protégée, ne peut être référencée (lu ou écrit) uniquement lorsque la clé du PSW est équivalente à celle de la zone (0 pour SCMS+). Cette précaution est prise pour rendre le système imperméable à toute forme d'indiscrétion des utilisateurs.

L'emploi de la clé 0 dans le PSW ou le CAW qui est en fait l'absence de protection, peut être discutable, même à l'intérieur de SCMS+. L'introduction d'une autre clé pour SCMS+, 2 par exemple, imposerait toutefois le recours à la clé 0 pendant de mêmes périodes: Par exemple, pour ranger des résultats de fonctions dans la zone UCMS+. Cette méthode qui superposerait deux niveaux de protection dans SCMS+ n'a pas été retenue en première étude.

REMARQUE : L'examen du couple "clé,mode" du PSW réel, permet de déterminer sans ambiguïté l'état de CP+ et de CMS+ dans le cas où CMS+ est le système actif sous CP+ *(fig.4.1.(e))*.

4.1.5. Conventions de liaison et gestion de la mémoire libre dans CMS+.

4.1.5.1. Conventions de liaisons entre les différents niveaux.

a) SCMS+ communique avec CP+ grâce à l'instruction "DIAGNOSE", selon un processus défini par CP-67 version 3.0.(Réf.C4). C'est par ce moyen équivalent à une instruction "SVC" au 2ème niveau, que sont précisées en particulier les fonctions d'entrées-sorties définies au paragraphe 3.6.

b) UCMS+ communique avec SCMS+ à l'aide d'un appel spécifique et unique (SVC 202). Dans ce cas, le registre général 1 repère une liste de paramètres dont le premier définit le nom du module de SCMS+ concerné. La suite des paramètres dépend seulement du module appelé.

c) A l'intérieur de SCMS+, les composants communiquant entre eux, par l'instruction "BALR 14,15" avec des zones de sauvegarde du même type que celles utilisées en OS-360. Ces zones sont, soit définies comme éléments statiques dans le segment 0, soit acquises dynamiquement à partir de la mémoire libre du segment 0.

4.1.5.2. Gestion de la mémoire libre du M-Espace.

Il existe un module Unique "FREEMAIN, GETMAIN" pour l'ensemble SCMS+ et UCMS+. Ce module doit connaître le numéro du segment dans lequel la place libre doit être réservée ou rendue. Si le numéro de segment n'est pas précisé, la gestion de mémoire libre, utilise par défaut le segment contenant le code qui a émis la demande.

On remarquera que lorsqu'un bloc est rendu à la mémoire libre et que ce bloc complète une ou plusieurs pages, SCMS+ en informe CP+ selon le mécanisme détaillé au paragraphe 3.1.

Les segments partagés en lecture-seulement, n'ont pas de mémoire libre.

4.1.6. Partage en Lecture-Ecriture.

Le partage de parties de M-Espace en lecture et écriture, par deux ou plusieurs environnements est relativement complexe à réaliser. En effet, si par exemple deux environnements se partagent un même segment, ils se comportent pour ce segment comme un "bi-processeur"; par contre pour tous les segments qu'ils n'ont pas en commun, ils se comportent domme deux "processeurs" distincts. Les solutions au problème posé par le partage sont nombreuses et actuellement, nous n'avons pas fixé notre choix de manière précise. Nous avons pourtant établi

un classement entre les qualités que devra posséder la solution choisie. Nous tiendrons compte: en priorité de la sécurité d'emploi, ensuite de la facilité de programmation avec le souci de conserver des performances acceptables, et enfin seulement de la facilité d'utilisation du mécanisme choisi.

Il sera bien sûr fait usage de sémaphores (Ref.D8) qui constituent une solution sûre pour réaliser le partage de ressources entre plusieurs processus parallèles. Nous aurons donc à prévoir des files d'attentes sur chaque ressource partagée, pour permettre de réactiver les environnements en attente.

Le problème actuel est de savoir si nous devons réaliser un verrouillage des M-Espaces globalement au niveau du segment ou si nous devons le réaliser plus sélectivement au niveau de la page intéressée. Cette dernière solution évite de bloquer inutilement des environnements, mais nécessite une gestion plus complexe des sémaphores et des files d'attente. De plus, l'environnement CMS+ est conversationnel de ce fait, faut-il le mettre systématiquement en attente quand il demande l'accès à une ressource déjà occupée, ou faut-il au préalable demander à son utilisateur (assis devant son Terminal), s'il désire attendre ou non ?

Nous ne pouvons pas actuellement répondre de manière précise à ces questions, aussi nous ne dirons rien de plus sur ce sujet. Par contre, l'état de nos travaux nous permet de décrire comment nous avons utilisé, dans le cadre de CMS+, les M-Espaces comme support du catalogue des fichiers résidents sur un F-Espace. C'est cette organisation que nous allons décrire dans ce qui suit.

4.2. F-ESPACES

4.2.1. Rappels - Définitions complémentaires

Nous ne reviendrons pas sur la définition d'un F-Espace qui a été donnée au paragraphe 3.5. Rappelons cependant qu'un F-Espace est un ensemble de segments composés de pages. Le nombre de pages par segment, ainsi que le nombre de segments, sont fixé statiquement et modifiables par l'intervention de l'administrateur du système. Un utilisateur peut posséder un ou plusieurs F-Espaces, chacun d'eux constituant une entité indépendante, et autonome. Réciproquement, un F-Espace peut être partagé par un nombre quelconque d'utilisateurs, en lecture ou en écriture. Pour CMS+, un F-Espace, se comporte comme un ensemble de pages logiquement contiguës, qui contient un nombre plus ou moins grand de fichiers. La gestion de ceux-ci est à la charge de CMS+. Grâce à ce mécanisme de F-Espace, il devient possible de conserver des fichiers sur de vastes espaces virtuels définis sur plusieurs unités physiques, puisque rien ne s'oppose à ce que les différents segments, résident sur des unités physiques différentes. Ainsi, si l'on exclut le cas où l'on accepte un partage entre utilisateurs, chaque F-Espace constitue, par construction de CP+, un espace virtuel ayant un seul propriétaire. Ce dernier, ne peut accéder aux fichiers des autres utilisateurs, ce qui accroit la sécurité dans le délicat problème de la protection des informations.

