Le système MVS

<u>Avertissement</u> : ce guide était destiné à l'origine à la formation d'ingénieurs-systèmes.

Nous recommandons aux débutants de commencer par étudier les chapitres 1, 5, 14, puis 4.

Ce livre n'a pas été remis à jour depuis la seconde édition (1993). Certaines parties peuvent donc être obsolètes aujourd'hui, tandis que d'autres demanderaient à être davantage étoffées (OS/390 et z/OS, le CMOS, Sysplex, WLM, les réseaux avec TCP/IP et Internet).

Le système MVS

Sommaire

1. Les architectures IBM 370 et 390

A) Notions générales sur les séries 360, 370 et 390

1) La série 360.

L'architecture d'un ordinateur est l'ensemble des règles logiques qui ont présidé à son élaboration, et qui pourvoient la machine d'un certain nombre de fonctions bien définies. La notion d'architecture d'ordinateur permet de distinguer le point de vue de celui qui construit la machine du point de vue de celui qui l'utilise, le programmeur. Pour ce dernier elle se concrétise dans un éventail d'instructions de base (langage machine) qu'il peut directement utiliser pour mener à bien les tâches qui lui sont confiées. Chez IBM ce jeu d'instructions est résumé dans la brochure "Principles of Operations".

L'architecture d'une machine est en fait commune à une série d'ordinateurs faisant partie de la gamme du constructeur ; elle dépend de l'état de la technologie et des contraintes de coût sur les divers composants de la machine. Le système d'exploitation est cet ensemble de logiciels découlant directement de l'architecture dont la finalité est de rendre disponibles aux utilisateurs toutes les ressources matérielles.

Avant le lancement de la série 360, IBM et les autres fabricants commercialisaient un ensemble assez disparate de machines : le client se voyait proposer des modèles entièrement différents selon qu'il souhaitait effectuer des impressions sur papier, des calculs scientifiques ou gérer une entreprise.

La série IBM 360, série de troisième génération, apparut en 1964. Avec cette série le constructeur tentait d'unifier son offre avec une famille d'ordinateurs de toutes tailles, utilisant en principe un système d'exploitation unique (l'OS/360, mais le DOS apparut ensuite pour les petits modèles), et à vocation universelle, c'est-à-dire capables de traiter aussi bien des applications de gestion que des problèmes scientifiques.

Beaucoup de choix architecturaux marquants sont opérés à cette époque, qui seront conservés pour la plupart jusqu'à présent : l'octet à 8 bits, le code EBCDIC, l'interruption, la protection mémoire, un adressage à 24 bits sous la forme base-index, la micro-programmation, une indépendance plus grande entre le système de disques et le processeur (qui ouvre la porte à la concurrence des matériels "compatibles"), et surtout l'engagement d'IBM de "préserver l'investissement du client" en fournissant par la suite des machines compatibles avec cette nouvelle architecture. Pari tenu, puisqu'aujourd'hui encore un programme écrit dans les années 60 sur une machine 360 peut fonctionner sur la plus récente des machines IBM ES/9000.

Les modèles de la gamme 360, commercialisés par IBM entre 1964 et 1971, sont les suivants : les modèles 22, 25, 30, 40, 44, 50, 65, 67, 75, 85, 91, 95, 195.

2) La série 370.

La série IBM 370, apparue en 1970, dans la continuité de la série IBM 360, constituait une autre famille d'ordinateurs à vocation universelle, auxquels on avait intégré une fonction capitale: la mémoire virtuelle. Les entrées-sorties étaient améliorées grâce aux canaux "block multiplexer", apparus pour la première fois sur l'IBM S/360 modèle 85, qui permettent un accès rapide aux données sur disque (un canal peut supporter plusieurs opérations disque simultanées). La série 370 a constamment bénéficié des progrès techniques des dernières décennies; elle fut et est encore aujourd'hui synonyme de grands ou très grands systèmes informatiques.

A titre d'information, voici les machines de la gamme IBM 370, regroupées en 6 familles différentes, chronologiquement:

- série 370: compatibles avec les 360, occupèrent le marché de 1970 à 1977 (modèles 115, 125, 135, 138, 145, 148, 155, 158, 165, 168, 195);
- série 303X (3031, 3032, 3033) deux fois plus puissants que les précédents, série de transition (1977-1979) basée sur une plus grande densité des circuits intégrés;
- série E ou série 43XX (4321, 4331, 4341, 4361, 4381) apparue en 1979 avec la technologie LSI; on parle à cette époque de quatrième génération d'ordinateurs, succédant aux première, seconde et troisième générations, marquées respectivement par les technologies des tubes à vide (vieilles séries IBM 701, 702, 650), des transistors (séries IBM 1401 et 70XX) et des circuits intégrés (série 360);
- série H ou série 308X (3083, 3081, 3084) apparue en 1981, de plus grande puissance de traitement, basée sur la technologie TTL et le TCM. Le TCM est un module à dissipation thermique contenant plusieurs dizaines de puces (ce qui représente des dizaines de milliers de circuits). Le TCM est rempli d'hélium gazeux (qui protège de la corrosion) et réfrigéré par eau. Une machine telle que le 3081 comporte 36 TCMs. Avec cette série l'architecture 370 subit une évolution avec l'apparition de l'architecture 370 étendue (XA);
- série "Sierra" ou série 3090, de technologie semblable à celle des 308X (emploi de TCMs améliorés), mais plus puissante (la technologie de base est l'ECL - emitter coupled logic), sortie en 1985. Depuis 1988 les modèles 3090-E -S et -J (et certains 43XX) supportent l'architecture ESA (Enterprise Systems Architecture) qui est une évolution supplémentaire par rapport aux architectures 370 et 370-XA;
- série 9370, ordinateurs "départementaux", annoncés en 1986, apparus en 1987, orientés vers le traitement distribué en coopération avec de plus gros ordinateurs.

