Contribution à l'architecture de mécanismes élémentaires pour certains systèmes générateurs de machines virtuelles

Claude Hans

THESE

présentée à

l'université scientifique et médicale de grenoble

pour obtenir

le grade de docteur ès-sciences

par

Claude Hans

Contribution à l'Architecture de mécanismes élémentaires pour certains systèmes générateurs de machines virtuelles

Thèse soutenue le 24 novembre 1973 devant la commission d'examen

Président: *J. Kuntzmann*

Examinateurs: *N. Gastinel*

L. Bolliet

M. Griffiths

L. Nolin

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
	AUBERT Guy	Physique
	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
	BERNARD Alain	Mathématiques Pures
	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
	BOUSSARD Jean-Claude	Mathématiques Appliquées
	BRAVARD Yves	Géographie
	BRISSONNEAU Pierre	Physique du solide
	BUYLE-BODIN Maurice	Electronique
	CABANAC Jean	Pathologie chirurgicale
	CABANEL Jean	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
	DESRE Pierre	Métallurgie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée
	DOLIQUE Jean-Michel	Physique des plasmas
	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
	GASTINEL Noël	Analyse numérique
	GEINDRE Michel	Electroradiologie
	GERBER Robert	Mathématiques Pures
	GIRAUD Pierre	Géologie
	KLEIN Joseph	Mathématiques Pures
Mme	KOFLER Lucie	Botanique et Physilogie végétale
MM.	KOSZUL Jean-Louis	Mathématiques Pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LAJZEROWICZ Joseph	Physique
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre-Jean	Mathématiques appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques Pures
MM.	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

MM.	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLAND François	Zoologie
	VALENTIN Jacques	Physique nucléaire
	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
	HANO JUN-ICHI	Mathématiques Pures
	STEPHENS Michaël	Mathématiques appliquées

PROFESSEURS SANS CHAIRE

MM.	BEAUDOING André	Pédiatrie
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BERTRANDIAS Jean-Paul	Mathématiques appliquées
	BIAREZ 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
	GAUTHIER Yves	Sciences biologiques
	GAVEND Michel	Pharmacologie
	GERMAIN Jean-Pierre	Mécanique
	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
	DE ROUGEMONT Jacques	Neurochirurgie
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIBILLE Robert	Construction mécanique
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale

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
	BILLET Jean	Géographie
	BLIMAN Samuel	Electronique (EIE)
	BLOCH Daniel	Electrotechnique
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BOUCHET Yves	Anatomie
	BOUVARD Maurice	Mécanique des fluides
	BRODEAU François	Mathématiques (IUT B)
	BRUGEL Lucien	Energétique
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse
	CHIAVERINA Jean	Biologie appliquée (EFP)
	CHIBON Pierre	Biologie animale
	COHEN-ADDAD Jean-Pierre	Spectrométrie physique
	COLOMB Maurice	Biochimie médicale
	CONTE René	Physique
	COULOMB Max	Radiologie
	CROUZET Guy	Radiologie
	DURAND Francis	Métallurgie
	DUSSAUD René	Mathématiques (CUS)
Mme	ETERRADOSSI Jacqueline	Physiologie
MM.	FAURE Jacques	Médecine légale
	GENSAC Pierre	Botanique
	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
	LACOUME Jean-Louis	Physique
Mme	LAJZEROWICZ Jeannine	Physique
MM.	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)

MM.	MAYNARD Roger	Physique du solide
	MICHOULIER Jean	Physique (IUT A)
	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
Mlle	RIERY Yvette	Biologie animale
MM.	RACHAIL Michel	Médecine interne
	RACINET Claude	Gynécologie et obstétrique
	RENAUD Maurice	Chimie
	RICHARD Lucien	Botanique
Mme	RINAUDO Marquerite	Chimie macromoléculaire
MM.	ROMIER Guy	Mathématiques (IUT B)
	SHOM Jean-Claude	Chimie générale
	STIEGLITZ Paul	Anesthésiologie
	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
	SIDNEY STUARD	Mathématiques Pures
	YACOUD Mahmoud	Médecine légale

CHARGES DE FONCTIONS DE MAITRES DE CONFERENCES

Mme	BERIEL Hélène	Physilogie
Mme	RENAUDET Jacqueline	Microbiologie

Fait le 30 mai 1972.

Je souhaite remercier:

-Messieurs les professeurs Jean KUNTZMANN et Noël GASTINEL qui ont bien voulu faire partie du jury de cette thèse;

-Monsieur le professeur Louis BOLLIET qui m'a accordé sa confiance en m'encourageant à constituer en 1968, puis à animer ensuite, un petit groupe d'étudiants inscrits en thèse d'informatique;

-Monsieur le professeur Louis NOLIN qui, en m'accueillant en 1962 à l'Institut Blaise Pascal à Paris, m'a définitivement 'converti' à l'informatique;

-Monsieur Michäel GRIFFITHS, maître de conférences, qui a accepté d'être de ceux qui jugeraient mon travail.

Toute activité de recherche dans le domaine des systèmes de contrôle des calculateurs qui prétend donner lieu à la réalisation de prototypes opérationnels, ne peut être qu'un travail d'équipe.

A ce titre, j'adresse tous mes remerciements aux étudiants et amis qui ont participé ou qui ont encore aujourd'hui une part active au travail de notre groupe:

-ceux qui ont déjà présenté leurs thèses, dans l'ordre de soutenance:

Michel ADIBA, Bernard PETEUL, Patrick LEFEBVRE, Juan RODRIGUEZ, Jean-Pierre LE HEIGET, NGUYEN THAN THI, Xavier de LAMBERTERIE.

-ceux qui présenteront ultérieurement le résultat de leurs recherches:

Max CREVEUIL, Jean-Pierre DUPUY, Danielle HANS et Pierre SAUVAGE.

Ce groupe doit beaucoup aux relations constantes qu'il entretient avec ceux qui sont déjà engagés dans une vie professionnelle orientée vers l'informatique.

Qu'il me soit ainsi permis de remercier l'équipe de Maurice BELLOT de l'Institut de Mathématiques Appliquées de Grenoble ainsi que tous mes collègues du Centre Scientifique.

Je remercie la compagnie IBM, plus particulièrement Max PELTIER, grâce à qui j'ai pu réaliser ce travail et qui permet au groupe de s'exprimer de façon constante, dans de très bonnes conditions.

Je ne saurais enfin oublier le Service de Reproduction pour la réalisation matérielle de ce mémoire.

CONTRIBUTION

A

L'ARCHITECTURE DE MECANISMES ELEMENTAIRES

POUR

CERTAINS SYSTEMES GENERATEURS

DE

MACHINES VIRTUELLES

A mon épouse,

-Présentation-

Ce mémoire décrit des mécanismes élémentaires qui ont été soit imaginés, soit adaptés pour développer et étoffer l'architecture de base d'une certaine classe de systèmes.

Il s'agit de générateurs de machines virtuelles. Ces programmes appelés 'hyperviseurs' créent à partir d'une architecture de calculateurs donnée, des exemplaires multiples, identiques ou différents, de l'architecture de base.

Ils transforment le calculateur réel en une suite d'entités appelées 'machines virtuelles' qui sont la réplique d'ordinateurs existants. Elles sont indépendantes les unes des autres et ignorent l'existence du générateur. De plus, il est possible d'activer sur chacune d'elles le système de son choix, en particulier l'hyperviseur qui fonctionne alors sous contrôle de lui-même (situation récursive, généralisable au degré n).

Grâce à ces propriétés, une solution efficace peut être donnée aux divers problèmes de protection et de sécurité ainsi qu'à ceux qui concernent les transferts de programmes entre des entités d'architectures différentes.

D'autre part, les objectifs de l'hyperviseur ont été dès l'origine parfaitement définis parce que les environnements destinés aux utilisateurs étaient à priori connus de façon précise et non ambigüe.
En conséquence, l'hyperviseur est doté de structures claires et simples qui contribuent à lui garder une dimension 'humaine'.

Les mécanismes présentés dans le premier chapitre permettent d'établir une relation d'inclusion entre machines virtuelles dans le but de faciliter le développement et la mise au point de systèmes.

Le second chapitre expose les moyens mis en oeuvre pour réaliser des transferts d'information unidirectionnels depuis l'hyperviseur vers les machines virtuelles permettant ainsi d'étudier le comportement du générateur depuis l'environnement virtuel.

Enfin, dans le troisième chapitre, nous présentons des entités virtuelles, n'ayant aucun équivalent réel, qui peuvent être générées sous le contrôle de l'hyperviseur. Dans cet esprit, nous cherchons à simplifier la structure des systèmes dits 'virtuels' qui contrôlent ces entités sans pour autant négliger les performances de fonctionnement.

Le principe même de la génération des machines virtuelles impose à la machine réelle utilisée des contraintes architecturales sévères; aussi, le nombre d'hyperviseurs est-il assez restreint car peu d'ordinateurs satisfont les critères demandés.
C'est l'une des raisons pour lesquelles les différents prototypes construits, dans le but de valider les mécanismes proposés, ont tous été réalisés sur machine IBM 360/67, sous le contrôle de CP67 et que peu de références sont faîtes à d'autres systèmes.

L'ordinateur IBM 360/67 utilisé, est celui de l'Institut de Mathématiques Appliquées (IMAG) de l'Université de Grenoble.

Le sujet du travail présenté n'est pas une comparaison entre divers systèmes mais l'exposé d'idées qui ont conduit à plusieurs réalisations décrites en particulier, de façon plus détaillée, dans les références 17, 35, 36, 37, 40 et 42.

-Table des matières-

-Introduction-

Nous avons voulu présenter ici un des aspects de notre activité de recherche concernant les systèmes générateurs de machines virtuelles (ref 7) et notamment parmi ceux-ci, CP67.

Nous nous intéresserons tout particulièrement aux travaux qui ont été réalisés de 1966 à 1972.
Pourquoi ces dates limites?

1966 représente le début de notre participation à l'étude, au développement et à l'utilisation de cette classe de systèmes dans le cadre du Centre Scientifique IBM de Cambridge aux Etats-Unis où nous avons travaillé aux projets CP40 et CP67 sous la direction de Norman RASMUSSEN.
C'est là que le concept de machine virtuelle a été imaginé et effectivement mis en oeuvre avec le premier système générateur CP40 (ref 1) dans le but de permettre à une petite communauté d'utilisateurs de travailler simultanément sur une seule et même machine

à l'analyse du comportement de systèmes, à des développements de nouveaux systèmes et de programmes d'application.
Cette simultanéité était jugée préférable à la solution qui consistait à affecter l'ordinateur par vacations successives aux différents groupes.
Toutefois, l'étude et la construction de systèmes impliquaient de garder des environnements identiques à ceux des machines réelles. C'est pour résoudre ce problème que l'idée de machine virtuelle, réplique d'un élément réel, est née.

Or, le hardware utilisé est en fait une contrainte majeure qui dans la plupart des cas ne permet pas de générer de telles machines. Robert GOLDBERG a fait sur ce sujet, une analyse intéressante des rares ordinateurs sur lesquels cela est possible (ref 19).

Le Centre Scientifique de Cambridge a pu réaliser en 1966-1967 un prototype de système générateur, CP40, sur un 360/40 modifié.
En fait, CP40 a surtout un intérêt historique: il n'a fonctionné qu'un temps limité sur une seule machine qui comportait une modification technologique majeure; il s'agissait d'un mécanisme de traduction dynamique

d'adresses qui n'a jamais été produit de façon industrielle.

Mais très rapidement, le succès de cette expérience préliminaire a entraîné le développement d'une seconde version pour le 360/67 appelée CP67. Elle avait l'avantage de fonctionner sur une machine produite en série et par là-même, elle était susceptible d'intéresser plus d'une seule communauté d'utilisateurs.

Comme toute idée neuve, le concept de systèmes générateurs de machines virtuelles, a eu des difficultés à être admis et la majorité de ses détracteurs a été longtemps critique sans que leur position soit totalement justifiée.

Cette période est maintenant révolue et face aux nombreux services rendus (ref 39), on a vu apparaître au moins chez un constructeur, IBM, un système appelé VM qui applique ce concept sur la plus grande partie des ordinateurs de la gamme 370; il est ainsi proposé à tous les utilisateurs et non plus seulement réservé à quelques centres de recherche privilégiés susceptibles d'acquérir une machine aussi coûteuse qu'un 360/67.

1972, date limite supérieure de notre étude, correspond précisément à la disponibilité de ce générateur de machines virtuelles sur cette nouvelle classe de machines dont le rapport coût/performances modifie bien des données. Nous avons atteint l'étape du développement industriel et bon nombre de mécanismes qui ont été imaginés et créés au cours de ces dernières années, sont maintenant inclus de façon standard dans la nouvelle version.

Le générateur CP67, ou encore l'hyperviseur CP67 (ainsi appelé parce qu'il est 'superviseur de superviseur'), exploite intensivement l'idée de mémoire virtuelle.
Il n'est pas question d'expliciter ici les différentes variantes appliquées à cette notion; elles sont abondamment commentées dans de nombreux ouvrages (voir en particulier, ref 14 et la bibliographie correspondante ainsi que ref 24). On y trouve toutes les propositions depuis celle qui consiste en un réarrangement des adresses réelles d'un calculateur conduisant à la génération d'une mémoire virtuelle unique de même taille, jusqu'à la plus ambitieuse qui génére des mémoires virtuelles multiples recouvrant

l'ensemble de la hiérarchie de mémoires (y compris tambours, disques, bandes, etc...).
Cette dernière solution fait disparaître la notion d'unité physique pour présenter l'ensemble de l'espace disponible sous la forme d'un espace virtuel généralement structuré de façon multi-dimensionnelle; elle veut ainsi masquer totalement les mécanismes de transferts entre les différents niveaux de la hiérarchie de mémoires au point même que l'utilisateur en ignore l'existence.

Il y a quelques années, deux grands systèmes MULTICS (ref 41) et TSS (ref 33) ont commencé à utiliser systématiquement de telles mémoires virtuelles; ils se sont toutefois heurtés à une difficulté majeure dans le domaine des performances: la technologie de 1966 était alors incapable de fournir de vastes mémoires (ayant un temps d'accès d'au plus quelques microsecondes) de façon industrielle, pour un coût abordable. Cette déficience a perturbé le fonctionnement de tels systèmes qui, pour être utilisables, impliquaient l'emploi de configurations très importantes.

Il faut noter que la plupart des programmes qui ne satisfont pas à des critères de performances

raisonnables sont condamnés à une mort plus ou moins rapide. Ils présentent malgrè tout l'intérêt d'être le véhicule d'idées qui peuvent être reprises ultérieurement lorsque la technologie adéquate devient disponible.

CP67 pouvait, et peut accepter, des mémoires virtuelles multiples d'au plus 16 millions de caractères chacune mais, dans la pratique, notre expérience nous a montré que les diverses applications se satisfaisaient de mémoires de 256 K (1K=1024 caractères mémoire), quelquefois 512 K et rarement plus de 1024 K.
La somme de toutes les mémoires qui peuvent être activées à un instant donné, reste ainsi dans un rapport raisonnable vis-à-vis de celle de la mémoire réelle. Les performances obtenues peuvent alors être satisfaisantes.

C'est dans ce cadre et avec ces contraintes que nous avons travaillé en cherchant tout d'abord à isoler, puis exploiter certaines propriétés des mémoires et des machines virtuelles.

Nous décrirons ici quelques mécanismes que nous avons soit proposés, soit adaptés au cours de ces dernières années.

Il s'agit d'un travail d'équipe; aux trois chapitres qui vont suivre, sont directement associées sept thèses dont nous avons eu la responsabilité technique et pour lesquelles nous avons soumis les sujets aux chercheurs qui ont accepté de travailler avec nous pour réaliser des prototypes mettant en oeuvre les mécanismes en question. Outre ces thèses, plusieurs articles et notes techniques se rapportant à ce travail ont été publiés.

Pour ces raisons, nous n'exposons que les principales idées directrices qui ont constamment orienté ce travail sans en montrer la réalisation de façon trop détaillée. Nous espérons que la présentation globale de nos recherches et de nos réalisations donnera au lecteur qui n'a pas participé aux travaux, le désir d'en savoir un peu plus et l'entraînera ainsi à lire les références qui y sont citées.

Le premier chapitre introduit des mécanismes d'inclusion dont l'objet est de faciliter la création, la mise au point et la modification de systèmes.

Le second chapitre propose des outils qui permettent d'analyser le comportement de l'hyperviseur (CP67 en l'occurrence) à partir de machines virtuelles.

Le troisième chapitre enfin, présente l'ébauche partiellement réalisée d'une extension de l'hyperviseur autorisant non seulement la génération de machines virtuelles mais aussi d' entités sans réplique réelle.