Lorsque le F-Espace se réduit à un seul segment on retrouve sensiblement la notion de Mini-Disque. Il y a toutefois une différence fondamentale: la notion de caractéristiques physiques de l'unité (piste, cylindre, etc...) a disparu; l'environnement demande la Nième page du F-Espace et CP+ a la responsabilité d'assurer la conversion d'adresse grâce à la FTABLE (Cf.3.5.) et d'amener la page. La phase de compilation des programmes canaux a complètement disparu.

4.2.2. Définition et création des F-Espaces par CP+.

Le fichier de CP+ appelé "DIRECTORY" contient, pour chaque utilisateur susceptible d'utiliser le système, la liste des F-Espaces qui lui appartiennent. A chaque F-Espace, est associée une suite ordonnée de segments définie ici comme une suite de cylindres contigus qui contiennent chacun un nombre entier de pages : 30.

L'exemple de la *figure 4.2.1.(a)* définit un F-Espace d'adresse a=191 composé de 3 segments, de taille 5,10 et 17 cylindres. Le F-Espace considéré contient donc N=(5+10+17)*30 = 960 pages.

Lorsqu'un utilisateur joint le système, CP+ construit à partir du DIRECTORY, la FTABLE associée (Cf.paragraphe 3.5.). Parallèllement à cette construction, CP+ vérifie que les unités physiques mentionnées sont effectivement accessibles. Si ce n'est pas le cas, le F-Espace tout entier est considéré comme Non-Disponible.

A la suite de ces opérations préliminaires un environnement peut accéder aux F-Espaces à l'aide des opérations d'entrée-sortie qui se font grâce aux fonctions FREAD, FWRITE, FSTATE définies au paragraphe 3.5. Suivant la définition de ces fonctions, l'environnement qui ignore la structure du F-Espace, adresse celui-ci uniquement par le numéro des pages qui lui sont nécessaires.

4.3. STRUCTURE ET GESTION DES F-ESPACES UTILISEE PAR L'ENVIRONNEMENT CMS+: Gestion de fichier.

Pour CMS+, un F-Espace est un lot de pages contiguës, adressable de 0 à p-1, si p est le nombre de pages du F-Espace.

Cette suite de pages sert de support aux informations que l'utilisateur (propriétaire du F-Espace) désire conserver. Ces informations doivent être facilement accessibles, et leur classement par fichiers est à la charge de CMS+ qui fournit à l'utilisateur certaines facilités que nous allons étudier.

4.3.1. Description externe de la gestion de fichiers de CMS+.

4.3.1.1. Noms des fichiers - Le catalogue.

Nous avons décidé de créer plusieurs niveaux d'index pour définir un fichier (5 ou 6 vraissemblablement). Cette organisation donne à l'utilisateur plus de souplesse et de facilité pour classer ses fichiers par centre d'intérêt. Et ainsi le catalogue prend une structure d'arbre classique.

Un fichier est nommé par une suite d'identificateurs qui décrit le cheminement dans l'arbre (*figure 4.3.1.(a)*) et, deux identificateurs successifs sont séparés par le caractère"."

On appelle "NOM COMPLET" d'un fichier, la suite d'identificateurs nécessaire pour déterminer sans ambiguïté le bloc terminal de l'arbre, qui décrit le fichier proprement dit, à partir de la racine de l'arbre.

Exemple de nom complet sur la *figure 4.3.1.(a)*

RDTAPE . SOURCE . TAP5 . SYSIN

Ce nom complet identifie sans ambiguité, la famille de l'arbre marquée (1) sur la figure.

Cette façon de nommer les fichiers, si elle est très puissante en soi, est aussi mal commode à utiliser de façon systématique. Pour l'alléger, nous introduisons dans CMS+ le concept de "NOM RELATIF". Auparavant, nous définissons le "NOEUD PREFIXE", qui est le nom complet d'un "noeud" de l'arbre (élément non terminal) appelé aussi"niveau". On pourra nommer par exemple :

NOEUD - PREFIXE RDTAPE - SOURCE

Si nous écrivons alors

.TAP5 . SYSIN

nous dirons que l'on utilise le "NOM RELATIF" vis-à-vis du NOEUD - PREFIXE imposé.

Cette manière d'identifier un fichier est reconnue par CMS+,

grâce à la présence du caractère spécial "." qui apparaît au début du nom.

Le système préfixe alors automatiquement le nom relatif par le NOEUD-PREFIXE, pour obtenir le "NOM-COMPLET". Avec les exemples précédents, nous obtenons :

RDTAPE.SOURCE.TAP5.SYSIN

A un instant donné, CMS+ ne connait qu'un seul noeud préfixe; celui-ci peut changer de valeur si l'utilisateur reémet la commande "NOEUD-PREFIXE". A tout instant, et ceci quel que soit le noeud préfixe, il est possible évidemment d'utiliser un nom complet, pour désigner un fichier qui n'est pas accessible à partir du dernier noeud préfixe défini. Citons par exemple RDTAPE.MODULE (*figure 4.3.1(a)*).

Avant de continuer, faisons une remarque sur le système CMS actuel qui définit les fichiers avec deux niveaux seulement : NOM et TYPE. Certains composants utilisent un TYPE particulier; citons entre autes, les assembleurs qui utilisent le TYPE : SYSIN, le compilateur FORTRAN, le TYPE : FORTRAN etc... Cette façon de procéder équivaut à identifier le genre d'informations que contient le fichier. Cet aspect très séduisant permet d'éviter de perdre du temps pour certaines erreurs grossières comme par exemple : confier un fichier contenant des ordres FORTRAN au Compilateur PL1.