Seules les 4 dernières séries sont vivantes aujourd'hui, et toutes sont destinées à être remplacées par les nouvelles machines de type S/390.

Nous ne passerons pas en revue les nombreuses séries "compatibles" proposées par les concurrents d'lBM depuis le premier AMDAHL en 1975. Les compatibles offrent souvent des solutions intéressantes au moins par leur prix, sinon par leur originalité technique ; à ce titre ils constituent une alternative qui doit sérieusement être prise en compte. Le décideur veillera cependant à ne pas trop s'écarter du "standard du marché" (c'est-à-dire des solutions d'origine IBM) de façon à ne pas être prisonnier d'autres solutions "propriétaires" moins diffusées et d'avenir plus incertain, et à se ménager ainsi toute liberté de choix par la suite.

Dans la suite de ce chapitre nous regrouperons sous le terme d'architecture 370 les 3 architectures 370, 370/XA et 370/ESA.

3) La série 390.

En septembre 1990, IBM annonce sa nouvelle architecture de machines pour la décennie, l'architecture 390, concrétisée par de nouveaux ordinateurs: les ES/9000, destinés à remplacer aussi bien les 3090 (grâce aux modèles 9021 refroidis à eau), que les 43XX (par les modèles 9121 refroidis par air) et les 9370 (avec les modèles 9221 montés "en rack"). La principale innovation liée aux ES/9000 (appelés parfois pour les plus puissantes machines "Summit" ou "Future System", et Foothill pour les autres modèles à eau issus du 3090J) réside moins dans un nouveau packaging (avec des TCMs améliorés) que dans un système d'entrées-sorties par canaux à fibre optique, associés à de nouveaux périphériques "routeurs", permettant une transmission asynchrone en série. Les limitations de la transmission parallèle synchrone traditionnelle sur câbles en cuivre se trouvent ainsi dépassées : alors que les transferts de données nécessitaient une synchronisation entre canal, unité de contrôle et périphérique, les données sont maintenant organisées en "paquets" acheminés par les routeurs comme dans un réseau commuté.

Cette architecture ESCON (Enterprise System Connectivity) permet aux sites d'alléger la configuration matérielle (en remplaçant des tonnes ou dizaines de tonnes de câbles de cuivre par des câbles en fibre optique), d'accroître les distances de connexion largement au-delà de la limite des 400 pieds (122 m) entre périphériques et ordinateur, tout en améliorant les performances (avec des vitesses de transfert canal atteignant 17 MO/s).

Pour le reste, la règle de compatibilité ascendante est toujours respectée, et les bases de l'architecture 370 restent valables (le programmeur n'a pas conscience des changements apportés).

4) Les systèmes.

Pour mettre en oeuvre les architectures 370 et 390, IBM propose actuellement plusieurs systèmes d'exploitation bien différents par leurs origines et par leurs fonctions:

- **VSE** (Virtual Storage Extended): le plus ancien de tous, issu de DOS (Disk Operating System, système né avec la gamme 360), destiné aux machines "intermédiaires" que sont par exemple les 9370 ou les 4381.
- **MVS** (Multiple Virtual Storage): plus adapté aux grosses machines de la gamme (308X, 3090, ES/9000), apparu vers 1975 (plus tard sous la forme MVS-SP), lointain rejeton de l'OS (Operating System) de la gamme 360.
- **UNIX** : la version IBM "IX/370" du système Unix System V, date de 1985; elle a été remplacée depuis par AIX/370 (Advanced Interactive Executive), annoncé en 1989, livré en 1990, et par AIX/ESA, basé sur OSF/1 (noyau Mach), qui peut fonctionner sous PR/SM, sous VM ou en natif.
- **VM** (Virtual Machine): date des années 1970 (on peut situer son origine en 1967 avec CP/67 sur le 360-67), c'est un "hyperviseur" qui permet, grâce au concept de "machine virtuelle" de faire cohabiter sur une même machine réelle plusieurs systèmes incompatibles a priori, tels que les précédents (ou d'autres systèmes VM). VM est le seul système utilisable sur toutes les gammes 370 et 390.

MVS et VM sont actuellement les systèmes d'exploitation les plus "en pointe" et sur lesquels IBM fait le plus porter ses efforts. Pour les trois systèmes "historiques" MVS, VM et VSE, l'architecture ESA est l'architecture de convergence qui devrait remplacer toutes les versions précédentes.

Appartiennent aussi à l'univers 370 des systèmes moins connus tels que :

- ACP/TPF (Airline Control Program Transaction Processing Facility), système transactionnel utilisé à l'origine pour les réservations d'avion. Il date des années 60, à l'époque où "ACP" supportait le réseau de réservation SABRE d'American Airlines;
- DPPX/370 (Distributed Processing Programming Executive), un système distribué (issu de DPPX sur les machines IBM 8100) qui peut fonctionner sur des petites machines telles que les 9370, connectées à un site central MVS.