Quant à la bibliographie, elle est volontairement restreinte mais elle indique les ouvrages importants qu'il est bon de connaître. En se reportant à celles des textes cités, le lecteur assidu trouvera ainsi par adressage indirect, les coordonnées de nombreux articles qui traitent du sujet de cette présentation.

Mise au point de systèmes
par
inclusion de machines virtuelles

1.1 Présentation

Dans ce premier chapitre, nous introduisons des mécanismes d'inclusion qui vont permettre à une machine virtuelle spécialisée d'avoir accès à toutes les ressources disponibles d'une autre machine virtuelle. Nous les avons imaginés et réalisés pour tenter d'aplanir les difficultés que posent la construction et la modification de systèmes (ref 26).

Nous aborderons successivement:

- -le problème de la mise au point,
- -les mécanismes de base qui ont été réalisés,
- -la façon dont on les emploie pour les mettre au service de l'utilisateur,
- -et enfin, diverses extensions jugées fondamentales.

1.2 Le problème de la mise au point

La mise au point d'un système est une opération qui intervient soit lors de sa construction, soit à la suite d'un fonctionnement défectueux que l'on cherche à corriger, soit enfin au cours d'une tentative de modification destinée à étendre ses possibilités.

Pour la réaliser, une première solution consiste à utiliser certains outils câblés intégrés au calculateur. Ce sont en général, des dispositifs disponibles au pupitre de l'ordinateur qui permettent:

-d'arrêter et de relancer le déroulement d'un programme,

-de choisir la cadence d'exécution en demandant soit l'arrêt après chaque instruction, soit l'enchaînement avec la (ou les) instruction(s) suivante(s),

-d'examiner et de modifier le contenu de la mémoire et des registres,

-de définir une adresse telle que, si une instruction y fait référence ultérieurement, l'unité de traitement doive s'arrêter.

L'emploi de ce type d'outils est cependant particulièrement coûteux car toutes les ressources

physiques de l'ordinateur sont alors affectées à la seule personne chargée de réaliser la mise au point. Si cette méthode de travail est concevable au niveau des petits calculateurs, elle devient irréaliste lorsqu'il s'agit d'ensembles de traitement plus importants; leur fonctionnement est en effet contrôlé par des systèmes évolués dans le but de rechercher une efficacité optimale par la mise en oeuvre du principe de multiprogrammation.

En dehors de cette contrainte de coût, les possibilités offertes par ces outils sont, en outre, limitées:

-D'une part, on ne peut en effet accéder qu'au contenu de la mémoire physique principale et non à celui de la mémoire logique qui, à un instant donné, est constituée de multiples éléments disséminés sur les différents composants de la hiérarchie de mémoires (disques, tambours etc...). De plus, cet examen et l'éventuelle modification qui en découle, doivent être réalisés sous la seule forme reconnue par l'unité de traitement, c'est-à-dire des chaînes de bits: ceci ne facilite guère la mise au point!

-D'autre part, une demande d'arrêt sur référence à une adresse prédéfinie ne peut être formulée par

une expression conditionnelle du type: 'si telle éventualité ou (et) telle autre se produit, alors arrêter, sinon continuer'.

-La dernière critique enfin, dans cette liste non exhaustive, vient du fait qu'il est en général impossible de modifier ou d'accroître les possibilités de l'outil de base qui est câblé.

La solution uniquement hardware est donc insuffisante; il faut alors envisager la création d'une série de programmes auxquels on donne le contrôle dès qu'un événement insolite et précis se produit. On peut ainsi établir un dialogue entre le système et le programmeur chargé de sa mise au point.

Nous nous sommes intéressés à ces phénomènes qui ont constitué notre premier domaine de recherche.

Pour cela, nous avions introduit la terminologie suivante:

-'espion' pour désigner la suite de programmes dont nous venons de parler,

-'objet' pour qualifier le système testé.

Nous utiliserons ici ces mêmes termes qui ont au moins le mérite d'être imagés.

Pour que l'espion puisse jouer son rôle et fournir une aide efficace à la mise au point, il doit satisfaire deux critères:

-son fonctionnement doit être sûr et indépendant de l'objet;

-il doit permettre la mise au point de divers systèmes qui n'ont pas été conçus pour cohabiter avec lui.

La notion d'indépendance fait apparaître, entre autres, que les opérations d'entrée-sortie assurant les transmissions des informations, pour le compte de l'espion, en provenance ou à destination soit du terminal, soit des différents niveaux de la hiérarchie de mémoires, doivent être autonomes.

Ceci implique au minimum, l'existence dans l'outil de mise au point, de sous-programmes susceptibles de gérer les unités physiques, de reconnaître et d'aiguiller les interruptions. On peut ensuite l'étoffer en lui ajoutant une gestion de mémoire libre puis une gestion de fichiers; à ce stade, l'espion devient un véritable système indépendant, généralement plus simple que l'objet puisqu'il peut être de nature séquentielle (en effet, il n'est pas nécessaire d'optimiser la gestion des ressources par multiprogrammation).

La deuxième notion fait apparaître une contrainte imposée par le problème de la protection qu'il faut assurer à l'espion quand l'objet est actif.

Trois solutions peuvent être envisagées:

-L'une consiste à installer l'espion en mémoire puis à lui faire interpréter toutes les instructions exécutées par l'objet afin de vérifier, en particulier, la validité des adresses générées. Une telle simulation systématique est fort coûteuse; en effet, elle accroît le temps d'exécution par un facteur multiplicatif de plusieurs dizaines.

-La seconde consiste à faire cohabiter l'espion et l'objet en mémoire; cet aspect met en jeu la protection mémoire et implique ainsi deux restrictions au niveau de l'objet:

-celui-ci perd, en effet, la liberté d'employer le mécanisme ainsi réquisitionné (ce qui limite la classe des systèmes susceptibles d'être testés);

-il doit être modifié afin de rendre le contrôle à l'espion en cas de tentative de violation de la zone de mémoire protégée.

-La troisième solution enfin, propose de faire résider objet et espion dans deux espaces mémoires distincts.

Précisons qu'en fait, l'objet ne doit pas pouvoir adresser l'espace espion lorsqu'il a le contrôle; par contre l'espion, lors de son exécution, doit pouvoir accéder à tout ce qui appartient à l'objet. On définit ainsi une relation d'inclusion: l'espace objet est inclus dans celui de l'espion. Les calculateurs qui permettent la mise en oeuvre du concept de mémoire virtuelle en distinguant les notions d'espaces physique et logique, autorisent en général, la création d'une telle relation (au moins, en ce qui concerne la mémoire).

Nous avons entrepris avec Bernard PETEUL (ref 42) puis Patrick LEFEBVRE (ref 36), la réalisation de cette dernière solution sous le système CP67.

Grâce aux propriétés des machines virtuelles générées par CP67, nous avons pu obtenir, en modifiant l'hyperviseur, des configurations incluses les unes dans les autres.

Cette méthode offre plusieurs avantages:

-l'objet et l'espion fonctionnant sur des machines virtuelles, l'arrêt de l'une d'elles (ou des deux) n'implique pas celui du calculateur réel; les problèmes de coût précédemment cités sont ainsi éliminés.

-disposant pour l'espion d'une machine complète, il est alors possible de développer, sans aucune contrainte, un outillage spécialisé dans les problèmes de mise au point, efficace et volumineux.

1.3 Réalisation du couplage

Il s'agissait donc de réaliser le couplage de deux machines virtuelles de telle sorte que l'objet soit un sous-ensemble de l'espion.

Les problèmes posés par cette opération d'inclusion concernent:

-la mémoire,

-les unités centrales (ou processeurs),

-les unités d'entrée-sortie.

1.3.1 Inclusion de mémoires

La mémoire réelle R, accessible d'une façon aléatoire par l'unité centrale est découpée en (p+r) blocs de longueur fixe appelés 'pages'.

La zone constituée par les p premières pages est réservée à l'hyperviseur; il y réside. Nous la noterons R'=(R(0), R(1),...R(p-1)).

La seconde zone de R, soit:

R"=(R(p), R(p+1),...R(p+r-1)) forme l'ensemble des pages qui seront utilisées dynamiquement comme lieu de résidence temporaire de tout ou partie des mémoires virtuelles sur lequel l'hyperviseur multiprogramme à un instant donné.

Soit L=(L(0), L(1),...L(q-1)) une suite de q pages réparties sur différents niveaux de la hiérarchie mémoire (tambours et disques) telle que l'intersection de R" et L soit vide. L est un réservoir de pages disponibles où seront rangées les mémoires virtuelles qui ne peuvent résider dans la zone R" par manque de place.

Remarquons toutefois, qu'un élément de mémoire virtuelle doit appartenir à R" pour être utilisable par

l'unité centrale. Toute référence à un élément de L implique donc un transfert de la page correspondante dans R". Cette opération qui impose de trouver un emplacement disponible dans R", peut elle-même être précédée d'une recopie d'un élément de R" dans L afin de procurer la place recherchée.

Considérons alors N mémoires virtuelles MV(b,1), MV(b,2)...MV(b,N) dont la ième comporte v(i) pages nommées (MV(0,i), MV(1,i),...MV(v(i)-1,i)).
En général, pour $i \neq j$, le nombre de pages de MV(b,i) soit v(i),est différent de celui de MV(b,j) soit v(j).
Pour tout i, on peut définir une application f(i) qui applique MV(b,i) dans la réunion de R" et de L appelée RL.
Soit k une page de MV(b,i); elle a une image par f(i) et une seule qui est, à un instant donné, soit dans R", soit dans L.
Inversement, toute page de RL n'est pas nécessairement à un instant donné, l'image d'une page appartenant à un MV(b,i) quelconque.

Notons C(k) le contenu de la page k. C(k)=0 indique que tous les éléments de la page k sont égaux à zéro.

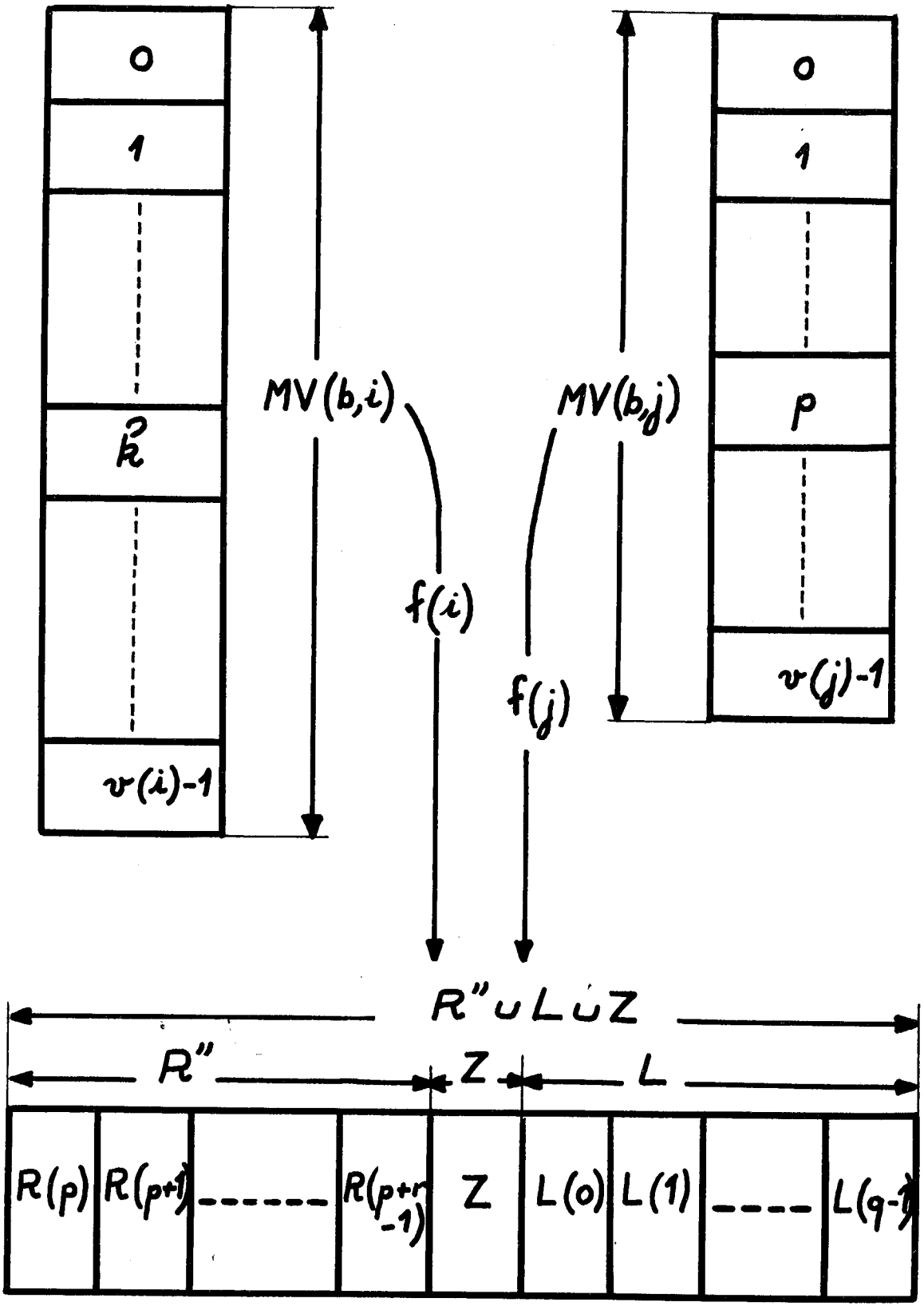

Figure 1: Images de $MV(b,i)$ et de $MV(b,j)$ dans la réunion de R'', L et Z

L'image Z d'une page appartenant à une mémoire quelconque et satisfaisant cette relation, est unique pour l'ensemble du système; nous l'appelons 'page identiquement nulle'. De façon pratique, Z peut être soit un élément particulier de L initialisé une fois pour toutes à la valeur 0, soit une entité créée dynamiquement en fonction des besoins.

Soit $i \neq j$; MV(b,i) et MV(b,j) sont par construction, deux mémoires virtuelles disjointes si, et seulement si, pour toute page k, élément de MV(b,i) et pour toute page p, élément de MV(b,j) telles que $C(k) \neq 0$ et $C(p) \neq 0$, l'image de k par f(i) est différente de celle de p par f(j) (voir figure 1).
Nous définissons ainsi deux mémoires virtuelles disjointes comme deux ensembles de pages ne pouvant avoir que l'élément Z commun, répété autant de fois que nécessaire.
Nous dirons au contraire que MV(b,i) est incluse dans MV(b,j) si tout élément (c'est-à-dire toute page) de MV(b,i) est aussi une page de MV(b,j).

De façon pratique, en supposant qu'il y ait N mémoires qui puissent être activées à un instant donné

(c'est-à-dire que leurs images sont dans la réunion de RL et de Z), il existe N suites de tables qui les décrivent.

En fait, pour représenter MV(b,i) par exemple, on trouve deux familles de tables:

-les R-tables donnant les adresses des images des pages de MV(k,i) contenues dans R",

-les L-tables indiquant les images des pages de MV(k,i) contenues dans L.

Ces tables résident en permanence dans R', zone de la mémoire réservée à l'hyperviseur. Elles sont normalement établies lors de l'initialisation d'une machine virtuelle et détruites à la fin de son utilisation. Leur contenu varie dynamiquement dans le temps en fonction du lieu de résidence instantané des pages. Dans le cas du 360/67, les R-tables sont consultées soit par le hardware pendant la phase de conversion d'une adresse virtuelle en adresse réelle, soit par l'hyperviseur qui seul, peut les modifier. Quant aux L-tables, aussi appelées 'tables des pages externes', elles ne sont manipulées que par l'hyperviseur.