Pour introduire cette technique dans CMS+, nous sommes amenés à définir deux nouveaux concepts :

Le "NOM-INCOMPLET" et le "SUFFIXE"

Un NOM-INCOMPLET est soit un NOM-COMPLET, soit un NOM-RELATIF dans lequel on omet le dernier élément, c'est-à-dire le nom de la feuille (élément terminal).

Un SUFFIXE est un nom de feuille précédé du caractère "." .

Ces deux notions réalisent dans CMS+ l'équivalent du TYPE de CMS. En effet, à l'appel de chaque composant, l'utilisateur précise uniquement le NOM-INCOMPLET du fichier qu'il désire voir traité. Chaque composant complète

suivant sa nature le nom-incomplet par le SUFFIXE qui lui est propre.

Considérons par exemple le fichier RDTAPE.SOURCE.TAP5.SYSIN, et l'assembleur de CMS+. Une commande telle que :

ASSEMBLE RDTAPE.SOURCE.TAPE5

entraine l'assemblage du fichier RDTAPE.SOURCE.TAPE5.SYSIN. Par contre, si l'on confie le même nom incomplet au compilateur Fortran par la commande :

FORTRAN RDTAPE.SOURCE.TAPE5

celui-ci cherche à compiler le fichier RDTAPE.SOURCE.TAPE5.FORTRAN, qui n'existe pas dans le catalogue et l'erreur est immédiatement détectée.

Donnons un nouvel exemple :

Considérons le NOEUD PREFIXE RDTAPE.SOURCE. Dans ce cas, on se contente de confier à CMS+ la commande :

ASSEMBLE.TAPE5 ou FORTRAN.CONVERT

L'assembleur traite le fichier.TAPE5.SYSIN, qui, complété à gauche par le système grâce au Noeud préfixe, devient : RDTAPE.SOURCE.TAPE5.SYSIN.
De même, le compilateur Fortran traite le fichier RDTAPE.SOURCE.CONVERT.FORTRAN.
Enfin, à la limite, on peut définir :

NEOUD PREFIXE RDTAPE.SOURCE.CONVERT

et demander la compilation du même fichier par la commande FORTRAN, ici simplifiée à l'extrême.

4.3.1.2. Indépendance des programmes vis à vis des fichiers.

Nous pouvons constater que, dans la plupart des programmes d'un utilisateur, il est fait usage de fichiers; pour accroître la souplesse de

traitement, nous donnons la possibilité de référencer un fichier par un nom intermédiaire quelconque. Ainsi pour la phase d'élaboration des programmes, le nom intermédiaire est considéré comme un paramètre formel. Par contre, c'est au moment de l'exécution seulement que nous appliquons une technique d'adressage indirect, qui met en correspondance le nom intermédiaire et le nom affectif du fichier mis en cause.

Le nom intermédiaire est formé d'un nom symbolique éventuellement complété par un suffixe si le programme ne traite qu'un ensemble particulier de fichiers.

Exemple : un programme ESSAI fait référence au fichier ENTREE-SYSIN, le nom symbolique est ENTREE, et le suffixe SYSIN.

Ainsi la suite de commande :

```
NOEUD-PREFIXE    RDTAPE . SOURCE
DECLARE          ENTREE=        .TAPE5
ESSAI
```

conduit à la prise en compte du fichier

RDTAPE . SOURCE	. TAPE5	. SYSIN
Noeud préfixe	subsitué à ENTREE	suffixe

au cours de l'exécution du programme ESSAI. Un tel mécanisme améliore très sensiblement l'accès aux différents fichiers et crée l'indépendance entre les fichiers et les programmes qui les utilisent.

REMARQUE : Un même programme, au cours de différentes exécutions, peut indifféremment se référer ainsi à plusieurs fichiers. Les unités d'entrées-sorties autres que les F-Espaces, sont considérées comme des fichiers ordinaires. Leurs noms symboliques sont réservés et ne rentrent pas dans le cadre des concepts de Noeud Préfixe et des suffixes. Ces noms sont :

TAPE1 à TAPE9	pour les dérouleurs de bande.
CONS	pour la console.
PUNCH1 à PUNCH9	pour les perforateurs de cartes.
READ1 à READ9	pour les lecteurs de cartes.
PRT1 à PRT9	pour les imprimantes

4.3.1.3. Partage d'un F-Espace entre plusieurs utilisateurs.

Un même F-Espace peut être déclaré dans la configuration de plusieurs utilisateurs. Il peut aussi dynamiquement être partagé en cours de travail par une commande de CP+ : "LINK". Lorsqu'un même F-Espace est partagé entre plusieurs utilisateurs, ceux-ci ont accès à tous les fichiers, sans autre contrôle que ceux précisés dans la définition du F-Espace, soit :

- "WRSHARE" accès en écriture permis si aucun autre utilisateur n'a déjà accès au F-Espace.

- "RDSHARE": accès en lecture uniquement (conditionné éventuellement par la connaissance d'un mot de passe).

- "WRMULT" : accès multiple en lecture et écriture (les verrouillages nécessaires étant à la charge de l'environnement). Toutefois, un utilisateur peut accéder à un sous-ensemble du F-Espace, qui ne lui appartient pas (après avoir, le cas échéant, fourni un mot de passe). En effet, à chaque noeud de l'arborescence, que constituent les noms de fichiers, peut être attachée une liste d'accès qui définit le nom des utilisateurs autorisés à suivre l'ensemble des chemins partant de ce noeud. Dans l'exemple de la *figure (4.3.1.(a))*, l'utilisateur CHARLIE a accès uniquement aux quatre fichiers appelés.

MEMOGSL . INTRO . PARAG1 . SCRIPT
MEMOGSL . INTRO . PARAG2 . SCRIPT
RDTAPE . SOURCE . CONVERT . FORTRAN
RDTAPE . SOURCE . CONVERT . TEXT

La liste d'accès précise aussi la condition d'accès (lecture seulement, écriture sans modification du catalogue, écriture avec modification du catalogue, accès sans contrôle).