5) SAA.

S'il est vrai que les architectures 370 et 390 sont chez IBM celles des grands systèmes et des grands sites, on rencontre chez IBM d'autres architectures (PC, RT/PC, S/88, PS/2, S/1, S/36, S/38, 8100, AS/400, RS/6000) encore peu homogènes entre elles, car souvent développées par des laboratoires différents et répondant à des besoins différents. On peut dire que l'idée de "normalisation" qui avait inspiré le S/360 avait disparu pour céder la place à de multiples systèmes répondant à des problèmes spécifiques. Le projet d'architecture unifiée d'applications SAA (System architecture applications), annoncé en 1987, doit selon IBM remédier à l'hétérogénéité actuelle en fournissant une plate-forme standard qui devrait permettre aux principaux systèmes de communiquer plus aisément entre eux. Les quatre systèmes concernés sont MVS, VM, OS/400 et OS/2 (mais VSE et AIX participent à SAA dans une moindre mesure). Le but est tout autant d'augmenter la cohérence entre systèmes différents (à la différence de ce qui se passe dans le monde UNIX l'option du système unique a été écartée) que de faire enfin bénéficier l'utilisateur de la convivialité des "petits" systèmes personnels (une vraie révolution pour IBM que cette prise en compte - tardive - des potentialités des micros !). Le cadre conceptuel est fourni par :

- **CUA** (common user access), qui traite de l'ergonomie du poste de travail, avec une normalisation de l'interface utilisateur inspirée du monde micro-informatique et de l'orientation "objet" (tout utilisateur des systèmes traditionnels aura ressenti plus ou moins la nécessité de disposer de touches de fonctions standardisées et d'écrans où les données soient présentées de façon lisible et homogène...) ;

- **CPI** (common programming interface), qui apporte les mêmes langages et services sur toutes les plates-formes de façon à ce que les applications puissent être aisément portables de l'une à l'autre ;
- **CCS** (common communication support), s'appuyant sur SNA et les normes OSI, est le support de communication entre les différents systèmes IBM (LU 6.2, APPC), et les autres (TCP/IP, etc.).

Deux concepts essentiels viennent compléter le tableau SAA :

- AD/Cycle vise à industrialiser le développement applicatif en proposant un cadre d'accueil à différents produits de développement (outils de génie logiciel) et une base d'information de référence pour les données de l'entreprise (repository) ; IBM a passé des accords avec un certain nombre d'entreprises qui fourniront des produits conformes à AD/Cycle ;

- SystemView, qui doit faciliter la gestion des systèmes sur des plans aussi variés que la gestion des changements, des incidents, de la configuration, de l'exploitation, des performances et de l'inventaire des ressources ; SystemView fédèrera les outils fournis, avec Netview comme interface toute désignée. Comme pour AD/Cycle, SystemView aura son "repository" (sous SQL) et est ouvert aux produits conformes au cadre proposé.

B) Éléments des architectures IBM 370 et 390

L'ADRESSAGE: la mémoire de l'ordinateur contient les données ou les programmes dont le processeur a besoin. Rappelons brièvement les unités de codage de l'information:

* le bit (digit binaire) est l'unité élémentaire de stockage, le "point mémoire", qui contient la valeur 0 ou 1;

* l'octet, séquence de 8 bits, représente l'information la plus élémentaire qui ait une signification pour l'homme: le caractère (lettre, chiffre ou autre symbole); l'octet est parfois décomposé en 2 groupes de 4 bits (digits ou "nibbles") ; on peut ainsi représenter un octet sous la forme de 2 chiffres hexadécimaux (de valeur 0 à 9, A à F);

* le mot, qui a plusieurs acceptions en informatique:

- un groupement de bits que la machine, par construction, traite en seul bloc lors de transferts ou d'accès en mémoire: le mot technologique (on parle ainsi de machines à n bits, n étant le nombre de bits de ce mot). La taille du mot ("largeur" du chemin de données) peut varier selon les modèles de machine (de 2 à 16 octets sur les IBM de la série 370);
- une unité d'information conventionnelle, que peut traiter la machine: le mot machine. Sur les IBM 370 (et sur d'autres machines) le mot comporte 32 bits, soit 4 octets. C'est dans cette dernière acception exclusivement que nous parlerons de "mot".

Les instructions doivent faire référence aux informations en mémoire de façon à pouvoir les lire ou les modifier: la convention utilisée pour ce faire est l'adressage. On numérote les octets de la mémoire à partir de zéro, le numéro de l'octet que l'on veut utiliser est son adresse. L'adresse d'une donnée est un nombre binaire de 3 ou 4 octets (pour être plus précis, de 24 ou 31 bits). Un mot mémoire a une adresse divisible par 4, un double-mot (8 octets) par 8, un demi-mot par 2. Il y a ainsi alignement de ces zones respectivement sur des frontières de mot, double-mot ou demi-mot; un tel "cadrage" existe pour des raisons de performance, même si l'adressage permet de descendre au niveau de l'octet.

Les instructions ne référencent pas directement la mémoire par des adresses absolues, mais par des adresses relatives de forme registre de base plus déplacement, ou base plus index plus déplacement (l'adresse est calculée en additionnant le contenu actuel des registres de base ou d'index et le déplacement).