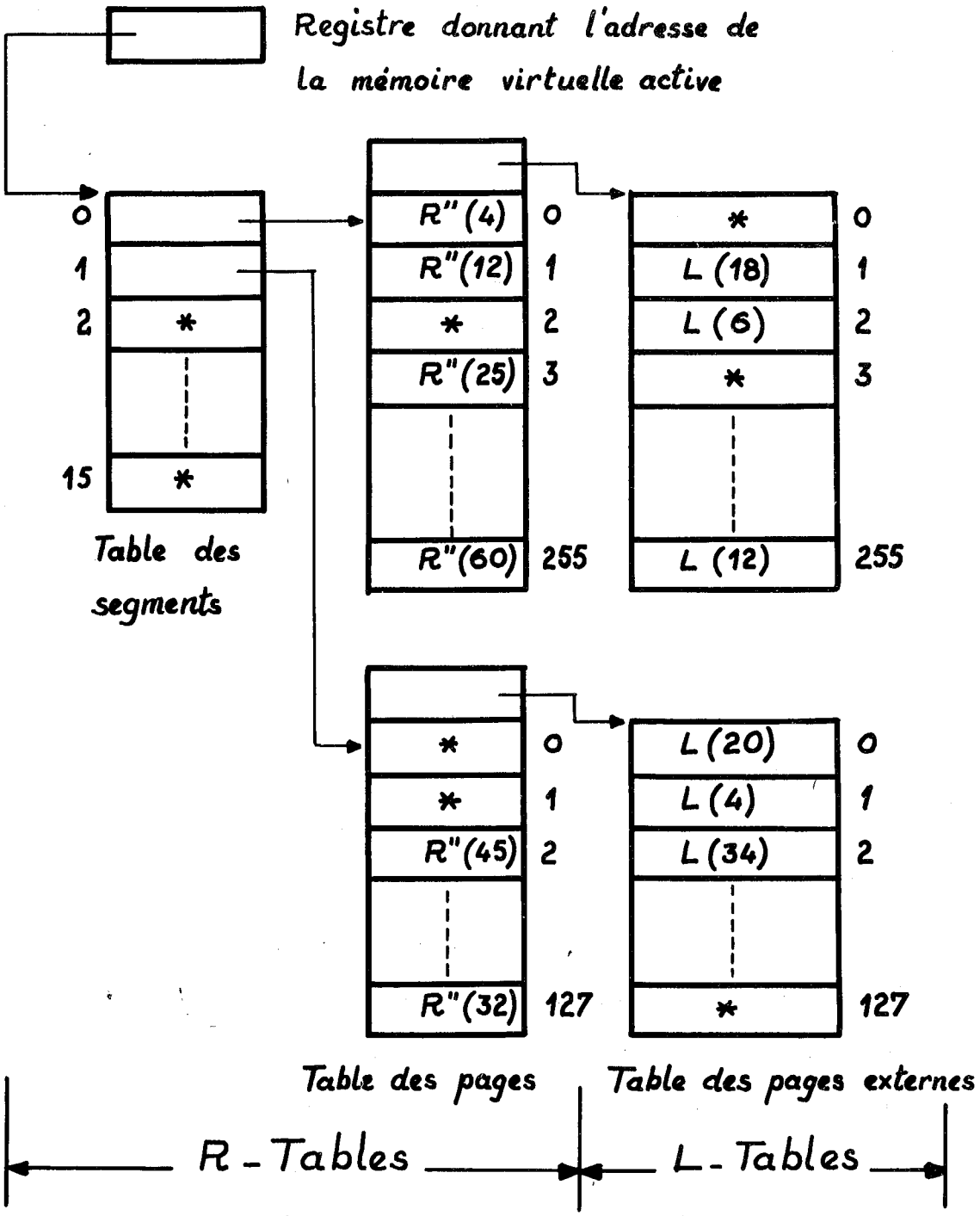

Figure 2 : Table des segments et tables des pages pour une mémoire virtuelle de 1,5 méga-caractère

Les R-tables ont ici une structure à deux niveaux et constituent l'ensemble 'table de segments/table de pages'. La table des segments décrivant la mémoire virtuelle active est indiquée par un registre du calculateur. Sur le 360/67 (ref 30, 21), une telle table comprend au plus 16 entrées, chacune d'elles donnant l'adresse d'une table de pages (ayant elle-même 256 entrées au maximum) et sa longueur.

Etant donné qu'une page comporte 4096 caractères, une table de pages complète adresse environ un million de caractères et une table de segments complète couvre l'adressage d'un peu plus de 16 millions de caractères. Il faudra donc pour décrire une mémoire virtuelle dans la mesure où il s'agit d'un espace continu:

-une seule entrée dans la table des segments pour une taille allant jusqu'à 1 méga-caractères,

-deux entrées pour une taille comprise entre 1 et 2 méga-caractères (voir figure 2) etc...

Quant à l'entrée dans la table des pages qui correspond à la kième page de la mémoire virtuelle MV(b,i), elle indique la valeur de son image dans R", si elle existe. Si elle n'existe pas à un instant donné, l'entrée correspondante dans la table des pages devient invalide.

Lorsqu'au cours de la conversion d'une adresse virtuelle en adresse réelle, les mécanismes câblés chargés de l'opération rencontrent une telle entrée, ils génèrent une interruption pour 'page manquante' et transfèrent le contrôle à l'hyperviseur. Celui-ci utilise alors les tables de pages externes L pour localiser la page qui doit être amenée dans R".

Appliquons ces définitions au problème du couplage qui nous intéresse.

Soient MV(b,j) la mémoire de la machine espion et MV(b,i) celle de l'objet:

-Les programmes de mise au point et de contrôle ne devant pas être accessibles par l'objet, il faut que la taille de la mémoire espion v(j) soit strictement supérieure à celle de l'objet.

-Pour adresser l'objet depuis l'espion de la façon la plus simple possible, il faut conserver l'ordre relatif des éléments de MV(b,i) à l'intérieur de MV(b,j).

-Les processeurs espion et objet doivent pouvoir fonctionner indépendamment. Ceci implique que les pages MV(0,i) et MV(0,j) soient distinctes;

en effet, l'hyperviseur simulant les mécanismes câblés de la machine y rangent des valeurs propres à chacune des unités centrales.

Toutes les conditions précédentes sont satisfaites si les pages de MV(b,j) qui contiennent les programmes de l'espion, constituent la zone d'adresse basse de la mémoire espion.

Nous avons donc donné à l'espion un segment 's' de plus que l'objet. Un segment joue dans le présent contexte, le rôle d'une hyperpage; il est numéroté automatiquement par l'hyperviseur. Lorsque l'espion est actif, s constitue le segment 0 de la mémoire et les autres segments sont réservés à l'objet; ce segment 0 disparaît quand l'objet fonctionne seul, sans couplage. Ceci revient à décaler les entrées de la table des segments de l'objet pour laisser la possibilité d'insérer une entrée pour s. Ce procédé n'impose qu'une contrainte: le nombre de segments de l'objet doit être au moins inférieur d'une unité par rapport au maximum nécessaire à la description d'une mémoire virtuelle.

Si plus d'un segment est inutilisé par l'objet, il est alors possible de superposer une suite d'espions de façon itérative. Cette propriété peut être utilisée pour mettre au point un espion évolué sous contrôle d'un autre plus rudimentaire mais opérationnel.

1.3.2 Inclusion des processeurs

L'inclusion des mémoires telle que nous l'avons réalisée impose que la machine-objet soit générée avant la prise de contrôle par l'espion. Pour cela, l'hyperviseur doit créer les tables définissant la configuration de l'objet en suivant les directives décrites dans le fichier catalogue des machines virtuelles.

Un certain nombre de caractéristiques sont ainsi précisées; elles concernent l'unité de traitement, la mémoire, les canaux, les unités de contrôle et d'entrée-sortie. Cette création est initialisée depuis un terminal qui devient le pupitre de la machine virtuelle objet. L'opérateur de cette machine peut alors obtenir dans la mémoire virtuelle la copie d'un système précis à l'aide d'une commande de simulation du chargement initial, encore appelée IPL ('initial program loading').

A ce stade, l'objet est sous le contrôle du système qui vient d'être initialisé et, vu par l'hyperviseur, il est une machine virtuelle standard. Quand cette machine est inactive, son état instantané apparaît dans la

structure de données constituée par l'ensemble des tables qui la décrivent. Par contre, lorsqu'elle est active, sa description instantanée est plus complexe à isoler parce qu'elle met en jeu, d'une façon variable dans le temps, le contenu de registres de la machine réelle. Or, dans notre application, nous voulons obtenir facilement le signalement instantané de l'objet; pour cela, nous avons choisi de rendre ce dernier temporairement inactif.

La procédure de prise de contrôle par l'espion est parfaitement définie. Elle consiste à émettre depuis le terminal de la machine objet, une interruption asynchrone ayant une signification analogue à la touche ARRET existant sur n'importe quel calculateur réel.
Il reste alors à demander à l'hyperviseur le remplacement de l'unité centrale de l'objet par celle de l'espion. L'état instantané de l'objet est sauvegardé pour servir de données à l'espion et l'opération se poursuit automatiquement par l'adjonction d'un segment de mémoire virtuelle supplémentaire selon le mécanisme exposé précédemment. Ce segment que nous venons d'ajouter doit être initialisé de façon à ce que son contenu corresponde aux programmes de mise au point.

Nous avons préalablement sauvegardé ces derniers sur disque, sous la forme d'une suite de pages dont l'ordre et l'emplacement sont indiqués dans une table de structure analogue à celle des tables de pages externes dont nous avons déjà parlé.
Par conséquent, l'opération d'initialisation consiste à établir une correspondance entre une table de pages externes dont le contenu est une simple copie de celle que nous venons de décrire et le segment de mise au point.

Ce procédé appelé 'IPL par nom' est extrêmement rapide car il ne met en jeu aucune opération d'entrée-sortie effective et permet ainsi de résoudre une difficulté qui se présente lors de la prise de contrôle par l'espion: en effet, à ce moment-là, l'objet qui est en cours de fonctionnement, voit son processeur artificiellement interrompu; or, il a probablement des opérations d'entrée-sortie qui se poursuivent et se terminent plus ou moins correctement; ceci implique qu'une partie des chemins qui mènent aux unités d'entrée-sortie virtuelles, sont encombrés d'interruptions en attente. Ces dernières vont être collectées par l'espion, puis distillées une à une à l'objet de façon à ne pas perturber son fonctionnement.

Il est à remarquer que ceci ne peut pas être réalisé tant que l'outillage de mise au point n'est pas initialisé; on ne dispose en effet, d'aucun mécanisme adéquat permettant d'exécuter ces sauvegardes.

Une fois l'espion initialisé, la fonction que nous avons ajoutée à l'hyperviseur relie la description de l'unité centrale de l'espion à la liste des machines susceptibles d'être activées, puis redonne le contrôle au programme de l'hyperviseur chargé d'affecter l'unité de traitement qui activera l'espion dès que possible.
Pour faire appel à l'espion, nous avons utilisé une interruption asynchrone réalisée par l'opérateur depuis le terminal de la machine-objet. En fait, cette méthode se révèle inadéquate quand le transfert de contrôle doit être réalisé à un moment précis du fonctionnement de l'objet; nous avons donc ajouté la possibilité d'émettre, par programme, un signal de signification équivalente.
Quelle que soit la méthode d'initialisation, l'espion ouvre normalement le dialogue avec le programmeur système par l'intermédiaire du terminal physique servant également de console-opérateur à l'objet. Grâce au langage de commandes que nous avons créé et dont quelques fonctions sont présentées dans la suite de ce

chapitre, l'utilisateur peut définir une certaine ligne de conduite et en particulier, redonner le contrôle à l'objet jusqu'à ce qu'une condition précise soit réalisée. Une décision de ce type entraîne le remplacement inverse de l'espion par l'objet, l'état instantané du premier étant conservé pour une utilisation ultérieure; la suppression du segment 0 et la translation de l'espace d'adressage sont pratiquement obtenues par un décalage du contenu de la table des segments.

Quand l'événement qui doit provoquer le transfert du contrôle se produit, l'état de l'objet est à nouveau sauvegardé alors que celui de l'espion est restauré;le segment supplémentaire est également ajouté.

Cette substitution périodique permettant de donner le contrôle tantôt à une machine virtuelle, tantôt à l'autre, rappelle l'échange des mots d'état-programme ayant lieu au moment d'une interruption; il s'agit en effet, d'un mécanisme analogue.

1.3.3 Inclusion des unités d'entrée-sortie

Après avoir pourvu l'espion d'une mémoire principale, d'un processeur et d'un terminal, nous lui avons donné

libre accès à un certain nombre d'unités d'entrée-sortie. Grâce à des échanges plus importants soit avec des unités rapides, soit avec le milieu extérieur, nous avons augmenté les facilités offertes par l'espion et amélioré ainsi la souplesse d'emploi de cet outil de mise au point. Nous avons par exemple, créé une gestion de fichiers sur disque, autorisé l'impression de longues listes sur des imprimantes rapides, etc...

Ces nouvelles possibilités imposent le tri correct des interruptions asynchrones indiquant la progression des opérations d'entrée-sortie. Certaines d'entre elles initialisées par l'objet, s'achèvent lorsque l'espion a pris le contrôle. D'autres commandées par l'objet sont exécutées sous le contrôle de l'espion et ce dernier enfin, en réalise à son propre profit; en outre, dans un souci de généralité, rien n'empêche l'espion d'utiliser certaines unités appartenant à l'objet.

Pour faire face à ces divers problèmes et optimiser le tri des interruptions, nous avons mis en place l'algorithme suivant: lorsque l'espion a le contrôle, toute opération réalisée pour son propre compte est exécutée sans simultanéité. Deux cas immédiatement distinguables peuvent se présenter au moment où l'interruption se produit:

-soit elle correspond à une unité sur laquelle l'espion vient d'émettre une opération d'entrée-sortie et ce dernier est alors précisément en train de l'attendre;
-soit elle est destinée à être réfléchie à l'objet.

Chaque interruption concernant l'activité de l'espion est traitée avant que celui-ci cède la place à l'objet. Par contre, celles qui se produisent lorsque l'objet reprend un contrôle total, lui sont alors toutes destinées.

Lorsque l'espion est actif et cherche à réaliser une opération sur une unité occupée (c'est-à-dire en cours d'utilisation par l'objet), il attend la fin de l'opération et sauvegarde les conditions d'achèvement pour les réfléchir ultérieurement à l'objet. Dans ce cas, et celui-ci seulement, les conditions de mauvais fonctionnement sont traitées par l'espion et non transmises à l'objet.

1.4 Outillage de mise au point

Nous avons successivement présenté et justifié l'approche adoptée pour résoudre le problème de la mise au point par l'emploi de machines virtuelles couplées, puis montré les points fondamentaux de cette solution. Intéressons-nous maintenant à ce qui permet à l'espion de jouer son rôle d'une façon efficace.

La première méthode possible consiste à interpréter en totalité et systématiquement les instructions qu'exécute l'objet tant que l'espion a le contrôle.
Pour cela, il faut disposer d'un processeur software que l'on substitue à l'unité centrale objet; dans notre cas, il s'agit d'un interpréteur du calculateur IBM 360. La construction d'un tel outil est délicate parce qu'il doit exécuter les instructions de façon strictement identique à un 360 câblé ou microprogrammé; on trouve la description, en principe exhaustive, du fonctionnement de ce programme dans le manuel de référence définissant l'architecture de la machine (ref 29). Cet outil doit également savoir gérer les différents types d'interruptions.
A l'aide de cet interpréteur, il est relativement aisé d'installer à peu près n'importe quel outil de mise au

point puisque nous contrôlons aussi bien le flot d'adresses générées par l'objet que l'ensemble des interruptions. C'est une méthode puissante mais qui se révèle coûteuse: la solution adoptée ici est basée uniquement sur l'emploi de programmes et ne bénéficie d'aucune assistance micro-programmée. Pour résoudre ce problème, nous avons ajouté un mécanisme complémentaire qui autorise l'objet à fonctionner tantôt de façon autonome, tantôt en mode purement interprétatif.

Ceci est réalisé simplement grâce aux deux conventions suivantes:

-d'une part, lorsqu'une interruption se produit sur la machine-objet, l'hyperviseur donne toujours le contrôle à l'espion et c'est ce dernier qui décide de réactiver ou non l'objet en fonctionnement autonome.

-d'autre part, l'espion peut insérer des points d'arrêt dans le programme-objet en substituant un code-opération invalide à n'importe quelle instruction exécutable; le code-opération initial et son adresse sont alors sauvegardés dans une zone de travail de l'espion. Après cette insertion, l'objet peut être remis en fonctionnement. Aucune limitation n'est imposée quant au nombre de points d'arrêt dans l'espace

objet; ils représentent autant de demandes de transfert à l'espion au moment où l'on tentera d'exécuter les codes-opérations invalides.

En effet, cette tentative provoque une interruption et entraîne le passage du contrôle à l'espion comme on l'a vu précédemment; celui-ci compare alors la valeur du compteur ordinal (partie adresse du mot d'état-programme PSW) à la liste des adresses correspondant aux points d'arrêt:

-s'il y a coïncidence entre un élément de cette liste et l'emplacement-mémoire du code opération invalide, un point d'arrêt a été effectivement rencontré; l'espion peut alors exécuter une séquence d'opérations soit prédéterminées, soit définies de façon interactive par l'utilisateur.

-si aucune coïncidence n'est constatée, il suffit de réfléchir, à l'objet une interruption adéquate. Celui-ci doit alors décider de la suite des opérations.

Le simulateur de l'architecture 360 et la pose de points d'arrêt que nous venons de décrire, constituent l'outillage de base disponible. Il reste encore à

définir le language de commandes permettant son utilisation.