Les listes d'accès permettent de contrôler l'emploi de la commande NOEUD-PREFIXE, qui dans le cas de ce type de partage, n'est acceptée que si le chemin précisé dans la commande comporte au moins une liste d'accès avec le nom de l'utilisateur. Sur l'exemple, les "NOEUD-PREFIXE" minimums utilisables par CHARLIE sont :

RDTAPE.SOURCE.CONVERT

et

MEMOGSL.INTRO.

Si le chemin suivi pour définir un"NOEUD-PREFIXE" comporte plusieurs types d'accès pour un même utilisateur, l'accès aux fichiers par nom relatif est fonction du privilège d'accès le plus restrictif.

4.3.1.4. F-Espaces Multi-utilisateurs.

Cette possibilité permet à un sous-ensemble d'utilisateurs de ranger tous leurs fichiers sur un même F-Espace. Pour ce faire, son propriétaire est le seul à posséder le F-Espace dans la configuration de son environnement Les noms des utilisateurs qui partagent ce F-Espace figurent dans la branche principale du catalogue (*figure 4.3.1.(c)*); et cette opération ne peut être réalisée que par la propriétaire du F-Espace.

Lorsqu'un des utilisateurs "locataire" désire utiliser ses fichiers, il émet la commande

SHARE "nom du propriétaire", "adresse du F-Espace pour le propriétaire", "adresse du F-Espace pour le locataire".

Dans notre exemple, CHARLIE donnera la commande :

SHARE ALPHA , 191 , 193.

Cette commande a pour effet d'affecter l'adresse 193 au F-Espace, pour CHARLIE. Le F-Espace origine est à l'adresse 191 pour ALPHA.

A la suite de cette commande , l'utilisateur CHARLIE nomme ses fichiers de façon normale. Cependant le système préfixe automatiquement tous les noms donnés par le nom de l'utilisateur. Par exemple, si l'utilisateur CHARLIE donne la commande :

NOEUD-PREFIXE RDTAPE.SOURCE,

le système concatène à son tour CHARLIE au noeud préfixe, et tout se passe alors comme si la commande était :

NOEUD-PREFIXE CHARLIE.RDTAPE.SOURCE

Ce mécanisme découpe le catalogue, donc le F-Espace, en autant de sous-arbres qu'il y a de "locataires".

4.3.1.5. Sauvegarde

Tous les fichiers qui sont modifiés ou créés au cours d'une session, sont marqués par l'environnement qui les utilise.

Lorsqu'un F-Espace est libéré, c'est-à-dire lorsque les utilisateurs ont quitté le système, ce F-Espace est lié automatiquement à un environnement privilégié qui procède à une sauvegarde sur bande de tous les fichiers marqués "modifiés" ou "créés". Cette sauvegarde se présente comme une opération de "spooling" lent, qui a lieu pendant le fonctionnement normal du système.

Si un utilisateur de ce F-Espace joint le système alors que l'opération de sauvegarde se déroule, il peut accéder à tous les fichiers suivant leurs privilèges; mais il ne peut accéder au fichier en cours de sauvegarde qu'en lecture seulemnt.

Enfin, une copie complète de chaque F-Espace est régulièrement effectuée (une fois par semaine par exemple) et alors les F-Espaces ne peuvent être accédés qu'en lecture pendant le temps de la sauvegarde.

4.3.2. Définition des sous-ensembles composant un F-Espace dans CMS+ *(fig.4.3.2.(a))*.

Un F-Espace sert de support aux informations composant les fichiers de l'utilisateur. La totalité de ces informations constitue le sous-ensemble I (Informations) du F-Espace. I est un lot de pages, lui-même structuré en fichiers décrits dans le sous-ensemble D (Descripteur) qui est aussi un ensem semble de pages disjoint de I. Le complément à F de I ∪ D constitue une provision de pages libre, dans laquelle la gestion de fichiers vient puiser, pour alimenter soit D, soit I. Le sous-ensemble de F appelé P(Provision) est aussi décrit dans D. Le Descripteur D constitue donc le sous-ensemble maitre de I et de P.

4.3.3. Structure du Descripteur D

4.3.3.1. Différentes parties du Descripteur *(fig.4.3.3.(a))*

Le descripteur est constitué de 4 parties :

- La partie C, qui donne la description de I, et qui constitue en fait le Catalogue des fichiers.
- la partie DP, qui décrit le sous-ensemble P.
- la partie UB qui donne la structure de l'espace libre du sous-ensemble D.

- la page zéro de D, que nous notons D(o), joue un rôle particulier que nous étudierons dans le paragraphe suivant. Nous devons préciser toutefois que, par construction, cette page zéro de D est aussi la page zéro de F.

Remarquons aussi que UB(Unused Blocks), est le complément à D de C ∪ DP ∪ D(o); elle constitue la zone libre de D indispensable à la gestion de C et DP.

4.3.3.2. La page zéro du Descripteur : D(o) *(fig.4.3.3.(b))*.

Cette page d'adresse fixe dans F (F(o)) contient la description de D dans F; elle est découpée en 1024 mots de 4 octets, et chaque mot de rang j de D(o) donne le numéro dans F de la page D(j). Par construction, les pages D(o) et D(1) sont toujours les pages F(o) et F(1) du F-Espace.Si le nombre de pages "d" de D est inférieur à 1024, les mots inutilisés de D(o) sont marqués par la valeur -1(c'est-à-dire que tous les bits de ce mot sont à 1).

4.3.3.3. Le Descripteur en mémoire Virtuelle-Gestion de UB.

La totalité du descripteur est installée dans le M-Espace, dès que le F-Espace est actif. Suivant sa taille "d", le Descripteur occupe de un à quatre segments en mémoire. Dans tout ce qui suit, nous supposerons que "d"≤ 256, c'est-à-dire que tout le Descripteur est dans un tel segment. Cette hypothèse a pour seul but de simplifier la présentation, mais n'a aucune influence sur le principe de gestion du Descripteur.