LES REGISTRES:

Les registres sont des zones de stockage à accès rapide couramment utilisées par les ordinateurs (voir introduction). Dans l'architecture 370 chaque processeur dispose des registres suivants:

- 16 registres "généraux", de taille 4 octets (32 bits), pouvant contenir aussi bien l'adresse en mémoire d'une donnée, qu'un nombre ou une information quelconque (codée évidemment en binaire). Par convention ces registres sont numérotés de 0 à 15.
- 4 registres "flottants", de taille 8 octets (64 bits), contenant des nombres "flottants" (représentés par un nombre binaire associé à une puissance de 2, ces nombres sont utilisés dans les calculs scientifiques), portant les numéros 0, 2, 4, 6.
- 16 registres "de contrôle", de taille 4 octets et numérotés de 0 à 15, normalement inaccessibles au programmeur, utilisés par le système pour la gestion de la machine (des canaux à l'origine, puis notamment de la pagination). Ces registres n'existaient pas en architecture IBM 360.
- des registres utilisés pour des fonctions de "timing": la TOD clock (time of day clock) qui contient sur 64 bits un nombre binaire représentant la date et l'heure comptés à partir du 1er janvier 1900 à 0 heure (le 52e bit est incrémenté de 1 toutes les microsecondes, ce qui fournit une très grande précision tout en permettant de tenir une période d'à peu près 143 ans...), le clock comparator (qui permet d'interrompre le processeur à une heure donnée, par comparaison avec la valeur de la TOD clock), le CPU timer (qui permet d'interrompre le processeur après une période de temps donnée). Ces trois registres sont en quelque sorte l'horloge, le réveille-matin et le chronomètre de l'ordinateur. L'ordinateur dispose aussi d'une horloge matérielle d'alimentation indépendante, non accessible par le système d'exploitation, qui sert aux diagnostics et à donner l'heure au démarrage du système (voir aussi chapitre 2,G).
- le PSW et le registre de préfixage sont des registres particuliers dont nous verrons l'intérêt au chapitre suivant.
- l'architecture ESA utilise 16 nouveaux registres (access registers) qui, couplés avec les registres généraux, permettent d'accéder aux nouveaux espaces virtuels d'ESA.

C) Structure des données

Nous utiliserons la notation hexadécimale qui permet de représenter le contenu d'un octet par 2 "digits" hexadécimaux, entre 00 et FF (soit entre 0 et 255, valeur maximale que l'on peut coder sur les 8 bits de l'octet). Nous allons d'abord étudier les diverses arithmétiques qu'offre l'architecture 370.

1) ARITHMÉTIQUE DECIMALE PACKEE

L'arithmétique décimale porte sur des nombres en représentation "packée". Ces nombres sont codés en mettant 2 chiffres décimaux par octet, le dernier demi-octet symbolisant le signe (A, C, E ou F pour les nombres positifs, B ou D pour les négatifs). Par exemple les nombres 598F (2 octets) et 03271D (3 octets) représentent les valeurs +598 et -3271.

2) REPRÉSENTATION DÉCIMALE ÉTENDUE

L'arithmétique décimale packée est peu performante mais les nombres packés ont l'avantage de pouvoir être facilement convertis en nombres décimaux "étendus", seuls éditables sur les périphériques écran ou imprimante, mais impossibles à utiliser directement dans les calculs (un langage orienté gestion tel que Cobol donne cette illusion, mais en fait il passe d'abord par des conversions en nombres packés). Dans la représentation étendue (appelée aussi décimal zoné) chaque octet contient un chiffre décimal, le signe étant transféré dans l'avant-dernier demi-octet. Ainsi les 2 nombres précédents seront traduits par: F5F9F8 (3 octets) et F3F2F7D1 (4 octets). Les représentations décimales sont utilisées surtout dans les applications de gestion et dès qu'il est indispensable à un moment ou à un autre de "sortir" des états sur papier ou des résultats à l'écran.

3) ARITHMÉTIQUE ENTIERE

L'arithmétique entière s'applique aux nombres à représentation binaire (2, 4 ou 8 octets cadrés sur des frontières de demi-mot, mot et double-mot). Le signe est codé sur le premier bit du premier octet (bit à 0 pour positif, à 1 pour négatif, avec inversion des autres bits). Par exemple, avec des valeurs codées sur des demi-mots:

en base 2	**en hexadécimal**	**valeur décimale**
0000001010011100	**0256**	**+ 598**
1111001100111001	**F339**	**- 3 271**

la représentation binaire du nombre négatif - 3 271 est en fait obtenue ainsi:

0000110011000111 (soit 0CC7 en hexadécimal) représente + 3 271
1111001100111000 : on inverse tous les bits
1111001100111001 : on ajoute 0000000000000001

Il s'agit de la représentation dite "en complément à 2". On voit ainsi que les valeurs minimales et maximales qui pourront tenir sur le demi-mot sont les suivantes :

minimum: 1000000000000000 soit 8000 en hexadécimal et -32768 en décimal
maximum: 0111111111111111 soit 7FFF en hexadécimal et 32767 en décimal

De même un mot de 4 octets permettra de coder des valeurs entre -2147483648 et 2147483647.