Dans la machine-espion, ce language a été intégré au système conversationnel standard CMS qui offre de nombreuses possibilités; en particulier, celle qui permet d'avoir accès à sa gestion de fichiers.

Nous avons vu précédemment, qu'au moment du couplage initial entre l'objet et l'espion, ce dernier établit le dialogue avec le programmeur-système, après avoir réalisé un certain nombre d'opérations préliminaires. Nous dirons que l'espion est dans l'état 'attente de requêtes'. Ceci revient à déverrouiller le clavier du terminal qui sert également de pupitre à l'opérateur-objet. Un jeu de commandes classiques permet alors d'examiner le contenu de la mémoire de l'objet et également de décider du mode ultérieur de son fonctionnement. Il peut être en effet, soit sous le contrôle de l'espion (MONITOR ON), chacune de ses instructions sera alors simulée, soit sans aucun contrôle de celui-ci (MONITOR OFF).

Il est intéressant de noter que dans ce dernier cas, l'émission d'une interruption asynchrone de type externe depuis le pupitre de l'opérateur-objet entraîne automatiquement la reprise de contact instantanée avec l'espion.

Lorsque l'espion est dans le mode 'attente de requêtes', d'autres commandes permettent de définir des conditions de reprise de contrôle intermittente.
Ainsi, par exemple:

-BREAK rend possible la pose des points d'arrêt;

-WHEN définit les types d'interruptions qui doivent déclancher le transfert à l'espion (par défaut, il a lieu pour toute interruption).

D'une façon analogue, certaines commandes autorisent le passage depuis le mode de fonctionnement sous contrôle de l'espion imposé par MONITOR ON à un autre plus rapide dit 'de grande vitesse' (c'est-à-dire sans que l'interprétation ait lieu).
Soit, par exemple:

-STOP commande l'arrêt de la simulation sur l'occurrence d'un type particulier d'instruction ou d'une référence à une zone de mémoire préalablement définie,

-AT permet la construction d'une macro-commande (programme écrit à partir de commandes élémentaires) qui sera exécutée lors d'une référence à l'instruction située à l'adresse donnée en argument de la commande.

-GO déclanche le fonctionnement de l'objet en mode 'grande vitesse'.

Il existe environ une trentaine de commandes disponibles présentées dans les références 36 et 42.

1.5 Extensions diverses

Nous avons ajouté la possibilité d'utiliser des noms symboliques pour citer les adresses. Pour cela, nous avons forcé l'assembleur à générer des tables de symboles sous forme de fichiers d'un type particulier conservés sur disque. Ainsi, lors d'une opération de mise au point, l'espion peut acquérir ces tables pour les structurer de façon plus optimale et autoriser, par la suite, le programmeur-système à faire référence aux éléments de l'objet de manière symbolique.

Des travaux complémentaires ont été réalisés dans des directions différentes:

-Henri SAVARY, au sein d'un autre groupe de recherches, a utilisé le mécanisme de couplage en lui adjoignant d'autres outils de mise au point pour l'écriture de

systèmes dans le language de niveau élevé GSL (ref 45, 11).

-C.GATEAU et R.PERRONEAU (ref 18) ont écrit une autre version expérimentale du simulateur dans laquelle ils ont inclus les fonctions étendues du 360/67 permettant de générer des mémoires virtuelles.

D'autres extensions sont souhaitables:

La simulation induit une charge très lourde au niveau de l'unité centrale en contre-partie des avantages que nous avons vu au cours de ce chapitre. Il est toujours possible, dans un premier temps de chercher à optimiser la suite d'instructions qui doit être exécutée mais ce n'est pas suffisant. Si l'on veut voir décroître le coût de la simulation, il faut employer des mécanismes câblés ou microprogrammés. Sur des calculateurs plus récents que le 360, on peut utiliser certains outils hardware qui sont disponibles. En particulier, le mécanisme appelé PER pour 'Program Event Recording' tel qu'il existe sur le 370, permet de minimiser le recours au simulateur; on l'utilise par exemple, pour intercepter une référence à un emplacement de l'espace virtuel, ce qui jusqu'à présent était réalisé par software.

Nous aimerions aussi étendre les possibilités du mécanisme de macro-commande afin de pouvoir composer facilement des commandes évoluées à partir d'autres préalablement construites. Il s'agit en fait, d'un problème général lié à tout language de commandes.
Pour le résoudre, on peut envisager deux solutions extrêmes:

-soit, imaginer d'utiliser un language de programmation complet pour étendre les commandes de base,

-soit, utiliser un mécanisme de substitution de chaînes de caractères qui facilite l'écriture de macro-commandes.

Cette seconde approche nous a semblé plus attrayante. Nous nous sommes inspirés des travaux réalisés sous CTSS au projet MAC (probablement par Louis POUZIN), puis de ceux de Frank BELVIN et Joel WINNETT du Lincoln Laboratory sous CMS (ref 32); tout en conservant un mécanisme interprétatif, nous avons envisagé la substitution de chaînes plus élaborées analogues à celles que l'on rencontre dans le langage Snobol (ref 20).

En continuant à enrichir cet outil, nous prenons le risque d'aboutir à un macro-mécanisme plus complexe,

donc à un temps d'exécution plus long puisqu'on accroît à la fois la quantité d'instructions exécutées et la pagination.

Pour qu'un tel mécanisme soit viable, il faut qu'il soit à la fois facile à mettre en oeuvre (ce qui est le cas de la version d'EXEC sous CMS par exemple) et aussi peu coûteux que possible en termes de ressources consommées.
Ce second point n'a pas été jusqu'ici tout à fait atteint et dans un travail en cours, Max CREVEUIL (ref 12) cherche à résoudre ce problème en isolant les primitives et en tentant d'améliorer les temps d'exécution par tous les moyens, y compris la microprogrammation.

En conclusion, le fonctionnement de ce système de couplage de machines virtuelles appliqué à la mise au point, nous a montré que:

- l'idée était intéressante et conduisait à un outil très souple et puissant. D'autres réalisations basées sur le même principe sont d'ailleurs actuellement en cours en dehors de Grenoble.

-d'autre part, l'existence d'un simulateur optimisé, au point et complet, donnait une souplesse extraordinaire à l'outil. En effet, grâce à lui, il était possible d'ajouter facilement presque toutes les commandes que l'on peut imaginer.

Analyse du comportement de l'hyperviseur CP67 par transferts unidirectionnels entre l'hyperviseur et les machines virtuelles

---

2.1 Présentation

Un des autres aspects de nos recherches nous a conduit à étudier et à réaliser des mécanismes de transfert permettant à une machine virtuelle d'acquérir des informations contenues dans l'espace d'adressage de l'hyperviseur. Ces travaux font l'objet de ce deuxième chapitre.

L'idée d'inclusion a été introduite pour trouver une solution à un problème pratique de mise au point; de façon similaire, les mécanismes de transferts auxquels nous allons nous intéresser, ont été élaborés pour voir l'origine des irrégularités dans les performances du calculateur de l'IMAG, de les expliquer et tenter de les résoudre.

En effet, dès le début de 1969, nous avons constaté l'apparition de brusques baisses de performances sur l'ordinateur 360/67 au cours de son fonctionnement sous le contrôle de CP67. Elles se produisaient à intervalles irréguliers et de façon imprévisible. Ces

troubles restaient malgré tout peu génants car ils étaient assez rares et de durée suffisamment brève pour permettre à l'ensemble du système de rendre un service global raisonnable.

Après nous être renseignés auprès d'autres utilisateurs de ce système, nous nous sommes rendus compte que le phénomène ne semblait apparaître que sur les machines de 'petite configuration' similaires à celle de Grenoble, à cette époque. La possibilité d'un mauvais fonctionnement de la fonction appelée 'dispatcher' nous avait alors été suggérée. Le rôle de cette dernière est de déterminer le sous-ensemble des tâches qui, à un instant donné, sont en concurrence pour utiliser l'unité centrale. En d'autres termes, ces tâches participent à la mise en oeuvre de la multiprogrammation. Le terme 'tâche' est utilisé ici, comme synonyme de 'machine virtuelle'.

Ne connaissant personne qui travaillait alors sur ce sujet, il nous a paru constituer un domaine de recherche intéressant. Il pouvait en particulier, avoir une incidence pratique car si nous trouvions une explication et une solution à ce problème, le fonctionnement du système disponible à l'IMAG ne pouvait qu'en être amélioré.

La première difficulté provenait de ce qu'il fallait cerner les causes du ralentissement alors que nous ne savions pas dans quelles conditions exactes, il se produisait.

Nous avons donc décidé d'enregistrer le plus grand nombre possible d'informations sur le comportement du système et de les visualiser sur un terminal à intervalles réguliers (de l'ordre de quelques dizaines de secondes). Nous nous sommes attachés ensuite à repérer les périodes de semi-blocage et à partir de là, tenter d'établir une corrélation entre la notion vague de mauvais fonctionnement et les valeurs recueillies pour les différents paramètres.

Cette 'campagne' de mesures que nous cherchions à initialiser consistait à prélever les valeurs de plusieurs variables appartenant à l'hyperviseur; il s'agissait donc d'une analyse ponctuelle.

Mais cette méthode rendait difficile, voire impossible, l'isolement de la suite quasi exhaustive des paramètres critiques. Il fallait trouver un mécanisme complémentaire qui permette de prendre des instantanés de l'ensemble du système au moment même où l'on identifiait de façon non ambiguë, un ralentissement anormal. On pouvait réaliser ainsi une analyse globale du comportement.

Dans ce qui suit, nous présentons la façon dont nous avons réalisé ces deux types d'examen.

2.2 Analyse ponctuelle

Nous avons tout d'abord pensé modifier l'hyperviseur et lui seul, afin d'y insérer les outils de mesures dont nous souhaitions disposer. Un examen rapide de cette solution nous a conduits à l'abandonner partiellement.
L'hyperviseur est un programme de taille moyenne par rapport aux autres systèmes mais d'un bon niveau de complexité; à cette époque, il occupait 80.000 octets de mémoire.
L'une des raisons majeures de cette complexité tient à ce que parmi les fonctions qui lui sont dévolues, l'une d'entre elles consiste à gérer de façon aussi optimale que possible, l'ensemble des ressources physiques connectées à l'ordinateur; or, les unités d'entrée-sortie, de par la diversité de leur comportement, sont peu faciles à administrer.
D'autre part, l'hyperviseur est très fréquemment sollicité (plusieurs dizaines de fois par seconde) et l'expérience nous a montré que toute modification imparfaitement mise au point réussissait généralement à

'tuer' le système après un temps très court de fonctionnement.

Nous retrouvons ici le problème évoqué au chapitre précédent mais à ce moment là, les mécanismes de couplage n'étaient pas complètement disponibles.

D'autre part, nous n'avions pas encore la possibilité d'activer une nouvelle version de l'hyperviseur en cours de modification sur une machine virtuelle générée par une précédente version du même hyperviseur. Ce n'est que vers la fin de l'année 1969, à la suite de travaux réalisés aux Etats-Unis, en partie par Alain Auroux (ref 5), qu'il est devenu possible de faire fonctionner l'hyperviseur sous contrôle de lui-même.

Ce manque de moyens imposait donc l'immobilisation de l'ensemble des ressources de la machine réelle en les affectant à une seule personne qui travaillait alors directement au pupitre du calculateur afin de réaliser la mise au point d'une version modifiée de l'hyperviseur. Cette solution n'étant envisageable que pendant quelques heures de week-end, la progression du travail était beaucoup trop lente pour autoriser des modifications majeures. Le 360/67 était en fait trop peu disponible pour l'emploi de telles méthodes.

Cette contrainte nous conduisit à trouver une solution n'entraînant que des rectifications mineures au niveau

de l'hyperviseur. La plus grande partie de l'outillage de mesures ponctuelles était reportée au niveau d'une machine virtuelle spécialisée. Nous pouvions alors nous permettre la mise au point d'algorithmes complexes puisqu'il est possible d'arrêter une machine virtuelle sans que cela ait la moindre incidence sur la machine réelle.

Nous avons alors décidé d'opérer de la façon suivante:

-D'une part, nous insérons dans l'hyperviseur des compteurs d'événements en tous genres (certains d'ailleurs s'y trouvant dès l'origine) et les instructions de mise à jour automatique sans que le fonctionnement du système en soit affecté;

-D'autre part, une machine virtuelle relève à intervalles de temps réguliers, les valeurs contenues dans les compteurs puis se charge de l'exploitation de ces paramètres.

Nous posions ainsi le problème de la communication entre une machine virtuelle et l'hyperviseur. Par définition, un générateur de machines virtuelles ne comprend aucun mécanisme autorisant des échanges explicites entre les diverses parties en présence.

Par définition aussi, tout ce qui fonctionne sur une machine réelle doit pouvoir, dans la mesure du possible, être activé sur une machine virtuelle. Ainsi, si l'on admet quelques restrictions dues à des distorsions de l'échelle de temps, on refuse toute modification au niveau de l'architecture simulée: donner par exemple, une signification à un des codes-opération qui semblent inutilisés est un choix dangereux puisqu'ils sont en fait, réservés à des extensions ultérieures.

De ce fait, le choix d'un moyen de communication ne pouvait donc pas se faire d'une façon quelconque et sans précaution particulière. La solution que nous avons adoptée a été construite à l'origine en collaboration avec Robert Adair du Centre Scientifique de Cambridge (Etats-Unis) au cours de nombreuses discussions sur ce sujet.

2.2.1 Communication entre machine virtuelle et hyperviseur

La demande de transfert d'informations entre machine virtuelle et hyperviseur a été réalisée en développant

l'emploi d'une instruction particulière prévue par 'l'Architecture 360'.

Celle-ci est définie de façon très précise ainsi que nous l'avons déjà dit, dans le manuel IBM intitulé 'IBM System/360 Principles of operation' (ref 29); elle est adaptée (on dit 'émulée') à des unités centrales très différentes les unes des autres aussi bien du point de vue des performances que de celui de la complexité. Ceci permet de disposer d'une gamme complète d'ordinateurs, en fonction de la puissance et du coût. Les unités de traitement sont en effet conçues de façon totalement indépendante; l'émulation est réalisée en faisant appel à un nombre plus ou moins important de dispositifs câblés et en utilisant ou non la microprogrammation.

Sans donner plus de détails car nous quittons ici le domaine de la programmation pour atteindre celui de la conception des ordinateurs, et là n'est pas notre propos, nous pouvons simplement constater que nous sommes en présence d'une superposition de deux calculateurs: l'un est un 360 répondant à la définition du manuel précité; l'autre, sous-jacent, est chargé d'adapter le premier au type d'unité centrale dont on dispose et il est par définition, complètement transparent à l'utilisateur normal.

Il est important par contre, que cette machine 'invisible' puisse être contrôlée pour s'assurer de son fonctionnement. Pour cela, une instruction privilégiée exécutable dans le mode superviseur seulement permet de radiographier les 'rouages' du calculateur: c'est l'instruction 'diagnose'.

Si l'on considère maintenant ce qu'est une machine virtuelle vis-à-vis de l'hyperviseur, on trouve une analogie certaine avec la notion qui associe un ordinateur 360 et le processeur qui l'émule. Il a semblé alors raisonnable, en prenant quelques précautions, de généraliser l'instruction 'diagnose' dans ce contexte particulier pour examiner et éventuellement, modifier la suite de programmes de l'hyperviseur. Ceux-ci sont normalement 'invisibles' aux différentes machines virtuelles générées.

Ce principe étant acquis, il suffit alors de définir les paramètres à associer à la nouvelle instruction pour rendre possible l'examen de l'hyperviseur par des systèmes virtuels. Nous appelons ici 'système virtuel' (ref 10), un système qui a la propriété exceptionnelle de savoir reconnaître qu'il fonctionne dans un environnement privilégié et non directement sur une machine réelle.

Dans le cas des analyses ponctuelles, nous attendons de l'instruction 'diagnose' un simple prélèvement d'informations dans l'espace d'adressage de l'hyperviseur et un transfert dans celui de la machine virtuelle.

A priori, la forme exacte de l'instruction retenue pour transmettre les valeurs des paramètres depuis l'hyperviseur vers les systèmes virtuels était très classique: deux adresses appartenant respectivement aux espaces d'adressage émetteur et récepteur, ainsi que la longueur de l'information à transmettre étaient suffisantes. Toutefois, avec le temps, ce moyen de communication, s'est largement développé et systématisé au point de devenir une méthode de base pour établir des échanges entre des systèmes virtuels et l'hyperviseur.

Il a donc fallu formaliser la définition de cette instruction 'diagnose généralisée' en lui associant un facteur qui précise le type de fonction que l'on souhaite voir exécutée (ref 31).

Ainsi, dans sa forme générale, l'instruction 'diagnose' joue le rôle d'une clé qui permet l'examen et, dans

certains cas très contrôlés, la modification de l'hyperviseur.