Comme nous l'avons énoncé précédemment, l'ensemble D est une suite de pages de F non nécessairement contiguës . Installer D en mémoire, dans un segment du M-Espace, a pour effet de concaténer toutes les pages pour créer une zone continue et, par là même, de permettre un adressage relatif par rapport au début du segment utilisé. Comme nous le verrons par la suite, le descripteur contient des références à I et P (références externes) ou à D lui-même (références internes). En conséquence, les références externes ont la forme de numéros

de pages dans F, et les références internes ont la forme d'adresses mémoire relatives au début du segment; ceci veut dire que toutes les liaisons entre les éléments de D sont résolues par l'adressage de la machine. Pour le besoins de sa gestion, le Descripteur est découpé en blocs de tailles bien définies. Si "t" est la taille en octets d'un bloc quelconque du Descripteur, la taille de ce bloc doit être une puissance de deux, telle que :

$$2^4 \leq t \leq 2^{11}$$

et l'adresse du début du bloc doit être un multiple de "t". Le découpage permet de mettre en oeuvre une gestion de mémoire libre du type "BUDDY SYSTEM" (Ref.K.1) qui présente les avantages suivants :

- grande rapidité
- Excellent coefficient de remplissage.
- Bonne adaptation à une gestion par page.($4096=2^{12}$).
- Restructuration automatique des blocs libres en pages.

Les blocs de taille identique sont chaînés les uns aux autres; et le début de liste des blocs de chaque taille se trouve dans une table de pointeurs.*(fig.4.3.3.(c))*. Cette table est appelée RUB (Root of UB).

Les deux premiers double-mots (2 x 16 octets) de chaque bloc libre ont une signification particulière *(fig.4.3.3.(c))*. Grâce aux propriétés du BUDDY SYSTEM, la configuration binaire spéciale permet de restructurer facilement deux blocs libres consécutifs de même taille, en un seul bloc de taille double, sans rechercher dans une liste. Il est impossible qu'une telle configuration bien choisie, puisse se trouver plusieurs fois à un endroit critique du Descripteur du fait du contenu des blocs utilisés dans celui-ci. De plus les pointeurs BLUP et BLNEXT satisfont à la restriction sur les adresses des blocs qui doivent être un multiple de la taille. L'ensemble de ces blocs constitue la partie UB de D.

4.3.3.4. Gestion de UB.

La gestion de fichier de CMS+ possède deux fonctions pour accéder à UB.

1/ Demande d'un bloc libre : Elle est réalisée par la fonction :

DEMANDE(t) où t est la taille du bloc désiré.

La fonction sélecte l'entrée dans la table RUB, en fonction de t. Si le pointeur est ≠ 0, la liste des blocs de cette taille est mise à jour, et l'adresse du 1er bloc est donnée en retour. Si le pointeur est nul (pas de blocs libres de cette taille), la fonction réalise alors une DEMANDE(2t), qui est donc récursive. La récursion est arrêtée dès qu'un bloc libre est présent, sinon elle se termine à la demande d'un bloc de taille 4096 (page), qui nécessite une gestion spéciale : c'est le cas de l'accroissement de D(cf paragraphe 4.3.4.2.).

Au retour de chaque demande récursive (taille double de celle nécessaire), la fonction donne l'adresse du bloc ainsi trouvé, et chaine dans la liste des blocs libres de taille appropriée, la deuxième moitié du bloc qui est inutilisée.

2/ Libération d'un bloc. Elle est réalisée par la fonction :

LIBERE(t,a) où "t" est la taille du bloc libéré, et "a" l'adresse de ce bloc.

La fonction vérifie la validité des paramètres (a multiple de t) et examine le bloc associé à celui qui est rendu:le bloc associé est un bloc de même taille tel que les deux blocs forment un nouveau bloc de taille double (Buddies au sens de BUDDY SYSTEM). Si ce bloc possède la configuration binaire particulière, et si les pointeurs BLUP et BLNEXT sont valides (multiples de "t"), le bloc associé est déchainé de la liste libre de cette taille; et le bloc de taille double ainsi formé, est rendu par LIBERE(2t,a') où "a" est soit l'adresse du bloc associé ("a'" multiple de 2t). Là encore, la fonction est récursive,

et si l'on aboutit à LIBERE (4096,a), la récursion s'arrête et l'on doit appliquer une fonction différente qui correspond au cas de diminution de D. (Cf.§ 4.3.2.).

4.3.4. Gestion de P dans le Descripteur.

4.3.4.1. Définitions

A chaque fois qu'à l'intérieur de D il est nécessaire de décrire une partie de F extérieure à D, la gestion de fichiers de CMS+ utilise un bloc particulier: le DPB (DP Block) *fig.4.3.4.(a)*. La structure de ce bloc a été choisie afin de réduire la place occupée dans D pour décrire une partie de F. En effet, un seul bloc de ce type peut décrire n'importe quel groupe de pages contiguës dans F, et à la limite F tout entier.

4.3.4.2. Fonctions de gestion de P .

Nous avons vu que le descripteur contient la partie DP qui décrit P.

DP est composée d'une liste de DPB où chacun d'eux, définit un élément de P. Le premier DPB de la liste se trouve à une adresse fixe dans le Descripteur, c'est le RDP (Root of DP). Pour faciliter la gestion de P, la liste des DPB est classée de telle façon que les pages libres de F soient rangées par ordre de numéro croissant *(fig.4.3.4.(b))*.

Cet exemple montre que le 1er DPB décrit les pages P(o) à P(4) qui se trouvent dans F de F(4) à F(8); le deuxième bloc décrit les pages P(5) à P(7) qui se trouvent de F(20) à F(22) (8 < 20) etc...

Lorsque la gestion de fichiers de CMS+ a besoin d'une page, soit pour étendre D, soit pour étendre I, elle utilise la fonction DEMPAGE. Cette fonction explore la liste DPB pour trouver un bloc tel que DPNBP soit minimum.