4) ARITHMÉTIQUE FLOTTANTE

La représentation binaire, extrêmement commode et performante (avec l'emploi direct à cet effet des registres généraux), a cependant l'inconvénient de méconnaître les nombres à virgule et de représenter des valeurs binaires limitées, souvent suffisantes pour des applications de gestion mais certainement pas en informatique scientifique... L'arithmétique en virgule flottante adopte une codification consistant à représenter un nombre par:

- son signe (1er bit du 1er octet, comme en binaire);
- sa mantisse, inférieure à 1, contenant les chiffres significatifs, placés à partir du 2e octet; en représentation "normalisée" elle est cadrée à gauche pour dégager le plus de chiffres significatifs;
- sa caractéristique, qui représente la puissance de 16 par laquelle on doit multiplier la mantisse pour obtenir le nombre. Elle est codée sur les 7 derniers bits du 1er octet du nombre. La caractéristique pouvant être aussi bien positive (nombre plus grand que 1 ou plus petit que -1) ou négative (nombre compris entre -1 et 1), on convient que la chaîne de bits 1000000 (40 en hexadécimal) représente une caractéristique de 0 (puissance de 16 valant 0). On peut donc coder des nombres depuis un ordre de 16 puissance -64 (caractéristique représentée par 0000000) jusqu'à un ordre de 16 puissance 63 (caractéristique représentée par 1111111 soit 7F hexadécimal). La "vraie" puissance de 16 est obtenue en enlevant 40 (hexa) à la caractéristique.

Il existe 3 formats flottants en architecture 370: le format court (1 mot), le format long (2 mots) et le format étendu (4 mots, nouveauté par rapport à l'architecture 360). La valeur + 598 sera codée ainsi en flottant court:

43256000 (hexa): signe positif (bit à 0), caractéristique 43 - 40 = 3 (598 étant immédiatement inférieur à 16 puissance 3), mantisse 256. Cette représentation correspond simplement à l'égalité suivante (en base 16 !):

0,256 x (10 puissance 3) = 256.

5) LE CODE EBCDIC

Les données non numériques sont représentées par un codage sur les 8 bits de l'octet: le code EBCDIC (Extended Binary Coded Decimal Interchange Code), qui est une extension à 8 bits d'un code BCD à 6 bits. On peut ainsi représenter sur les IBM 370 jusqu'à 256 caractères différents, ce qui paraît offrir un alphabet très étendu par rapport à d'autres codes anciens à 6 bits ou à 7 bits (comme le code ASCII courant sur les micro-ordinateurs). Ce code permet de représenter les lettres majuscules, minuscules et les caractères "spéciaux" pour la ponctuation et autres symboles. Pour chaque pays un jeu de caractères particulier permet de combler les lacunes du jeu anglo-saxon (pour le français il faut des voyelles accentuées, la cédille, le tréma, etc.).

D) Les instructions

Les instructions 370 ont en général une longueur de 2, 4 ou 6 octets. Elles peuvent porter sur des données en mémoire ou sur des registres. L'arithmétique décimale permet de traiter des zones de longueur quelconque (inférieure à 256 octets). L'arithmétique fixe opère sur des demi-mots, des mots et des double-mots, tandis que l'arithmétique flottante permet de traiter des nombres sur 64 bits (128 bits en couplant les registres étendus).

L'ordinateur ne "comprend" que le langage machine, constitué des instructions de base de l'architecture, bien distinctes de celles des langages symboliques (dits "évolués"), COBOL, FORTRAN, PL/1, etc. qui se ramènent en fait, après le processus de "compilation", à un certain nombre de ces instructions de base seules exécutables par la machine, mais dont le programmeur en langage évolué n'a pas besoin de connaître la signification. Le langage assembleur est un langage particulier dont chaque instruction correspond à une instruction en langage machine; c'est le langage de prédilection des programmeurs systèmes. L'écriture de programmes en assembleur est simplifiée par l'emploi de "macro-instructions" paramétrables qui génèrent un certain nombre d'instructions assembleur pur.

Voici des exemples d'instructions en langage machine et en assembleur:

langage assembleur		langage machine (notation hexadécimale)	signification
LR	9,3	1893	Charger dans le registre général 9 le contenu du registre général 3.
MVC	0(4,8),0(5)	D20380005000	Copier 4 octets à l'adresse contenue dans le registre 8 à partir des 4 octets qui sont à l'adresse contenue dans le registre 5.
BR	14	07FE	Se brancher à l'instruction dont l'adresse est contenue dans le registre 14.
L	6,0(3)	58603000	Charger dans le registre 6 le contenu du mot de 4 octets qui est à l'adresse contenue dans le registre 3.

Ces exemples nous montrent le format des instructions:

- le premier octet est toujours le code instruction (18 pour LR, chargement registre, 58 pour L, chargement d'un mot dans un registre, etc.). Le code instruction occupe parfois les 2 premiers octets.

- les octets suivants indiquent tantôt la longueur sur laquelle porte l'instruction (cas du MVC), tantôt l'adresse des zones affectées, sous la forme déplacement + registre de base, ou déplacement + registre d'index + registre de base. L'adresse d'une donnée n'est jamais "en clair" dans l'instruction, elle résulte de

l'addition d'un déplacement (de 000 à FFF) et du contenu des registres de base ou d'index, qui sont les registres généraux, excepté le registre 0. Cet adressage indirect rend les programmes indépendants de l'endroit où ils seront chargés en mémoire: il suffit que le registre de base soit initialisé au début du programme à l'adresse de chargement, les adresses manipulées dans le programme seront exprimées par des déplacements par rapport à cette adresse de début.