Par ce biais, on dispose d'un répertoire complet de nouvelles instructions spécialisées dans les communications entre machines virtuelles et générateur. Il s'agit en fait, d'une forme transposée d'adressage indirect puisque l'instruction exécutée par le processeur virtuel définit le code de la fonction à réaliser sur la machine sous-jacente, c'est-à-dire l'hyperviseur.

2.2.2 Reconnaissance d'un contexte réel ou virtuel

La généralisation de l'instruction 'diagnose' introduit toutefois un problème de reconnaissance de l'environnement: en effet, si le système virtuel est susceptible de fonctionner non seulement sous le contrôle de l'hyperviseur mais également sur une machine réelle, il faut qu'il puisse identifier le contexte dans lequel il se trouve. De plus, l'instruction 'diagnose' ne dispose pas d'une fonction commune permettant par exemple, d'identifier le modèle de la machine ou de repérer l'existence de l'hyperviseur.

Il a donc fallu trouver un artifice qui bien qu'imparfait, s'est révélé suffisant: étant donné qu'il est possible d'attacher une unité d'entrée-sortie à une machine virtuelle de façon statique ou dynamique, nous avons décidé de lui attribuer une unité virtuelle qui, par convention, a la plus grande adresse disponible et acceptable sur l'un des canaux. Ceci repose sur l'hypothèse très réaliste qu'aucun système fonctionnant sur une machine virtuelle n'ait effectivement besoin de cette adresse, ce qui jusqu'à présent, semble toujours s'être vérifié.

Pour reconnaître alors son contexte, le système virtuel commande une opération élémentaire d'entrée-sortie sur cette pseudo-unité. S'il reçoit une réponse positive, la présence de l'hyperviseur lui est confirmée; sinon, il croit fonctionner hors de son contrôle. Un autre intérêt de cette solution est dû au fait que n'importe quelle unité virtuelle peut être détachée dynamiquement: on obtient ainsi un fonctionnement plus subtil qui consiste à faire travailler le système virtuel sous le contrôle de l'hyperviseur tout en lui laissant croire que ce dernier est absent. Cette façon de procéder qui semble tout d'abord aller à l'encontre de la raison pour laquelle nous avons créé une pseudo-unité, permet la mise au point, sous contrôle de

l'hyperviseur, de certaines parties du système virtuel conçues pour ne fonctionner qu'en contexte réel.
L'addition ou le retrait d'une pseudo-unité imaginée pour résoudre un problème spécifique a donné lieu à des extensions nombreuses et intéressantes. On peut, en effet, considérer que cette unité n'existe pas ou est en cours de conception. Par le biais d'une simulation programmée, on peut alors entreprendre la mise au point de systèmes employant cette unité non disponible. On peut également, en changeant les programmes de simulation, modifier et affiner la logique d'une telle unité évitant ainsi d'avoir à changer plus tard une partie de son câblage, opération moins simple et beaucoup plus coûteuse.

2.2.3 Transfert de pages entre l'hyperviseur et la machine virtuelle.

Ainsi que nous l'avons vu, l'instruction 'diagnose' permet le transfert d'informations entre l'espace mémoire propre à l'hyperviseur et celui de la machine virtuelle. Cette opération peut également être réalisée à l'aide d'une autre méthode qui utilise le mécanisme de pagination.

L'hyperviseur peut résider en mémoire virtuelle ou en mémoire réelle; sur le 360/67, c'est la seconde solution qui a été retenue pour des raisons de performances. Ainsi, dans notre cas, lorsque l'hyperviseur s'exécute, la conversion automatique d'adresses virtuelles en adresses réelles est supprimée. Il est à remarquer que l'on aurait pu disposer d'une table de segments et d'une table de pages décrivant le même espace et dont la propriété eût été d'avoir des adresses virtuelles identiques aux réelles, la fonction de conversion appliquée étant la fonction d'identité. Toutefois, ceci aurait conduit à un ralentissement d'environ 20% de l'unité de traitement du 360/67.

Cette perte de performance est liée au mécanisme de conversion qui impose tout d'abord une recherche dans une série de registres associatifs (ref 21) pour déterminer si la partie 'segment/page' de l'adresse n'a pas été récemment traduite; ceci dure approximativement 150 nanosecondes. Si aucune correspondance n'est trouvée, la consultation de la table des segments et de la table des pages provoque alors une double référence à la mémoire pour effectuer le calcul d'adresse; le coût est évalué à 1,5 microsecondes.

C'est pour cette raison que des sytèmes ayant un rôle équivalent à celui de l'hyperviseur du point de vue fonctionnel, tels que UMMPS dans MTS (ref 4) ou le superviseur résidant de TSS (ref 34) suppriment aussi le mécanisme de conversion.

La résidence de l'hyperviseur en mémoire virtuelle offre pourtant une grande souplesse et devrait à notre avis, être utilisée toutes les fois que les pertes de performances sont acceptables (comme pour le 370 par exemple, elles sont évaluées entre 3 et 5%).
En effet, au début de l'implantation du système, on peut disposer d'une fonction de transformation identité (Virtuel = Réel); puis, quand l'hyperviseur devient trop volumineux, on ne laisse alors en mémoire réelle que les pages les plus fréquemment utilisées.
Nous obtenons ainsi une situation optimale de fonctionnement puisqu'un réglage très fin permet d'ajuster le nombre des fonctions résidentes sur des mémoires réelles un peu trop étriquées.
Indépendamment de ce problème de performances, lorsque l'hyperviseur réside entièrement en mémoire, cette solution permet de le réorganiser sans le forcer à occuper une zone contiguë dans la partie basse de cette

dernière (zone dont l'adresse origine est 0). Il s'agit certes, d'une utilisation dégénérée de la mémoire virtuelle puisqu'on fait alors fi de l'une de ses principales propriétés qui est d'offrir un espace d'adressage de taille indépendante et largement supérieure à celui de la mémoire réelle.

Les raisons pour lesquelles l'hyperviseur ne réside pas en mémoire virtuelle étant explicitées, le transfert de pages entre lui et une machine virtuelle prend alors une forme assez particulière puisque les notions de tables de segments et de pages ne s'appliquent pas à l'hyperviseur.
Le terme de 'page' devient alors synonyme d'un bloc de 4096 caractères dont le premier a pour adresse un multiple de 4096. Pour qu'une page de l'hyperviseur soit accessible par une machine virtuelle, il suffit d'insérer son adresse de début dans une entrée particulière de la table des pages qui décrit la mémoire de cette machine.
Cette seule modification dans une table résidant en mémoire rapide suffit à donner l'accès à un bloc de 4096 octets appartenant à l'hyperviseur.
Cette technique qui permet donc le partage d'une zone de mémoire et non sa duplication est suffisante dans la

mesure où la machine virtuelle se contente d'explorer la zone sans jamais la modifier. Si pour une raison quelconque, une telle tentative apparaît, les éléments câblés du 360/67 doivent rejeter cet essai; toutefois, un tel contrôle signifie que l'on interdit à la machine virtuelle d'utiliser une ressource particulière, à savoir la protection de la mémoire employée pour empêcher toute altération de la zone commune. Si une telle restriction est admissible pour un système virtuel non destiné à fonctionner sur une machine réelle, elle viole malgré tout le principe même de la génération des machines virtuelles puisqu'elle restreint le domaine des ressources qui peut leur être affectées.

L'expérience a montré que l'on ne peut se contenter, en général, d'un partage en lecture seulement; il devient alors nécessaire de réaliser la duplication. Celle-ci est faîte suivant des méthodes différentes selon le lieu de résidence de la page de l'hyperviseur destinée à être recopiée;

-Si la page est en mémoire réelle, il faut:

-forcer une opération de pagination de la mémoire réelle vers un support externe en utilisant les mécanismes existants au sein de l'hyperviseur,

-insérer l'adresse de ce bloc d'informations sur support externe dans la table de pages externes de la machine virtuelle;

-invalider enfin, l'entrée de la table des pages correspondante.

-Si la page est déjà sur un support externe, on peut alors se contenter d'écrire son adresse dans la table des pages externes à condition toutefois de mettre en place un indicateur particulier.

Ultérieurement, lorsqu'une référence sera faite à cette page par la machine virtuelle, l'entrée invalide correspondante dans la table des pages produira une interruption. Après exploration de la table des pages externes, le mécanisme de pagination amène en mémoire réelle une copie du bloc à partager avec l'hyperviseur. Celle-ci pourra alors être consultée et modifiée par la machine virtuelle.

Lorsque par la suite, la place occupée par cette page devra être libérée, il faudra ranger un nouveau contenu sur le support externe; pour cela, l'indicateur dont nous venons de parler précédemment, précise si les mécanismes de rangement de l'hyperviseur doivent rechercher un emplacement disponible, différent de celui qui a été précédemment utilisé.

Ce procédé de duplication minimise le nombre d'opérations d'entrée-sortie et diffère la création d'un second exemplaire de la page copiée jusqu'au dernier moment.

Remarquons que cette procédure peut être précédée par l'examen d'un autre indicateur permettant de savoir si la copie de la page qui est actuellement en mémoire centrale a été ou non modifiée. Si elle est restée identique à l'original, il est inutile de rechercher un emplacement sur le support externe et de la dupliquer.

Dans les analyses ponctuelles, nous utilisons une forme simplifiée des différentes possibilités de la pagination pour copier une page de l'hyperviseur en mémoire virtuelle: nous avons inclus dans la mémoire de cette machine une table d'adresses permettant de localiser les compteurs en mémoire réelle. Ainsi, toute ou partie de la table devient le paramètre de l'instruction 'diagnose' à l'instant du relevé des valeurs.

2.2.4 Principe de fonctionnement d'une machine virtuelle de mesures.

La machine qui exploite l'ensemble des mécanismes que l'on vient de décrire, est contrôlée par le système virtuel CMS. Son initialisation, généralement réalisée par l'opérateur, pourrait être effectuée automatiquement par l'hyperviseur lorsqu'il prend le contrôle de la configuration réelle.
Cette première étape achevée, le système virtuel CMS exécute une macro-commande particulière de la machine appelée 'PROFILE' (ref 6). Celle-ci contient une suite de commandes nécessaires à la mise en oeuvre de la machine de mesures; il s'agit en fait, de charger en mémoire les procédures qui vont diriger le travail de prélévement et d'exploitation des valeurs recueillies dans l'hyperviseur.
Rappelons que cette technique d'enchaînement automatique d'une suite de commandes prédéfinies est un mécanisme standard de CMS. Elle permet, entre autres, de masquer totalement le langage de commandes élémentaires du système et d'offrir des macro-commandes dont l'une d'entre elles peut être par exemple, un interpréteur d'une suite de fonctions propres à une application.

La machine de mesures peut alors acquérir les adresses des variables réparties au sein de l'hyperviseur puis en obtenir les contenus à l'aide de l'instruction 'diagnose' généralisée. Pour réitérer le processus de prélévement, elle insère ensuite dans son compteur d'intervalles de temps, une valeur qui sera régulièrement décrémentée par l'hyperviseur, et se met en état d'attente tout en restant prête à accepter des interruptions de type externe.

Lorsque le compteur d'intervalles de temps devient négatif, l'hyperviseur propose alors au processeur virtuel une interruption de type externe. Celui-ci l'attendait et exécute une nouvelle fois l'instruction 'diagnose' selon le schéma que nous avons précédemment décrit. Nous obtenons ainsi d'autres valeurs et l'on peut explorer cette suite d'observations de façon différée et immédiate.

Chaque lot de valeurs recueillies est sauvegardé dans un fichier qui, après une 'campagne de mesures', est destiné à être exploité par des programmes de statistiques pour détailler le fonctionnement de l'hyperviseur durant la période surveillée. Par contre, certaines des valeurs sont extraites du lot recueilli pour subir un traitement élémentaire immédiat afin d'être visualisées; ceci permet d'obtenir une image

succinte du fonctionnement de l'hyperviseur au cours de la même période.
Ainsi, la machine reste active aussi longtemps que nous le désirons et la fréquence de prélévement des mesures n'est fonction que de la valeur de l'intervalle rangé dans son compteur de temps; cet intervalle peut varier de quelques secondes à plusieurs minutes suivant le degré de finesse de surveillance que l'on veut avoir.

La technique que nous avons adoptée, impose quelques précautions élémentaires:

-Pour que l'information recueillie dans les différents compteurs soit cohérente, il ne faut pas que la machine de mesures perde le contrôle pendant l'exécution de l'instruction 'diagnose'. Ceci peut se produire si la zone de mémoire virtuelle mise en jeu par l'opération n'est pas entièrement résidente en mémoire réelle au même instant. De ce fait, une ou plusieurs opérations de pagination seront nécessaires pour amener les pages manquantes et, pendant ce temps, l'hyperviseur donnera le contrôle à d'autres tâches, et par conséquent, les compteurs seront modifiés.

Pour se prévenir d'un tel phénomène, nous imposons à l'instruction 'diagnose' et aux zones de réception appartenant à l'espace virtuel de résider globalement dans une seule et même page. Cette condition est vérifiée par l'hyperviseur pendant l'exécution de l'instruction 'diagnose'.

-Il faut éviter qu'une rupture brutale de fonctionnement du système entraîne la perte des valeurs collectées au cours de l'expérience; la machine de mesures doit donc signaler au système virtuel CMS, après chaque sauvegarde dans le fichier, que ce dernier est à conserver; pour cela, il exécute une opération de 'fermeture' du dit fichier.

2.2.5 Présentation succinte des résultats fondamentaux obtenus grâce à ces mécanismes de transfert.

L'exploitation des valeurs collectées s'est prolongée pratiquement sans interruption à l'IMAG depuis la mise en place du procédé.

De nombreuses améliorations ont été apportées par plusieurs personnes d'une part, pour affiner l'affichage des paramètres qui nous intéressent sur un

petit écran permettant ainsi de disposer d'un véritable 'tableau de bord' du système et d'autre part, pour présenter les résultats complets sous forme de courbes.

Quant aux compteurs insérés dans l'hyperviseur, ils sont nombreux (de l'ordre de plusieurs dizaines) et fournissent des renseignements sur deux grandes classes d'événements:

-l'activité de la machine réelle,

-le comportement des machines virtuelles.

Dans le premier cas, nous enregistrons, en particulier:

-le comportement de l'unité centrale en évaluant les temps passés en modes superviseur, problème et attente;

-l'activité de la pagination (nombre de pages lues et ré-écrites, longueur de la file d'attente sur le canal du tambour réservé aux opérations de pagination;

-le nombre des opérations d'entrées et de sorties initialisées sur les canaux des disques contenant les fichiers des utilisateurs et les fichiers du système réservés aux fonctions de 'spooling';

-l'occupation du canal multiple sur lequel sont connectées les unités lentes comme les terminaux,

le lecteur de cartes, les imprimantes, etc...

-le nombre d'appels de certaines séquences d'instructions critiques de l'hyperviseur,

-etc...

Dans le second cas, nous relevons de façon analogue le comportement des machines virtuelles; on peut également s'intéresser au fonctionnement d'un système virtuel particulier (travaux de Jacques Leroudier (ref 38) et Philippe Potin).

Dès le début de sa mise en fonction, la machine de mesures nous a permis d'observer une série de phénomènes intéressants. Au moment où se produisaient les baisses de performances dont nous avons parlé au début de ce chapitre, le temps passé en mode problème (temps pendant lequel les machines virtuelles sont actives) qui oscillait entre 30 et 40%, diminuait de façon brutale. En contre-partie, le temps utilisé par l'hyperviseur devenait de plus en plus important. Il fallait donc trouver la raison de ce déséquilibre.
Nous avons également observé un taux de pagination anormalement élevé (supérieur à 60 pages par seconde).

Or,le coût d'une opération élémentaire de pagination a pu être évalué à 5 millisecondes; 30% au moins du temps était dans un tel cas, consommé par la fonction de pagination.

Durant cette phase critique, une autre variable fondamentale de l'hyperviseur atteignait une valeur très élevée (12). Il s'agit du niveau instantané de multiprogrammation qui représente le nombre de tâches concurrentes que le 'dispatcher' a rendues activables et qui sont prêtes à utiliser l'unité centrale. Pour observer l'influence de ce paramètre, nous l'avons fait varier expérimentalement pendant les périodes de forte charge: le résultat le plus spectaculaire montre que l'augmentation de sa valeur fait disparaître le temps d'attente alors que celui du superviseur atteint 95 à 98%, le faible pourcentage restant représente le temps problème. L'explication devient alors évidente: l'augmentation du nombre de tâches susceptibles d'être activées, entraîne une diminution importante de la zone mémoire affectée à chacune d'elles. Aussi, lorsqu'une tâche prend le contrôle, elle ne peut le garder qu'un temps très court car une interruption pour page manquante l'empêche de continuer. La fonction de pagination cherche une place en mémoire réelle pour y

transférer la page manquante. Or, aucun emplacement n'est libre, la fonction qui décide de la stratégie à suivre, élimine une page qui trop rapidement se révèle à nouveau utile. Pendant ce transfert, une autre tâche prend le contrôle et son comportement est analogue à celui de la précédente.