Lorsque le DPB est trouvé, DPNBP est diminué de 1, DPNOP qui figure le premier numéro de page libre dans F, est donné en retour, et DPNOP est augmenté de 1. Si DPNBP est égal à zéro, le DPB correspondant est enlevé de la chaîne, et est rendu à la mémoire libre (UB) par la fonction LIBERE.

Inversement, lorsque la gestion de fichiers de CMS+ désire rendre une page à P, elle utilise la fonction LIBPAGE(p), où p est le numéro dans F de la page rendue.

Trois cas peuvent alors se présenter suivant la position de la page p, par rapport aux parties libres de F :

1) Il n'existe pas DPB (i).(DPB de rang i dans la liste) tel que :

- soit DPNOP(i) + DPNBP(i) = p
- soit DPNOP(i) - 1 = p

Ceci signifie que la page libérée ne compète aucune partie libre; et dans ce cas, un nouveau DPB est inséré dans la liste.

2) Il existe DPB(i) tel que :

- soit DPNOP(i) + DPNBP(i) = p
- soit DPNOP(i) - 1 = p

Ces deux éventualités représentent le cas où la page libérée complète une seule partie libre; dans ce cas, la nombre de DPB ne change pas et DPB(i) est mis à jour jour pour refléter l'adjonction d'une nouvelle page.

3) Il existe DPB(i) tel que
   DPNOP(i) + DPNBP(i) = p
et DPNOP(i) - 1 = p

Cette relation implique que la page libérée fait se rejoindre deux parties libres. Dans ce cas, DPB(i+1) est détruit, et DPB(i) est modifié pour décrire la

nouvelle suite de pages contiguës :

DPNBP(i) := DPNBP(i) + 1 + DPNBP(i+1)

4.3.5. Structure et gestion du catalogue C dans le descripteur.

4.3.5.1. Structure du catalogue.

Nous avons vu que le catalogue possède une structure en arbre (cf.§ 4.3.1.1.) Pour décrire cette arborescence, différents blocs de contrôle sont utilisés; ce sont les CB(Catalogue Block), qui sont de 4 types :

1- Les MCB(Master Catalogue Block) *fig.4.3.5.(a)*
2- Les VCB(Valet Catalogue Block) *fig.4.3.5.(b)*
3- Les KCB(Key Catalogue Block) *fig.4.3.5.(c)*
4- Les LCB(Leave Catalogue Block) *fig.4.3.5.(d)*

Tous ces blocs possèdent une taille de 64 octets (8 doubles mots) et satisfont ainsi aux contraintes du § 4.3.3.3., sur le découpage du descripteur en blocs de tailles bien définies.

- Les blocs MCB permettent de décrire tout ce qui est relatif à un niveau donné du catalogue. Le niveau initial ou "racine" est détaillé par un MCB situé à une adresse fixe dans le descripteur D; c'est le R.C.B.(Root Catalogue Block). A partir de ce bloc privilégié, on peut retrouver n'importe quel élément du catalogue, en parcourant la structure d'arbre. La figure 4.3.5.(e) montre les liens entre les tables décrivant un niveau du catalogue. Nous voyons que :

- le MCB est prolongé par les VCB, s'il est insuffisant pour contenir tous les noms des niveaux inférieurs.

- la liste des KCB décrit la liste d'accès telle qu'elle est définie au § 4.3.1.3. elle identifie les utilisateurs qui peuvent avoir accès à tout ce qui se trouve aux niveaux inférieurs avec les privilèges associés;

par exemple : lecture seulement, lecture écriture sans modification du catalogue, sans contrôle, etc... Un utilisateur n'aura accès à une feuille (extrêmité terminale de l'arbre), que si son nom figure au moins une fois dans les KCB rencontrés en parcourant la structure.

- Un bloc LCB décrit l'emplacement du fichier correspondant dans DP. Conformément à la définition donnée au §.4.3.4.1., sur l'utilisation des DPB pour définir la structure de F à partir du descripteur; cette organisation est donnée par une liste de blocs DPB (*fig.4.3.5.(f)*). On peut voir sur la *figure 4.3.5.(g)*, comment il est possible de décrire un catalogue à l'aide des différents blocs définis ici.

4.3.5.2. Gestion du catalogue

a) Initialisation d'un F-Espace

Nous pouvons récapituler la structure du F-Espace par la *figure 4.3.5.(h)*. Cette figure montre les éléments indispensables à l'initialisation de F soit :

- la page zéro qui définit le descripteur D.
- le RCB: c'est la racine du catalogue dans D.
- le RUB: c'est la racine qui décrit la mémoire libre de D.
- le RDP: c'est la racine de DP qui décrit l'espace libre de F.

Par définitions, les blocs RUB, RDP et RCB sont résidents dans la page 1 du descripteur aux adresses hexadécimales suivantes : 1000, 1020, 1040 respectivement. Ces adresses sont imposées par le choix que nous avons fait sur le découpage du descripteur en blocs de taille égale à une puissance de 2, et dont les adresses suivent une règle précise (cf.4.3.3.3.). Ces blocs sont donc tous situés dans les 128 premiers octets de la page 1 de D, qui, rappelons-le, est aussi la page 1 de F.

Pour initialiser un F-Espace vierge, il faut donc construire une page zéro dont les deux premiers mots (4 octets) sont respectivement 0 et 1, pour respecter la définition selon laquelle les pages 0 et 1 de D sont les

pages 0 et 1 de F. Les autres mots de la page zéro sont initialisés à -1 puisque le descripteur est réduit aux seules pages 0 et 1.

-La page 1 du Descripteur est initialisée selon la *figure 4.3.5.(i)* et nous pouvons remarquer que :

- Le RUB est conforme aux exigences de la gestion de la mémoire libre de D et qu'en particulier, il n'existe pas de blocs de taille 64 et 32 octets. Ils sont déjà occupés respectivement par le RCB et le RUB lui-même.