E) Le système MVS : évolution et spécificités

MVS a été lancé en 1974 pour prendre la relève des systèmes MVT et SVS qui étaient déjà dépassés. Le but était de fournir un système très fiable pour supporter de gros environnements de production. La disponibilité fut améliorée en introduisant de nombreuses routines de reprise en cas d'erreur, capables d'isoler le problème et même de le réparer en reconstruisant les blocs de contrôle endommagés (une statistique non officielle chez IBM affirme que 50% du code MVS est voué à cette tâche). Aujourd'hui les "plantages" intempestifs de système sont devenus très rares. Les performances furent améliorées en augmentant le parallélisme des opérations, et cet effort a été constamment poursuivi à mesure que de nouvelles versions de MVS apparaissaient, sans que l'utilisateur n'ait à souffrir des modifications apportées, parfois profondes.

MVS est certainement le plus élaboré des systèmes d'exploitation IBM, sinon de tous les systèmes passés et présents. Ses détracteurs diront aussi qu'il est le plus complexe et le plus lourd de tous parce qu'il veut tout faire pour tout le monde... Comme son nom veut le faire comprendre ("multiple virtual storage"), il applique le principe de la mémoire virtuelle pour traiter différents travaux simultanément sur une machine comprenant un ou plusieurs processeurs (ce que ne peut faire un système tel que VSE, limité à un processeur). Il est destiné aux grosses machines que sont les 308X, 3090 et ES/9000, mais fonctionne aussi sur les 4381 et 9377-90.

Issu, comme VSE, de systèmes où le traitement par lots (batch processing) était la règle, MVS offre certainement une interactivité moins marquée que VM et son composant CMS (Conversational Monitoring System). Le traitement "batch", qui fut et est encore le lot quotidien de beaucoup de grands sites informatiques, consiste à soumettre à la machine un certain nombre de travaux (assez longs la plupart du temps) et à en attendre la "sortie" (fichiers, impressions) sans possibilité notable d'influer entre-temps sur les résultats. On parle ainsi de "mode différé", par opposition aux systèmes interactifs où des requêtes aisément modifiables et paramétrables reçoivent un résultat rapide sinon immédiat (sur écran en général). MVS se veut cependant universel, et gère des sous-systèmes aptes à répondre à des requêtes en mode interactif (appelé aussi "dialogué"), qu'il s'agisse aussi bien de "temps partagé" (l'ordinateur sert plusieurs utilisateurs - informaticiens le plus souvent - en même temps) que de "transactionnel", où l'utilisateur final, non forcément informaticien, dialogue avec des applications en rapport avec sa fonction dans l'entreprise. MVS reste cependant remarquablement adapté au traitement par lots, supporté par le "langage" JCL.

Schématiquement, les facteurs qui influencent le plus le développement d'un grand système tel que MVS, sont les suivants:

- la nécessité de supporter le passé pour préserver l'investissement du client (et aussi le dissuader de passer à la concurrence) oblige le constructeur à garder beaucoup de "vieilleries" et à conserver des options architecturales difficiles ou impossibles à outrepasser par la suite quand leurs limitations se font sentir ("barre" des 16 méga-octets, taille de bloc physique de fichier sur 2 octets, etc.);

- les besoins des plus gros clients contraignent le constructeur à faire évoluer matériel et logiciel vers plus de puissance, davantage de fonctionnalités, plus de sécurité et une disponibilité continue, plus de souplesse aussi ; les nouveaux ordinateurs, pour être mis complètement à contribution, conduisent à des adaptations du logiciel qui peuvent être importantes (en ce domaine le matériel précède toujours le logiciel). Les annonces d'IBM pendant ces dernières années montrent clairement un souci de fournir à ces clients qu'on appelle les "grands comptes" d'énormes capacités de traitement disponibles "24 h sur 24, 7 jours sur 7".

Sous l'effet de ces deux facteurs contradictoires, les systèmes deviennent de plus en plus volumineux. Il suffit pour s'en persuader de comparer au fil des années l'épaisseur de la brochure de référence "Principles of Operations". En 1960 l'IBM 1401 disposait d'un jeu d'instructions composé de 34 codes opérations; ce nombre passe à 143 en 1964 sur l'IBM 360; en 1981 il est de 204 sur les machines de type S/370, puis de 228 avec

ESA/370 sans compter les autres "facilités" qui peuvent être incorporées à la machine et font grossir d'autant le "POP". Parallèlement, alors qu'un système "préhistorique" tel que FMS (Fortran Monitor System) sur les IBM 704 et 7090 occupait un million d'octets, la taille de MVS aujourd'hui se mesure en milliards d'octets et en dizaines de millions de lignes de code.

Pour faire face à une complexité croissante, la tendance est à la modularité (une règle générale qui vaut autant pour le matériel que le logiciel). Le programmeur d'aujourd'hui en bénéficie directement : il n'a plus à se soucier (comme c'était le cas dans les années 60) du détail des entrées-sorties, de la gestion du terminal ou des lignes de communication, du recouvrement en cas de problème, etc. : des composants spécialisés offrant des interfaces bien définies lui évitent de se charger de ces tâches ingrates.

Actuellement quatre versions de MVS existent, chronologiquement:

- MVS/SP1 (appelé aussi MVS/370, ou "MVS/SP"): offre à chaque travail en machine une mémoire virtuelle de 16 méga-octets (limitation due aux adresses sur 3 octets) tout en pouvant gérer une mémoire réelle jusqu'à 64 méga-octets (les adresses réelles pouvant être en interne sur 26 bits). Le nombre de canaux vus "logiquement" est de 16 (même s'il peut y avoir un peu plus de canaux physiques en réalité). Cette version de MVS (stabilisée à 1.3.6) n'est plus commercialisée par IBM.