En résumé, le 'dispatcher' active des tâches qui doivent être abandonnées presque immédiatement parce qu'il leur manque des pages; quant à la pagination, elle n'est occupée qu'à réaliser des transferts en enlevant des pages qui n'auraient jamais dû l'être.

En fait, ce phénomène n'apparaît qu'en présence d'une charge trop importante sur une mémoire réelle de petite taille (le 360/67 de l'IMAG n'avait alors que 512 K); on explique ainsi l'absence de ce phénomène sur des configurations plus conséquentes.

Il est raisonnable bien entendu, de s'attendre à ce que les performances diminuent jusqu'au point même de devenir insupportables quand on amplifie la charge. Pourtant, cet accroissement n'est pas linéaire puisqu'à partir d'un certain seuil, les fautes de pages augmentent de façon extrêmement rapide comme nos mesures l'ont montré.

Ce problème dont souffrent tous les 'systèmes à pages' a été bien étudié dans de nombreux centres depuis l'époque de ces premières observations. Il s'agit du 'thrashing'; ce terme difficilement traduisible, définit l'espèce d'agitation stérile qui s'empare du système lorsque le niveau instantané de multiprogrammation induit une pagination trop importante.

En s'appuyant en particulier, sur le travail de thèse de DENNING (ref 13) et sur les principes de localité et de 'working-set', nous avons pu établir les bases d'un 'dispatcher' permettant d'éliminer ce phénomène.

Jean-Pierre DUPUY et Juan RODRIGUEZ ont alors pris ce travail à leur charge et publié les résultats (ref 16, 17, 43, 44).

2.3 Analyse globale

Ainsi que nous l'avons dit dans la présentation de ce chapitre, nous avions l'intention de faire appel à d'autres mécanismes de mesures au moment précis où l'on détectait un 'goulot d'étranglement' dans le

fonctionnement du système. Nous voulions en effet, prendre un instantané de toute la mémoire réelle du calculateur. Il s'agit d'un énorme volume d'informations dans lequel il faut reconnaître les structures existantes afin de pouvoir les exploiter.
Une première alternative nous a conduits à déclancher manuellement l'opération après avoir consulté les résultats de l'analyse ponctuelle sur le 'tableau de bord' de l'hyperviseur. Ultérieurement, nous l'avons rendu automatique en couplant la prise d'instantanés à la valeur de certains paramètres critiques.
De cette information ainsi rendue disponible, nous espérions trouver des idées intéressantes en recherchant ce qui pouvait sembler anormal.
L'implantation de ce procédé a impliqué un effort de programmation beaucoup plus important que celui qui avait été nécessaire pour la réalisation de l'analyse ponctuelle.

2.3.1 Création de l'image-mémoire du calculateur

Pour prendre l'instantané du fonctionnement du système, nous avons cherché à généraliser un mécanisme très efficace qui avait été créé pour faciliter la mise au

point de l'hyperviseur. Quand celui-ci détecte une condition anormale, non pas en ce qui concerne les performances, mais à propos d'un mauvais fonctionnement logique, une procédure spécialisée recopie le contenu de la mémoire réelle sur disque et initialise une séquence de chargement d'une nouvelle version du système. Celle-ci prend alors le contrôle tout en reconnaissant que l'hyperviseur occupait précédemment la machine; on procède ainsi à un démarrage 'à chaud' c'est-à-dire que tout ce qui était destiné à être perforé ou imprimé au moment où le système a interrompu son fonctionnement, sera normalement traité par la nouvelle version. De même, les informations qui sont en attente de lecture par des machines virtuelles ainsi que les données de comptabilité sont sauvegardées.
L'opération dure entre 10 et 20 secondes pendant lesquelles les lignes de transmissions sont automatiquement ré-initialisées.

Dans le cas qui nous intéresse, nous pouvons utiliser la fonction de copie de la mémoire réelle et empêcher le chargement d'une nouvelle version du système au moment où l'on détecte une baisse de performances.
L'image-mémoire est écrite sur disque dans des zones réservées à l'hyperviseur. Ces dernières, de l'ordre de

plusieurs dizaines de millions de caractères, peuvent admettre plusieurs copies de la mémoire réelle avant d'être saturées. Les fichiers ainsi créés par l'hyperviseur sont accessibles ultérieurement par une machine virtuelle spécialisée.

2.3.2. Acquisition de l'instantané

Pour analyser une image-mémoire auparavant sauvegardée (suivant le mécanisme décrit dans le paragraphe précédent), la machine virtuelle spécialisée, sous le contrôle d'un programmeur-système, émet simplement une opération d'entrée-sortie élémentaire sur l'unité réservée.

Cette image est entièrement contenue en mémoire virtuelle pour en faciliter l'examen; la mémoire de la machine spécialisée doit donc être suffisamment vaste pour contenir les outils nécessaires à l'exploitation des données (soit 256K ou 512K) et le contenu de la mémoire réelle. Tout élément de cette dernière est directement accessible à un facteur de translation près, qui a pour valeur l'adresse en mémoire virtuelle du premier octet de l'image.

Pour accélerer cette acquisition d'images depuis l'unité d'entrée-sortie réservée nous avons construit, la suite des adresses des pages contenues sur ce support externe. Il suffit alors de les transmettre à l'hyperviseur à l'aide d'une opération 'diagnose' généralisée pour les insérer dans la table des pages externes de la machine virtuelle.
Cette technique qui supprime la recopie de l'image réelle en mémoire virtuelle, est beaucoup plus économique que la précédente.

2.3.3 Reconnaissance des structures de l'hyperviseur

L'image obtenue doit pouvoir être examinée sur un terminal évolué comme par exemple, un écran cathodique sur lequel on visualise plusieurs dizaines de lignes simultanément en quelques fractions de seconde.
L'exploitation de ces données implique la disponibilité de tout un sous-système spécialisé. Ce travail proposé à NGUYEN-THANH-THI a été décrit dans sa thèse (ref 40); le lecteur intéressé par plus de détails, peut s'y reporter.

A l'aide de cet outil, le programmeur système peut rechercher les paramètres critiques en appliquant ses connaissances de la structure interne de l'hyperviseur. Nous avons proposé que toute les démarches réalisées par ce spécialiste soient automatiquement enregistrées dans des fichiers pour être ensuite analysées afin d'isoler des cheminements privilégiés.
Une extension de cette technique permet de construire un véritable mécanisme d'apprentissage et d'enseignement en rendant disponibles pour tous, sous la forme de macro-commandes, les méthodes de travail de ceux qui ont la meilleure connaissance du système.

Ce procédé apporte de nombreux avantages; en particulier, c'est un moyen original utilisable dans le cadre de la formation de spécialistes en analysant avec eux les méthodes de travail préalablement enregistrées de ceux qui ont une plus grande expérience.

Cette méthode d'analyse globale n'étant opérationnelle que depuis peu de temps, les dernières suggestions n'ont pas encore été appliquées et par là-même, elles n'ont pu être validées.

Dans ce second chapitre, nous avons facilité les mesures de performances et l'analyse du comportement de l'hyperviseur en introduisant des mécanismes qui permettaient des transferts unidirectionnels entre l'hyperviseur et une machine virtuelle spécialisée.
Leur création a entraîné la généralisation d'une instruction élémentaire particulière ainsi que l'emploi de certaines souplesses de fonctionnement dues à l'existence de la pagination.
Rappelons qu'il s'agissait principalement de transférer tout ou partie de la mémoire réelle de l'hyperviseur à une machine virtuelle en différant aussi longtemps que possible le moment de la recopie, quelquefois même en la supprimant si la page concernée n'a pas été modifiée par la machine qui l'acquiert.
On ne peut que recommander l'emploi systématique de tels mécanismes fort peu coûteux.

Coopération explicite
entre
hyperviseur et machines virtuelles généralisées

3.1 Présentation

Dans ce chapitre, nous allons examiner la possibilité de généraliser l'architecture de machines virtuelles en introduisant de nouveaux concepts qui ont pour but de simplifier la structure des systèmes tout en étant moins coûteux et plus performants que ceux actuellement utilisés pour exécuter des fonctions équivalentes.

Nous avons vu jusqu'à présent certains avantages de l'hyperviseur CP67 qui se révèle fort intéressant à plus d'un titre. Il est tout d'abord la première réalisation opérationnelle connue d'un générateur de machines virtuelles. Ensuite, ses objectifs ayant été dès l'origine parfaitement définis, ses structures internes sont particulièrement claires. Nous n'avons pas l'intention de nous livrer ici à une étude exhaustive de son fonctionnement; il suffit pour cela de se reporter à sa description (ref 22, 23, 31).
Cette rigueur dans la définition permet d'acquérir une connaissance approfondie du système en quelques mois, au point même de pouvoir y apporter des modifications non triviales.

Cette dimension 'humaine' présente un intérêt indiscutable par rapport à d'autres systèmes destinés à gérer autant de ressources et il faut la respecter quand on entreprend des modifications importantes au sein même de l'hyperviseur. En effet, l'expérience nous permet de dire qu' il vaut mieux reprendre les fonctions existantes en cherchant à les épurer et à les généraliser plutôt que d'en ajouter de nouvelles. Par ce procédé, on doit pouvoir finalement obtenir un programme suffisamment stable pour être inséré directement dans le hardware, sous forme d'un micro-programme par exemple.

C'est dans cette optique que nous avons cherché à généraliser CP67. On peut considérer ce dernier uniquement sous l'aspect d'un générateur de pures machines virtuelles comme l'a étudié Robert GOLDBERG dans sa thèse (ref 19).

Toutefois, dans un esprit d'extension et de prospection, nous pensons qu'un hyperviseur doit pouvoir générer des machines virtuelles conformes à une architecture prédéfinie et aussi des entités qui n'ont pas besoin d'être la réplique de machines réelles existantes. Elles seraient gérées par des systèmes virtuels et nous les appellerions 'machines virtuelles généralisées'. Elles seraient obtenues par addition de

mécanismes ne pouvant être définis que grâce à la présence constante de l'hyperviseur.
Dans ce domaine, nous avons étudié et proposé avec Jean-Pierre LE HEIGET (ref 25, 27, 37), un projet appelé 'CP+/CMS+'; son but est de faire travailler de façon complémentaire un système virtuel dérivé de CMS et un hyperviseur du même type que CP67 enrichi de certaines fonctions élémentaires généralisées.

Une partie seulement de ce que nous allons décrire a été réalisée. Ce chapitre est donc plus une introduction à un travail de recherche en cours que le bilan d'une opération terminée; ceci signifie que les propositions qui y sont faites sont susceptibles de subir certaines modifications majeures avant de donner naissance à un prototype opérationnel.

Un système virtuel tel que CMS+ est exclusivement conçu pour assurer le dialogue avec l'utilisateur, tandis que la gestion des ressources physiques est à la charge de l'hyperviseur. Au niveau de CMS+, on ne considère que des ressources purement logiques et, en particulier, nous cherchons à éliminer autant que possible les notions d'entrée-sortie au sens classique.

Du point de vue de l'utilisateur, un tel système virtuel doit avant tout, offrir une gestion de fichiers évoluée. C'est sur cet aspect du problème que nous avons commencé à travailler.

3.2 Création et mise en oeuvre de disques logiques par l'hyperviseur

Un système actif sur une machine virtuelle telle que CMS, utilise actuellement des mini-disques comme support de gestion de fichiers.
Par définition, un mini-disque est une suite de cylindres contigus apparaissant sur un seul disque physique. La taille maximale de cette portion de disque réel est celle d'un disque complet.
Les notions classiques de disques et celles de mini-disques s'identifient: la seule différence éventuelle concerne la taille. Aussi, la simulation doit-elle être suffisamment subtile pour qu'un système conçu pour fonctionner sur une machine réelle puisse utiliser des mini-disques, alors qu'il croît disposer de disques complets, sans être perturbé. Chaque mini-disque est désigné par un nom qui se présente, par

analogie avec les disques réels, sous la forme d'une adresse hexadécinale virtuelle.
La simulation est effectuée par l'hyperviseur et nous pouvons résumer les fonctions qu'il doit remplir.

-Repérer l'adresse réelle de l'unité physique sur laquelle le mini-disque a été défini; ceci lui permet d'assurer la conversion de l'adresse virtuelle en adresse réelle réelle juste avant l'exécution d'une opération d'entrée-sortie;

-Contrôler la position du bras du mini-disque; en effet, il faut appliquer un facteur de translation à toute opération de déplacement (définie par une commande 'SEEK'). Ce facteur est égal au numéro du premier cylindre réel baptisé 'cylindre 0' du mini-disque.

Dans la technologie des machines 360, une entrée-sortie sur disque se traduit par l'exécution d'une suite d'instructions destinée à un canal; la première est une instruction 'SEEK' de mise en position du bras du disque; les suivantes ont pour but de lire ou d'écrire un enregistrement. Il est alors possible de minimiser le temps d'occupation du(ou des) canal(aux) en isolant la première instruction pour la faire

exécuter seule; quand l'opération physique correspondante est achevée, la suite des autres instructions est alors exécutée. Cette technique est connue sous le nom de 'SPLIT-SEEK'.

-Simuler les conditions 'fin de disque' toutes les fois qu'une demande de déplacement du bras tend à l'amener en dehors des limites du mini-disque. La condition réfléchie à la machine virtuelle qui émet une telle opération invalide est identique à celle qui est obtenue sur un disque réel quand un programme-canal cherche à placer le bras sur un cylindre inexistant.

-Structurer le disque. Cette opération consiste à écrire un certain nombre d'enregistrements particuliers et obligatoires. L'un d'entre eux, appelé 'home address', identifie chaque piste en y écrivant son adresse relative par rapport au début du disque. Cette information est utilisée ultérieurement par l'unité de contrôle afin de s'assurer que le déplacement du bras est convenablement effectué. Cette vérification est réalisée de façon totalement automatique sans que l'unité de traitement puisse s'en rendre compte.

En cas d'anomalie, des procédures micro-programmées s'exécutent au niveau de l'unité de contrôle. Elles consistent à appliquer des algorithmes de corrections d'erreurs.

L'hyperviseur doit donc tenir compte de ces contraintes technologiques et en conséquence, translater les données fournies en argument d'une commande d'écriture d'un enregistrement du type 'home-address'; s'il ne respecte pas ces conventions, pour tout mini-disque dont l'origine est différente de celle d'un disque physique, l'opération de contrôle que nous venons de décrire conduirait à une erreur non corrigible.

Les divers aspects de la simulation imposent donc à l'hyperviseur d'explorer le programme-canal destiné à un mini-disque afin de repérer les commandes de mise en place du bras et d'écriture des enregistrements correspondant aux 'home-addresses'.

Il existe d'autres raisons qui imposent à l'hyperviseur l'analyse et éventuellement la modification du programme-canal; toutefois, elles ne sont pas propres aux disques; elles sont dues à l'emploi de mémoires virtuelles.

Une opération d'entrée-sortie fait toujours intervenir d'une part la mémoire réelle, d'autre part une unité périphérique réelle. Or, les mémoires virtuelles résident sur différents composants de la hiérarchie mémoire et non exclusivement en mémoire réelle.

Une opération de transfert entre la mémoire virtuelle et une unité virtuelle, doit donc être décomposée en trois étapes; en effet, il faut:

-amener en mémoire réelle les pages de la mémoire virtuelle impliquées dans le transfert à l'aide d'opérations de pagination;

-analyser les adresses virtuelles transmises par les instructions du programme-canal virtuel afin de créer le programme-canal réel;

-réaliser le transfert entre la mémoire réelle et l'unité réelle correspondant à l'unité virtuelle ou inversement.

Pour le premier point, l'hyperviseur doit tout d'abord reconnaître les numéros de pages virtuelles puis, obtenir ces pages en mémoire réelle et les verrouiller avant que l'opération d'entrée-sortie ne soit initialisée.

Pour le second point, si les adresses virtuelles doivent être modifiées en leur équivalent réel, il faut

aussi s'assurer de la longueur de l'information à transmettre. En effet, à cause de la pagination, certaines zones de données qui sont contiguës en mémoire virtuelle, peuvent ne plus l'être en mémoire réelle. Cette discontinuité conduit l'hyperviseur à créer un programme-canal réel plus conséquent que le programme-canal virtuel.