- Il est nécessaire de n'avoir qu'un seul DPB (le RDP) pour décrire la totalité de l'espace libre de F, puisqu'à l'état initial tout l'espace qui fait suite à la page 1 est inoccupé (DPNOP=2, DPNB=p-2).

- Le RCB est initialisé à 0; seul le nom du propriétaire du F-Espace figure dans ce bloc.

Cette initialisation est particulièrement simple et mise à part la présence de la taille du F-Espace dans le RDP (DPNBP=p-2), elle est identique pour tout F-Espace. Par contre il n'est effectué aucune analyse de surface du support physique du F-Espace, qui conduit à une opération beaucoup trop couteuse.

b) Gestion du Catalogue

Lorsque l'utilisateur crée un nouveau fichier, l'environnement CMS+ construit l'ensemble des MCB, non encore définis nécessaire à la description du nom complet de ce fichier, ainsi que le LCB final. Au cours de la création ou par suite de modifications, la taille du fichier peut devenir importante; auquel cas la liste de DPB attachée au LCB va croitre suivant les besoins. Inversement si un fichier est effacé du catalogue, l'espace occupé par le fichier dans I est rendu à P et la liste des DPB détruite, de même que le LCB. A partir de ce dernier on repère le MCB de niveau supérieur par le pointeur MCBUP. On remplace alors la CENTRY associée au LCB détruit par la dernière CENTRY du niveau, ce qui a pour effet de condenser l'information. Si la CENTRY à détruire est unique pour le niveau, le MCB lui-même est détruit et le processus recommence au niveau supérieur par le pointeur MCBUP.

c) Sauvegarde du catalogue

- La sauvegarde du catalogue dans le F-Espace, est une opération délicate mais obligatoire étant donné le contenu de celui-ci. En effet toute perturbation dans son fonctionnement peut entrainer la perte d'une partie ou de la totalité du contenu du F-Espace.

La sauvegarde du catalogue est à la charge de CP+ : Lorsqu'une page d'un segment appartenant à un F-Espace est modifiée, et qu'elle doit être réécrite sur support externe, par exemple pour laisser sa place en mémoire réelle à une autre page, CP+ initialise deux opérations d'écriture:

- Une écriture sur tambour comme cela se fait dans le cas général.
- Une écriture sur disque à l'emplacement qu'occupe cette page dans le F-Espace considéré.

L'écriture dans le F-Espace se justifie par le fait qu'à la suite d'un accident du système, le seul emplacement connu de la page est celui dans le F-Espace.

L'écriture sur tambour permet d'accélérer l'accès à la page pour les lectures ultérieures, si la page est référencée mais non modifiée (cas le plus fréquent).

A la fin de chaque commande et si le catalogue a été modifié, l'environnement CMS+ demande à CP+ de forcer une réécriture de toutes les pages du Descripteur qui ont été modifiées et qui sont actuellement en mémoire réelle. Nous imposons cette précaution pour forcer la sauvegarde du descripteur, dans le cas où certaines de ses pages sont restées constamment en mémoire réelle sans que dans le cadre de la Pagination, CP+ ait eu besoin de les réécrire (cas d'une charge peu importante).

Cette façon de procéder, associée à la sauvegarde automatique expliquée au § 4.3.1.5. doit assurer une protection quasi totale contre les accidents du système.

CHAPITRE 5

CONCLUSION

Nous nous étions fixés comme règle pour la rédaction de cet ouvrage, de simplifier certaines parties très techniques, et pour cela de ne donner que des présentations fonctionnelles, à chaque fois que c'était possible.

Le lecteur voudra bien ne pas nous tenir rigueur, si nous avons parfois manqué involontairement à cette règle soit en restant trop vague, soit en détaillant trop certaines parties.

Sur le fond, nous avons essayé de présenter une démarche relativement classique qui consiste à tester des nouveautés techniques, en apportant des modifications quelquefois discrètes à un produit déjà existant et de fonctionnement sûr, en l'occurrence CP/CMS. Ceci nous a permis d'imputer tous les défauts de fonctionnement à nos modifications. D'autre part, la possibilité d'activer sous CP-67 lui-même des Machines Virtuelles du type 360-67, nous a permis de mettre au point notre prototype sans perturber le fonctionnement de l'exploitation. En fait la mise au point par ce mécanisme ne peut être que partielle, car il n'est pas possible d'obtenir en simulation virtuelle les conditions critiques qui se produisent au cours du fonctionnement réel. Ces conditions peuvent soit provoquer une dégradation des performances du système, soit plus brutalement aboutir à un "Arrêt Complet".

Les options des nouveaux systèmes CP+/CMS+ que nous avons pu expérimenter, ont toutes apporté des améliorations de fonctionnement. Nous l'avons montré tout au long du Chapitre 3. De la dernière expérience, presque terminée à l'heure actuelle, nous escomptons aussi une amélioration de rendement général; cependant, la marge d'incertitude ne nous permet pas d'être catégorique sur ce point avant la mise en service définitive.

Dans le Chapitre 4 nous avons volontairement omis de parler du problème de partage des fichiers dans le cadre de F-Espaces. Ce problèmes est délicat, et tout comme pour le partage de M-Espaces, nous sommes confrontés à un certain nombre de solutions et nous n'avons pas encore déterminé comment fixer notre choix.

Dans le cadre de CMS+ et en corollaire des spécifications déjà établies, les travaux à venir porteront essentiellement sur une refonte du langage de commande. Nous voulons définir celui-ci sous forme de commandes primitives élémentaires et élaborer un mécanisme de MACRO-LANGAGE.

L'utilisateur pourra ainsi construire son propre jeu de commandes, adapté au mieux à ses désirs, ou à ses besoins, ceci en composant de véritables programmes dont les ordres seront soit des requêtes propres au Macro-langage et destinées à contrôler l'exécution de la MACRO-COMMANDE (boucles, tests, variables formelles, etc...), soit des commandes primitives élémentaires.