- MVS/SP2 (appelé aussi MVS/XA, MVS extended architecture): date de 1981 et correspond à une architecture 370 "étendue". Les adresses sont codées sur 31 bits (4 octets moins le bit de signe des nombres binaires de 1 mot) ce qui offre une mémoire virtuelle de 2 giga-octets (2 milliards d'octets). La mémoire réelle supportée pourrait être aussi de 2 giga-octets (en fait elle se limite à 512 méga-octets) et le nombre de canaux peut être porté à 256. La gestion des entrées-sorties a été refondue et s'appuie beaucoup plus sur le matériel, ce qui les améliore nettement par rapport à MVS/SP1. MVS/XA n'est plus vendu par IBM ; il a été stabilisé à la version 2.2.3 et sera remplacé par MVS/ESA.

- MVS/SP3 (ou MVS/ESA): avec cette version lancée par IBM en 1988 on dispose d'une mémoire virtuelle de 16 téra-octets (avec 8 000 espaces "données" de 2 giga-octets). Cette mémoire virtuelle n'est pas adressable directement (l'adressage reste à 31 bits comme en MVS/SP2), mais elle peut être référencée grâce à un nouveau jeu de registres. Cette version, qui au début intéressait plutôt les très grands sites arrivés au bout des capacités de MVS/XA, n'était disponible que sur les 3090 modèles E, S ou J et certains 4381. MVS/ESA devient à présent la version unique de MVS, qui s'imposera sur tous les matériels.

- MVS/SP4 (MVS/ESA SP4): cette dernière version, annoncée par IBM en 1990 conjointement avec les S/390, supporte les canaux ESCON, la gestion de complexes de systèmes MVS par le Sysplex, et comporte de nombreux autres apports. Elle poursuit l'évolution vers le concept de DIM (Data in Memory) apparu depuis ESA.

Les caractéristiques du système MVS qui seront étudiées plus loin dans cet ouvrage sont les suivantes:

- multiprogrammation: plusieurs utilisateurs ou travaux peuvent être servis simultanément par la machine. "Simultanément" signifie simplement ceci: bien qu'à un instant donné un seul travail puisse être traité par un processeur, dans les conditions habituelles tous les utilisateurs sont servis à tour de rôle si rapidement qu'à l'échelle humaine ils paraissent utiliser tous en même temps la machine. Nous verrons plus loin comment le processeur est capable d'abandonner le travail en cours pour en traiter un autre.

MVS réalise aussi une fonction de multi-tâche, plus fine que celle de multiprogrammation: un travail peut être constitué de différentes tâches fonctionnant les unes après les autres ou simultanément: dans ce dernier cas elles entrent ainsi en compétition pour l'usage du processeur.

La multiprogrammation et le multi-tâche offrent une "simultanéité" et une multiplicité d'accès au processeur satisfaisantes pour tous les utilisateurs. Ils optimisent aussi l'emploi du processeur en réduisant au maximum ses "temps morts": quand le travail en cours attend un événement quelconque (notamment la fin d'une entrée-sortie, opération "longue"), le processeur peut l'abandonner pour un autre processus prêt à être activé.

- multitraitement: plusieurs processeurs peuvent être gérés par le système (jusqu'à 2 en MVS-SP, 16 à partir de MVS-XA). Ainsi, à un instant donné, sont traités par la machine autant de processus que de

processeurs. Ceci améliore évidemment le rendement global du système. Sur le marché, la tendance est d'ailleurs, depuis quelques années, aux machines multi-processeurs (IBM vend des machines 3090 et ES/9000 ayant jusqu'à 6 processeurs).

- la mémoire virtuelle: chaque travail se voit offrir une taille de mémoire possible pouvant être bien supérieure à la mémoire réellement disponible. Ceci est rendu possible par le fait qu'à un instant donné un travail n'a pas besoin de toutes les données enregistrées en mémoire depuis le départ. Le système lui laissera les portions de mémoire qui lui sont nécessaires à cet instant, le reste pourra être transféré sur disque (on parle alors de mémoire auxiliaire), récupérable à tout moment à la demande du travail. Le reste de la mémoire est utilement occupé par les routines et données du système ou par d'autres travaux prêts à être traités par le ou les processeurs.

Quand on parle de mémoire virtuelle ou d'espace-adresse, il s'agit de mémoire disponible (au moins conceptuellement) pour le programmeur et le travail qu'il compte soumettre à la machine. On laisse au système le soin de savoir où seront en pratique les zones de mémoire utilisées. Elles seront situées en fait tantôt en mémoire réelle, et à des adresses qui n'ont rien à voir avec les adresses virtuelles que "voit" le programmeur, tantôt sur disque ou autre mémoire secondaire si ces zones sont devenues inutilisées ou si le travail lui-même a été mis en attente et n'utilise pas le processeur pour le moment. La seule limitation de la taille de la mémoire virtuelle est la longueur de l'adresse (24 ou 31 bits).