Prenons un exemple:

Soit une zone de données occupant 10K dans l'espace virtuel sous forme d'un seul bloc telle qu'il y ait 2K dans une première page, 4K dans celle qui lui est adjacente et 4K dans la suivante. Une opération d'entrée-sortie portant sur cette zone est réalisée à l'aide d'une seule commande. L'installation de ces données en mémoire réelle impose le verrouillage des trois pages précitées et probablement la création de trois commandes au lieu d'une seule. En effet, il n'y a aucune raison pour que les pages concernées soient voisines les unes des autres en mémoire réelle.

Le mécanisme de simulation d'une opération d'entrée-sortie implique une véritable compilation des programmes-canaux. Cette opération devient très complexe dès qu'on souhaite traduire correctement les commandes de branchement à l'intérieur du programme-canal (appelées 'TIC' en terminologie 360).

Elle le devient encore plus dès que l'on s'intéresse aux programmes-canaux qui s'auto-modifient pendant l'opération d'entrée-sortie.

La technique que l'on vient de présenter est appliquée par CP67 à tout programme-canal, quel que soit le type de l'unité destinataire, qui lui est soumis; en particulier, elle est employée avec ceux qui sont destinés aux mini-disques.
L'avantage principal de la notion de mini-disque est de permettre une certaine économie d'unités réelles. On peut en effet proposer à l'ensemble des machines virtuelles d'utiliser, à un instant donné, plus de disques (mini-disques en l'occurrence) que ne le permet la configuration réelle.

Nous nous sommes intéressés à cet aspect particulier d'unités virtuelles pour tenter de généraliser l'idée de mini-disque en s'éloignant des caractéristiques du support physique sur lequel ils sont générés. Nous avons souhaité également éliminer la compilation des programmes-canaux destinés à ces unités car, nous l'avons vu, c'est une opération complexe et coûteuse.

Par définition, on appelle donc disque logique de longueur N, une suite de N pages contiguës ou non qui résident sur une ou plusieurs unités physiques à accès direct.
L'espace ainsi défini est adressé de 0 à N-1 et chaque adresse repère une page de 4K; cette adresse a été choisie pour occuper 31 bits de telle sorte que la taille maximale d'un disque logique soit par conséquent d'environ 8.000 milliards de caractères, ce que l'on considère dans le contexte où l'on se place, comme un 'infiniment grand'.
Si malgré tout ce majorant se révélait insuffisant, rien n'empêcherait de définir l'adresse sur un nombre de bits plus important pour reculer ainsi très largement ces limites théoriques.

Au niveau de l'utilisateur, seules les adresses virtuelles, c'est-à-dire la numérotation de pages allant de 0 à N-1, sont connues. La conversion des adresses virtuelles en réelles est laissée à la responsabilité d'une fonction de CP+.

Soit alors u(1), u(2)... u(p) un ensemble de p unités physiques à accès direct (tambours et/ou disques)

utilisables pour générer en particulier des disques logiques.

Un espace comportant N pages est composé d'une suite de t extensions: e(1), e(2)... e(t), si l'on appelle extension le plus grand ensemble de pages physiquement contiguës et appartenant au même disque logique.

Soit l(e(i)) le nombre de pages contenues dans l'extension e(i).

Notons $F(t)=\sum_{i=1,t} l(e(i))$; on a la relation F(t)=N.

Appelons:

-u(e(i)), l'unité physique sur laquelle apparaît l'extension e(i);

-dep(e(i)), le déplacement (en terme de numéro de bloc physique) du premier bloc de e(i) par rapport au début de u(e(i)).

Le disque logique est alors défini par la suite de triplets:

(u(e(1)),dep(e(1)),l(e(1))),

...

(u(e(i)),dep(e(i)),l(e(i))),

...

(u(e(t)),dep(e(t)),l(e(t))).

numéro de l'entrée	numéro de la 1ère page de chaque extension	identification de l'unité physique	déplacement (position de la 1ère page)
1	0	$u(e(1))$	$dep(e(1))$
2	$p(e(1))$	$u(e(2))$	$dep(e(2))$
⋮			
i	$\sum_{j=1}^{i-1} p(e(j))$	$u(e(i))$	$dep(e(i))$
⋮			
t	$\sum_{j=1}^{t-1} p(e(j))$	$u(e(t))$	$dep(e(t))$
$t+1$	$\sum_{j=1}^{t} p(e(j))$	*	*

Figure 3 : Contenu de la table de conversion des adresses d'un disque logique

A partir de cette liste, on peut construire une table de conversion T des adresses virtuelles du disque logique en adresses réelles (figure 3). Sur cette figure, le symbole '*' signale une entrée invalide.

Soit A, une adresse virtuelle à convertir;

-On cherche dans T l'entrée i telle que:

$F(i-1) \leqslant A < F(i)$

-On en déduit u(e(i)) qui identifie l'unité physique de résidence et R(A), le numéro de la page cherchée, donnée par la relation suivante:

$R(A)=A+dep(e(i))-F(i-1)$

A partir du couple (u(e(i)),R(A)),il est possible de construire un programme-canal permettant d'accéder au bloc qui contient l'adresse désirée.

Pour se protéger des valeurs erronnées de A, on introduit une (t+1)ème entrée dans la table T. Elle définit le premier numéro de bloc invalide.
D'autre part, les discontinuités dans l'adressage virtuel des disques logiques sont reconnues dans la table T par une ou plusieurs entrées dont les couples (u(e(i)), dep (e(i))) sont invalides.

Pour permettre une conversion d'adresses, rapide et efficace, la table T doit contenir un nombre d'entrées aussi faible que possible; minimiser ce dernier est donc une contrainte imposée à cet algorithme de transformation.

Lorsque le disque logique se réduit à une seule extension, on retrouve approximativement la notion de mini-disque. La différence fondamentale vient du fait que les caractéristiques physiques liées à l'unité ont disparu (piste, cylindres, etc...).

Au niveau du système virtuel, on se contente de demander la nième page du disque logique; CP+ se charge d'assurer la conversion d'adresses grâce à la table T ainsi que le transfert de la page. La phase de compilation des programmes-canaux a complètement disparu.

3.3 Avantages offerts par les disques logiques

La notion de disque logique ainsi introduite présente un certain nombre d'avantages que nous allons examiner:

-L'hyperviseur répartit l'ensemble de l'espace disponible sur les unités physiques à accès direct

entre les différentes machines virtuelles pures ou généralisées. De plus, la place réservée à l'ensemble des utilisateurs de systèmes virtuels généralisés (CMS+, par exemple), doit être également divisée en sous-espaces disjoints. En effet, certains groupes d'utilisateurs souhaitent isoler la partie d'espace dont ils ont besoin pour des raisons de sécurité. Cette notion fondamentale justifie la présence de disques logiques multiples sous contrôle d'un même hyperviseur CP+. Leur nombre est modulé en fonction des impératifs de chaque installation. Ainsi, on peut avoir un seul disque logique couvrant alors tout l'espace disponible ou plusieurs disques logiques par utilisateur (on retrouve ici la possibilité offerte par CMS où chacun peut disposer d'un maximum de 10 mini-disques accessibles simultanément).
Nous pensons que cette technique des espaces virtuels multiples constitue actuellement l'un des mécanismes les plus efficaces de protection entre différents groupes d'utilisateurs.

-La notion de disque logique a fait disparaître toute subordination vis-à-vis des caractéristiques des unités physiques. Les nombres de cylindres ou de pistes sont en effet dépendants d'un matériel particulier et

susceptibles d'être modifiés. En fait, au niveau de CMS+, on souhaite que toutes ces notions soient transparentes; l'adjonction d'un nouveau type d'unité doit seulement impliquer l'addition d'un sous-programme dans l'hyperviseur; il aura pour rôle de calculer une adresse physique en fonction des paramètres propres à l'unité et du couple (u(e(i),R(Λ)) évalué précédemment. C'est d'ailleurs pour cette raison que dans CP+, nous obtenons l'emplacement effectif d'un bloc par un double adressage indirect qui consiste à:

-calculer à partir de l'adresse virtuelle une identification d'unité et un numéro de bloc relatif à l'origine de celle-ci;

-transformer le couple obtenu en programme-canal.

Rappelons que le disque logique dont la capacité est quasi-illimitée (ce qui nous a conduit à utiliser une adresse de 31 bits) doit englober des unités physiques multiples de types divers.

-La simplicité de l'adressage est également un facteur déterminant: l'ensemble des adresses qui servent à nommer les pages du disque logique est la suite des entiers. Calculer des adresses avec ces nombres est une opération très élémentaire qui n'a rien de comparable avec l'adressage classique basé sur la concaténation de

numéros de cylindre (C), de piste (P) et d'enregistrement (E). En particulier, il s'agit d'un adressage continu c'est-à-dire que la page qui suit la pième a pour adresse (p+1), alors que dans l'autre cas, l'adresse (C,P,E+1) vient rarement après (C,P,E).

-Pour simplifier les demandes de lecture ou d'écriture d'une page, émises par un système virtuel, nous utilisons une fonction primitive du type 'diagnose' (de façon similaire à ce qui a déjà été décrit précédemment dans le chapitre 2 (2.2.1)). Ceci permet de supprimer la construction du programme-canal au niveau du système virtuel correspondant à l'opération désirée.

-Pour optimiser les échanges d'information entre un disque logique et la mémoire virtuelle, nous voulons faire appel à des mécanismes homogènes. C'est ce qui nous a conduits à utiliser la page comme unité de transfert et, en conséquence, à employer toutes les fonctions de pagination de l'hyperviseur optimisées depuis suffisamment longtemps pour être devenues efficaces. L'insertion d'une page d'un disque logique dans une mémoire virtuelle est réalisée selon la technique utilisée dans l'échange d'une page, ne résidant pas en mémoire réelle, entre l'hyperviseur et

une machine virtuelle. Rappelons que l'opération d'entrée-sortie est différée jusqu'à ce qu'une référence soit réellement faite à cette page.

-En ce qui concerne les privilèges d'accès, le disque logique peut être globalement accessible par un ou plusieurs utilisateurs. Il est également possible de le déclarer disponible soit pour la lecture seulement soit pour la lecture et l'écriture; certaines zones du disque logique peuvent être affectées de privilèges particuliers plus restrictifs: ainsi par exemple, un fichier peut être accessible en lecture seulement alors qu'il réside sur un disque logique globalement déclaré en lecture et écriture.
Une description détaillée de la réalisation et de la mise en oeuvre de ces disques logiques est donnée dans la référence 35.

3.4 Emploi des disques logiques par le système virtuel CMS+.

Par définition et par construction, tout système virtuel désirant utiliser le mécanisme que nous venons de décrire, considère le disque logique comme une suite

de pages. Il dispose de fonctions primitives qui lui permettent d'insérer une ou plusieurs pages en mémoire virtuelle ou réciproquement d'en recopier sur le disque logique.

Dans ce vaste réservoir de pages, CMS+, qui est un système virtuel particulier, va ranger les fichiers de l'utilisateur. Ceux-ci se présentent également sous la forme d'une suite de pages et, pour y rechercher une information, il suffit de donner un numéro de page relatif au début du disque logique et un déplacement à l'intérieur de cette page.
Mais dès que les fonctions deviennent plus complexes, une telle définition d'adresse se révèle insuffisante. En effet, il n'est guère possible de vérifier le droit d'accès du programme à la page ainsi désignée; on retrouve le problème de la protection.
Il faut donc ajouter un niveau d'adressage indirect supplémentaire qui permette ce contrôle.

Il semble alors raisonnable de doter le fichier d'une structure voisine de celle d'une mémoire virtuelle: tout élément de ce fichier devient directement accessible par une adresse de 31 bits et la taille maximale théorique peut ainsi atteindre 2 milliards de

caractères. L'espace ainsi défini ne contient que l'information connue par l'utilisateur; aucun autre élément descriptif ne doit y être ajouté. Si ces derniers apparaissent ultérieurement nécessaires pour rechercher une unité d'information par contexte par exemple, on utilise alors des descripteurs externes aux fichiers. De cette façon, l'utilisateur peut adresser directement le contenu de son fichier sans avoir à transiter par des méthodes d'accès imposées de l'extérieur comme dans de nombreux systèmes.

L'allocation de pages du disque logique à un fichier est réalisée de façon totalement automatique par CMS+. La réservation effective d'une page n'a lieu qu'au moment où elle existe réellement, c'est-à-dire lorsqu'une demande d'écriture est explicitement faîte. Ceci veut dire qu'un fichier dans lequel apparaissent des caractères aux seules adresses 1 et 100.000, occupent deux pages du disque logique.
Réciproquement, lorsqu'un élément du fichier est lu alors qu'il n'a pas été préalablement créé, nous avons choisi de restituer une valeur nulle.

La mise en place de ce principe d'adressage nous permet de faire deux remarques:

-le mécanisme donne de façon triviale l'accès direct à un enregistrement logique de longueur fixe; en effet, le calcul de l'adresse de son premier caractère s'obtient par simple multiplication du numéro d'enregistrement par une longueur puis, l'accès a lieu par référence directe à l'adresse obtenue.

-l'emploi des méthodes du type 'adressage dispersé' ('hashing') pour ranger l'information dans le fichier est grandement facilité.

L'établissement des espaces-fichiers au sein d'un disque logique impose la création d'une partie descriptive qui contient toute l'information nécessaire pour retrouver la trace d'un fichier et permettre la gestion de la zone non encore utilisée du disque logique.

De plus, la localisation d'un élément dans un fichier impose de retrouver les tables qui décrivent ce dernier. Pour cela, nous avons prévu une procédure de gestion de catalogue à qui on transmet divers paramètres tel que le nom du fichier, la façon dont on veut y accéder et l'identification de l'utilisateur.

Cette opération préliminaire que l'on peut appeler 'ouverture du fichier' est exécutée lors de la première

lecture ou écriture. Elle s'accompagne de la création de tables dites 'étendues' dont la structure rappelle celle des tables de segments et de pages.

3.5 Nécessité et mise en oeuvre de la mémoire à trous

Quand un utilisateur est le premier candidat à désirer l'accès à un disque logique, la partie descriptive de celui-ci est insérée dans sa mémoire virtuelle sous la forme d'un segment. Elle est ainsi directement adressable par le système virtuel et peut être éventuellement partagée avec d'autres utilisateurs lorsqu'ils voudront eux-mêmes accéder à ce même disque logique.

Nous avons vu dans le chapitre précédent que le partage d'une page entre deux mémoires virtuelles consistait à inscrire un repère identique dans les deux tables de pages. Ceci reste vrai tant que la page commune réside en mémoire réelle. Lorsqu'elle est transférée sur une unité périphérique externe, il faut invalider les entrées des tables de tous ceux qui partageaient cette page, ce qui oblige à effectuer une mise à jour constante de plusieurs chaînes de pointeurs.

Pour éviter ce travail improductif, il est beaucoup plus réaliste d'employer la segmentation qui facilite la mise en oeuvre de ce partage surtout lorsqu'il y a plusieurs pages à partager entre de nombreux utilisateurs. C'est ce que nous avons choisi pour CMS+. La table des pages ainsi que la table des pages externes qui lui est associée, deviennent alors uniques et seul le repère dans la table des segments de chaque machine intéressée doit être mis à jour.

A titre d'illustration, nous présentons les deux situations (voir figures 4 et 5). L'une correspond au partage d'une suite de pages et l'autre à celui d'un segment. La simple observation montre une très nette diminution du nombre de repères communs entre la première et la seconde.

Dans un tel environnement, les segments ainsi utilisés lorsqu'on emploie des disques logiques multiples, ne sont généralement pas occupés en totalité. La mémoire virtuelle prend alors l'allure caractéristique du 'gruyère', certaines zones de l'espace d'adressage n'étant pas utilisées.

La mise en oeuvre de telles 'mémoires à trous' facilite non seulement le partage des descripteurs de fichiers mais aussi celui des programmes qui ont la propriété de pouvoir être partagés.

Dans ce cas, le système est constitué de deux parties distinctes: l'une qui recueille la suite des instructions, l'autre qui représente les données. On peut ainsi isoler chacun de ces éléments dans des segments différents. Puisque les instructions ne peuvent être modifiées, le segment qui les contient peut donc être partagé tandis que le segment des données reste distinct pour chaque application. Cette technique est simple tant que les suites d'instructions ont des adresses identiques au sein des différents espaces virtuels. Si ce n'est pas le cas, le mécanisme s'alourdit considérablement: il impose la création de liaisons spécifiques du type PSECT de TSS par exemple, entre les différentes procédures qu'il faut aligner sur des frontières de pages; ce qui a pour effet d'accroître la fragmentation. Sans rejeter cette possibilité, nous pensons que la simplicité et le coût peu élevé du calcul d'adresses justifient la faible contrainte qui consiste à imposer à chaque système virtuel l'utilisation des mêmes adresses pour partager des suites d'instructions.