L'architecture du système CMS devant être totalement revue, pour donner CMS+, nous utiliserons le langage spécialisé GSL (Ref.G1), ce qui devrait accélérer l'écriture des instructions et faciliter la mise au point.

Un certain nombre de problèmes restent posés actuellement par la définition d'un Environnement CMS+ spécialisé dans le traitement des travaux de type "BATCH", et qui donnerait à l'ensemble CP+/CMS+, la qualité de système intégré. Cette qualité nous semble fondamentale et, à l'heure actuelle, fait

cruellement défaut à CP/CMS. Nous avons participé aux travaux d'un groupe d'étude qui, en Septembre 1971, avait déjà abordé le problème, pour en déterminer le coût. Mais l'investissement à consentir dans un tel projet semble trop important pour qu'il puisse être retenu, tout au moins dans l'immédiat.

En résumé, nous pensons qu'un rôle important peut être joué, dans les années à venir, par des systèmes du type CP+/CMS+. Ceux-ci offrent d'une part des outils irremplaçables de mise au point et d'enseignement, grâce au concept de Machine Virtuelle, et d'autre part offrent une grande facilité d'emploi et un rendement très acceptable, grâce au concept d'Environnement.

B I B L I O G R A P H I E

A_1 B.W.ARDEN, B.A.GALLER, T.C.O'BRIEN and F.H. WESTERVELT
Program and adressing structure in a time-sharing environment; J.A.C.M. 13,1 (Jan.1966), 1-16.

A_2 A.AUROUX, C.HANS
Systèmes CP-67 et CMS; Publication IMAG (Juillet 1968).

B_1 Y.BARD
Performance Criteria and measurment for a time-sharing system; IBM Syst.J.Vol.10, n°3, 1971.

B_2 L.A.BELADY
A study of replacement algorithms for virtual storage computers; IBM Syst. J.5,2 (1966) 78-101.

B_3 A.BENSOUSSAN, C.T.CLINGEN and R.C. DALEY
The multics virtual memory; Proc.Second ACM Symp.on operating systems principles, Princeton, N.J., Oct.20-22, 1969, pp 30-42.

C_1 L.COMEAU
A study of the effect of user program optimization in a paging system; ACM symp.Gatlinburg, Tenn., Oct.1.4., 1967.

C_2 CMS Program Logic Manual IBM Gy20-0591.

C_3 CP-67 Operator's Guide IBM GH20-0856.

C_4 CP-67 Program Logic Manual IBM Gy20-0590.

C_5 CMS USER'S Guide IBM GH20-08591.

D_1 R.C.DALEY, P.G.NEUMANN

A General Purpose file system for secondary storage; Proc.FJCC, 1965.

D_2 P.J.DENNING

Virtual Memory; Computing Surveys Vol.2, n°3,Sept.1970.

D_3 P.J.DENNING

Effects of scheduling on file memory operations; SJCC, Vol.30, p 9-21.

D_4 P.J.DENNING

Resource allocation in multiprocess computer systems. Tech. Rep.MAC-TR-50, MIT Project MAC, Cambridge, Mars 1968 (Ph.D. Thesis).

D_5 P.J.DENNING

Equipment configuration in balanced computer systems; IEEE Trans.C-18 (Nov.1969), 1008-1012.

D_6 J.B.DENNIS

Segmentation and the design of multiprogrammed Computer Systems; J.ACM 12,4(Oct.1965), 589-602.

D_7 J.B.DENNIS, E.C.VAN HORN

Programming semantics for multiprogrammed computations; Comm.ACM 9,3 (Mars 1966),143-155.

D_8 DIJKSTRA

The structure of "THE"multiprogramming system; C.A.C.M. 5/68.

H_1 D.J.HATFIELD, J.GERALD

Program restructuring for virtual memory; IBM system.J. Vol. 10, n°3, 1971.

K_1 D.E. KNUTH

The Art of Computer programming; Vol.1 Addison-Wesley, Reading, Mass., 1968, p 435-455.

K_2 C.J.KUEHNER, B.RANDELL

Demand paging in perspective; FJCC, Vol.33, p 1011-1018.

G_1 GRENOBLE SYSTEM LANGUAGE

IBM Grenoble Scientific Center; Report n°FF2.0133, Janvier 1972.

L_1 H.LANER

Bulk core in a 360/67 time sharing system; FJCC, Vol.31, p 601-609.

L_2 P.LEFEBVRE

SPY: un système de contrôle pour la mise au point des systèmes de programmation par couplage de machines virtuelles; Thèse de 3e Cycle Grenoble, Juillet 1972.

L_3 J.P.LE HEIGET

Description de la gestion de fichier de CMS; Note interne IMAG. Mars 1971.

L_4 J.S.LIPTAY

The cache; IBM Syst.J.7,1 (1968), 15-21.

M_1 MTS Manual

The University of Michigan Computing Center - Ann Arbor, Michigan.

M_2 B.H. MARGOLIN, R.P. PARMELEE, M.SCHATZOFF

Analysis of free storage algorithm;
IBM Syst.J.Vol.10, n°4, 1971.

M_3 R.A.MEYER, L.H.SEAWRIGHT

A virtual machine time sharing system; IBM Syst.J.Vol.9, n°3, 1970.

P_1 T.PINKERTON

Program behavior and Control in virtual storage computer systems; CONCOMP Project. Rep.N°4, V.of Mich., Avril 1968 (Ph.D.Thesis).

R_1 B.RANDELL, C.J.KUCHNER

Dynamic storage allocation systems; Comm. ACM 11 (Mai 1968), 297-305.

T_1 TSS/360

System Logic Summary PLM-IBM GY 28-2009.

W_1 M.V. WIELKES

A model for cary space allocation in a time sharing system; SJCC Vol.34, 265-271.

TABLE DES MATIERES

* * * *

Pages

Vu,

Grenoble, le

Le Président de la Thèse

Vu, et permis d'imprimer

Grenoble, le

Le Président de l'Université Scientifique et Médicale.