Enfin, la dernière propriété importante que nous citerons concerne la mise à jour des descripteurs des disques logiques. Celle-ci rend définitive les

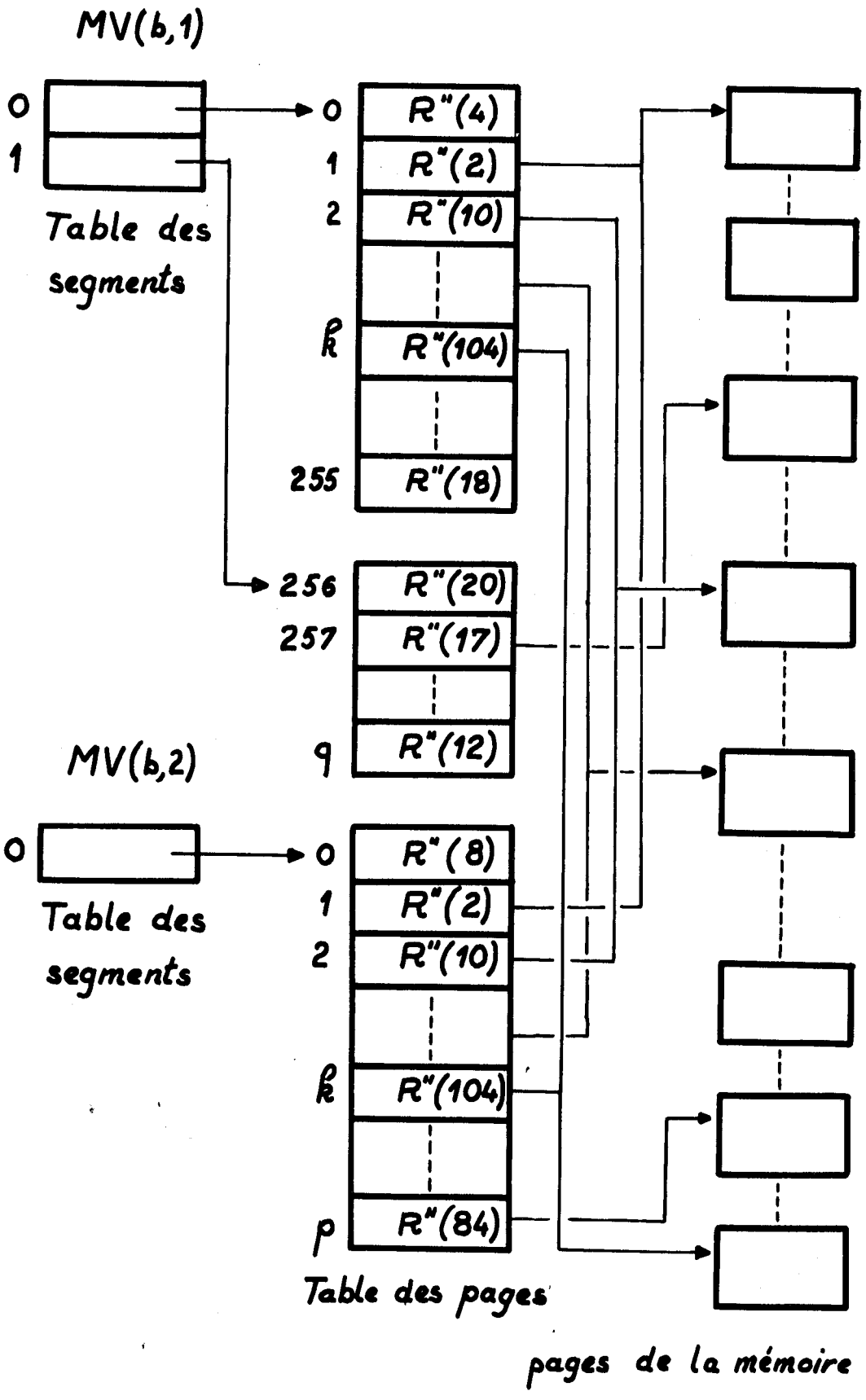

Figure 4: Partage avec emploi de la pagination

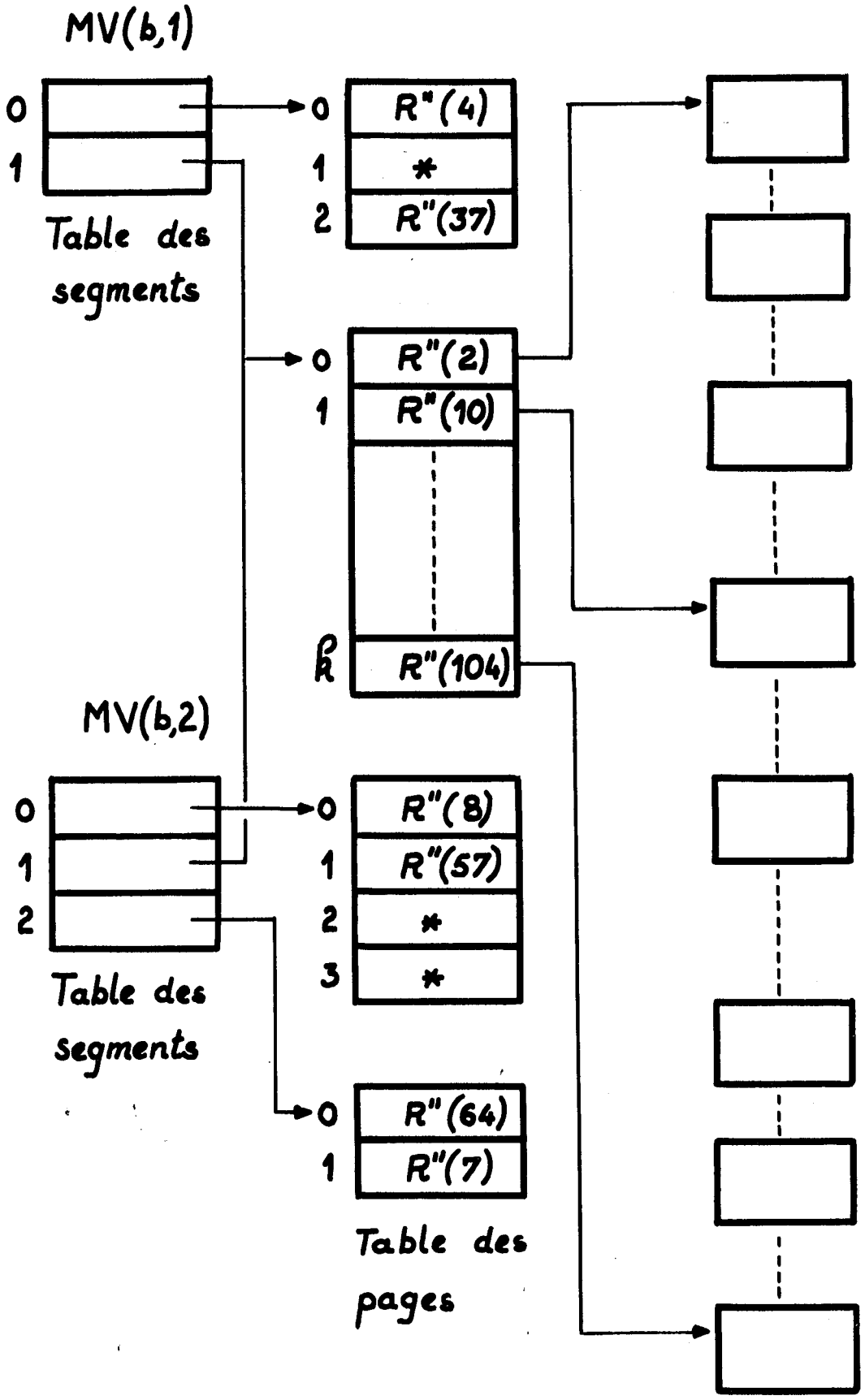

Figure 5 : Partage avec emploi de la segmentation

modifications apportées au disque logique et conduit normalement à la réécriture de tout le descripteur. Mais en fait, l'insertion du segment descripteur dans une mémoire virtuelle revient à transmettre la suite des pages externes où il réside à l'hyperviseur pour que celui-ci les inscrive dans la table des pages externes. Or, les pages modifiées se caractérisent à tout moment par un lieu de résidence différent du lieu initial; on peut donc ainsi les reconnaître et en exploitant cette caractéristique, réduire l'opération de recopie du descripteur.

Il est encore trop tôt pour juger les mécanismes introduits dans ce chapitre car bien que le sous-ensemble qui nous intéresse tout particulièrement soit opérationnel, son emploi s'est limité jusqu'à présent à des expériences fragmentaires.
Toutefois, les résultats obtenus par les S-dules qui sont un cas particulier des disques logiques (ref 35) permettent de penser que les performances peuvent être effectivement améliorées.

-Conclusion-

Dans les trois chapitres qui précèdent, nous avons essayé de présenter quelques mécanismes développés au cours de ces dernières années.

-Les premiers ont permis l'établissement d'une relation d'inclusion entre machines virtuelles tant au niveau des mémoires qu'à celui des unités de traitement et des unités périphériques.

-Les seconds ont autorisé l'examen de l'espace contenant l'hyperviseur depuis une machine virtuelle. Ceci a été réalisé soit par l'extension de l'architecture de base du calculateur considéré soit par l'utilisation intensive des mécanismes de pagination.

-Les troisièmes enfin, ont rendu possible la génération à la fois d'une architecture compatible avec une série de calculateurs existants (laissant ainsi disponibles de nombreux systèmes et programmes qui n'ont pas eu à subir de modifications pour être utilisés) et d'une autre architecture d'un niveau logique plus élevé.

Cette dernière est destinée à faciliter la création d'environnements utilisateurs et non pas à gérer les ressources physiques du calculateur. Pour ce type d'extensions, le concept d'hyperviseur est indéniablement l'outil adéquat qui permet de créer une architecture étendue et de la valider expérimentalement.

Ces divers mécanismes ont été effectivement utilisés dans la construction de prototypes qui avaient des objectifs précis et pratiques soit de mise au point, soit de mesures de performances ou d'examen du comportement de systèmes, soit de génération d'unités logiques nouvelles.
Ces réalisations ont fait l'objet de communications dans des séminaires internationaux; les références correspondantes ont été citées au cours du texte qui les décrit.

Tout ce qui a été présenté ici ne concerne que les mécanismes d'architecture de base et ne représente qu'un bilan partiel de notre activité; l'autre aspect

relatif aux langages de commande (que ce soit l'édition par contexte ou un macro-mécanisme d'extension), n'a été que partiellement évoqué (ref 2, 9 et 12). Il n'en constitue pas moins un intérêt majeur en ce qui nous concerne et nous semble être un point tout aussi fondamental dans le développement de l'utilisation des ordinateurs.

Pour ces raisons, nous pensons poursuivre le travail entrepris dans ces deux domaines (en particulier, en explorant les suggestions que nous avons faîtes dans les chapitres 1, 2 et 3), dans l'espoir de trouver d'autres outils qui faciliteront l'écriture des systèmes et qui rendront les calculateurs plus accessibles à l'utilisateur.

-Bibliographie-

-1- R.J ADAIR, R.U BAYLES, L.W COMEAU, R.J CREASY
A virtual machine system for the 360/40.
IBM Cambridge Scientific Report G320-2007
Mai 1966

-2- Michel ADIBA et Claude HANS
Description externe de l'éditeur
du système CMS
Publication IMAG, Avril 1969

-3- Michel ADIBA
Editeurs par contexte
pour systèmes conversationnels
Thèse de 3ème cycle, Grenoble 1971

-4- M.T ALEXANDER
Time-sharing supervisor programs
University of Michigan Computing Center
1969, revisé en mai 1970

-5- Alain AUROUX
Généralisation du concept d'adressage
Application au système CP67
Congrès AFCET 1972, Grenoble

-6- Alain AUROUX et Claude HANS
Les systèmes CP67 et CMS
Publication IMAG, Juillet 1968

-7- Alain AUROUX et Claude HANS
Le concept de machines virtuelles
AFIRO no.15, 1968

-8- Alain AUROUX et Claude HANS
Introduction aux systèmes CP67 et CMS
Colloque Université de Grenoble
Dunod, Novembre 1968

-9- Alain AUROUX et Claude HANS
Editeur de fichiers et macro-langage
d'un système conversationnel
Colloque international sur la Télé-informatique
Mars 1969

-10- Jacques BELLINO et Claude HANS
Virtual machine or virtual operating system ?
Workshop on virtual computer systems
ACM-SIGARCH-SIGOPS
Harvard University, 1973

-11- M. BERTHAUD, D. CLAUZEL et M. JACOLIN
Grenoble System Language
Language reference manual
FF2-0133

-12- Max CREVEUIL
EXEC, un macro-processeur pour CMS
Rapport DEA, Grenoble 1973

-13- P.DENNING
Ressource allocation
in multiprocess computer systems
MIT project MAC
Thèse de PhD, 1968

-14- P.DENNING
Virtual memory
Computing surveys, Septembre 1970

-15- J.P DUPUY et Juan RODRIGUEZ
The evaluation of a time-sharing page demand system
SJCC 1972

-16- J.P.DUPUY et Juan RODRIGUEZ
The design, implementation and evaluation of a working-set dispatcher
Communications of the ACM, Avril 1973

-17- J.P DUPUY
Thèse CNAM (à paraître, 1974)

-18- C. GATEAU et R. PERRONNEAU
Réalisation d'un simulateur de machine IBM 360 type 67
Rapport DEA, Grenoble 1972

-19- Robert P. GOLDBERG
Architectural principles for virtual computer systems
Thèse de PhD, Harvard University, Novembre 1972

-20- R.E GRISWOLD, J.F POAGE, I.P POLONSKY
The Snobol4 programming language
Prentice-Hall, 1971 (seconde édition)

-21- Claude HANS
Dispositifs câblés propres au 360/67 simplex
Note technique IMAG, Décembre 1969

-22- Claude HANS
Description d'une machine virtuelle
Note technique IMAG, Janvier 1970

-23- Claude HANS
Description des canaux réels
Note technique IMAG, Février 1970

-24- Claude HANS
Introduction à l'utilisation de mémoires
virtuelles
Convegno sul time-sharing
CNUCE Pise, Octobre 1971

-25- Claude HANS
Generalised virtual spaces under CP
Symposium IBM Hursley, Mars 1972

-26- Claude HANS, Patrick LEFEBVRE et Bernard PETEUL
Mise au point conversationnelle de systèmes grâce à un couplage particulier de machines virtuelles
Association canadienne d'informatique
Montréal 1972

-27- Claude HANS et J.P LE HEIGET
Généralisation de la notion d'espace virtuel
Congrès AFCET, Grenoble 1972

-28- Danielle HANS
Thèse CNAM (à paraître, 1974)

-29- IBM System/360 Principles of operation
A22-6821

-30- IBM System/360 model 67
Functional characteristics
A27-6719

-31- IBM CP Program logic manual
GY20-0590

-32- IBM CMS User's guide
GH20-0859

-33- IBM TSS/360
Time-sharing compendum
Y20-0450

-34- IBM TSS/360
TSS Resident Supervisor
GY28-2012

-35- Xavier de LAMBERTERIE
Espaces virtuels et gestion de fichiers
Thèse de 3ème cycle, Grenoble 1973

-36- Patrick LEFEBVRE
SPY, un système de contrôle
Thèse de 3ème cycle, Grenoble 1973

-37- J.P LE HEIGET
Généralisation de la notion d'espaces virtuels
sous les systèmes CP67-CMS
Thèse CNAM, Grenoble 1972

-38- Jacques LEROUDIER
Une analyse de systèmes
Thèse de 3ème cycle, Grenoble 1973

-39- R.A MEYER et L.H SEAWRIGHT
A virtual machine time-sharing system
IBM systems journal vol 9 no.3 (1970)

-40- NGUYEN-THANH-THI
XCP, un environnement graphique conversationnel
pour l'examen des structures d'un système
Application à CP67
Thèse de 3ème cycle, Grenoble 1973

-41- E.I ORGANICK
The MULTICS system :
an examination of its structure
MIT Press, 1972

-42- Bernard Peteul
La mise au point de programmes par simulation
Thèse de 3ème cycle, Grenoble 1971

-43- Juan RODRIGUEZ
Experimental data on how program behaviour affects the choice of scheduler parameters
ACM third symposium on operating system principles
Stanford, 1971

-44- Juan RODRIGUEZ
Empirical working-set behaviour
Communications of the ACM, Septembre 1973

-45- Henri SAVARY
Outils de mise au point pour langages de haut niveau :
association de modules et contrôle de l'exécution
Thèse de 3ème cycle, Grenoble 1973

Dernière page d'une thèse

VU

Grenoble, le

Le Président de la thèse

VU, et permis d'imprimer,

Grenoble, le

Le Président de l'Université
Scientifique et Médicale