Le système lui-même occupe constamment une certaine partie de la mémoire virtuelle (et de la mémoire réelle). Pour des raisons de performance certaines parties sont toujours résidentes en mémoire comme le noyau du système. D'autres sont souvent en mémoire mais peuvent être transférées sur disque si le système a besoin de récupérer de la place mémoire. Enfin certaines parties sont toujours sur disque (sous forme de programme exécutable) et chargées en mémoire uniquement quand on les demande. On est de nos jours bien loin du temps où le système était chargé par l'opérateur en même temps que le travail à exécuter, pour disparaître ensuite de la mémoire...

Un système d'exploitation tel que MVS, puissant, complexe, n'en est pas moins une oeuvre humaine complètement artificielle, et doit donc pouvoir être décrit sinon dans le détail, du moins en suivant les grandes lignes qui ont présidé à son élaboration. Comme il est matériellement impossible à l'informaticien le plus patient d'entrer dans ces millions de lignes de programme (qu'lBM ne rend d'ailleurs pas disponibles conformément à sa politique "OCO") qui donnent en fin de compte ce qu'on appelle MVS, et comme il n'est pas possible non plus de rendre compte de toutes les évolutions qui distinguent telle version ou telle "release" de telle autre, nous nous en tiendrons nécessairement à un aperçu synthétique de son fonctionnement dans la suite de ce livre. La version de MVS qui nous servira de référence sera le plus souvent MVS/ESA, de plus en plus utilisé, alors que MVS/SP1 et SP2 sont déjà obsolètes, et n'ont fait, dans l'esprit d'IBM, que montrer la voie vers les grands espaces (magnétiques et électroniques) du futur...

2. LA RESSOURCE PROCESSEUR

A) Notions de base

Nous allons donner quelques définitions avant de décrire le fonctionnement des processeurs en MVS. Il s'agit de notions qui ne sont pas toutes spécifiques à MVS, mais plutôt propres aux architectures 370. Nous désignerons par configuration "multi-processeur" tout ordinateur constitué de plusieurs processeurs capables de communiquer directement entre eux et partageant la même mémoire (et donc régis par le même système MVS).

LES INTERRUPTIONS: notion capitale. Tout événement venant interrompre le processeur dans la séquence d'instructions en cours est une interruption. On distingue 6 sortes d'interruptions en architecture 370:

entrée-sortie: le matériel signale au système par exemple la fin d'une entrée-sortie (opération demandée par un processeur), un changement d'état d'un périphérique. La nécessité de rendre le fonctionnement du processeur indépendant du déroulement des entrées-sorties a motivé l'emploi d'un signal d'interruption pour tenir le processeur informé; dès 1958 IBM introduisait sur ses ordinateurs la notion d'interruption de programme.

erreur machine ("machine check"): signale un mauvais fonctionnement de l'ordinateur: problème de fiabilité des informations mémoire ou registre (détecté par les bits de contrôle, erreurs éventuellement corrigibles si les circuits sont doublés ou triplés), problème d'alimentation électrique, problèmes de refroidissement, etc.

redémarrage (restart): peut survenir quand l'opérateur remet la machine en fonction, en intervenant par exemple après la commande MVS QUIESCE (qui met la machine en attente) ou quand un processeur en démarre un autre en environnement multi-processeur (par l'instruction SIGP, "signal processor").

interruption externe: quand l'opérateur appuie sur la touche "interrupt", quand un intervalle de temps s'est écoulé, ou sur signal d'un autre processeur (toujours par SIGP).

interruption programme ("program check"): lors d'erreur incorrigible dans un programme (division par zéro, tentative d'exécution d'une "instruction" de code instruction inconnu, tentative d'accéder à une zone mémoire inconnue ou protégée, etc.). Ces erreurs provoquent le plus souvent un ABEND (abnormal end) qui entraîne l'arrêt du travail fautif. Des interruptions programmes peuvent survenir aussi quand on référence des zones qui ne sont plus en mémoire (on parle de "faute de page"); il ne s'agit pas alors d'erreurs. De même pour d'autres interruptions programme liées aux fonctions de suivi et de "trace" (system trace, GTF trace, SLIP) que propose MVS.

appel superviseur ("SVC call"): il s'agit d'un appel aux routines du système (par l'instruction SVC) pour certaines fonctions déterminées: demande d'espace mémoire (GETMAIN), envoi de message au pupitreur (WTO write to operator"), demande de mise en attente (WAIT), d'abandon du programme (ABEND), d'exécution d'entrée-sortie (EXCP, execute channel program), etc. Il y a au maximum 256 sortes de SVC (portant des numéros de O à 255). Il s'agit de services toujours disponibles dont le programmeur se décharge sur le système, non seulement par facilité de programmation, mais aussi parce que ces services exigent un privilège dont ne disposent pas les programmes "courants" (voir plus loin les états du processeur).

Les deux derniers types d'interruption sont internes au programme actif dans la machine (il en est la cause directe -sauf dans certains cas de "program check") tandis que les 4 premiers surviennent de façon "aléatoire" (ils sont liés à des éléments extérieurs les périphériques, l'opérateur humain). On voit ici un aspect d'indéterminisme dans le fonctionnement de l'ordinateur prévu certes pour faire ce qu'on attend de lui et pas autre chose, mais trop complexe pour qu'on puisse savoir ce qu'il fera à un instant donné dans le futur..

Sans l'existence d'une possibilité d'interruption du processeur par des événements asynchrones, l'ordinateur serait une machine extrêmement limitée, non optimisée, sans possibilité de multiprogrammation, figée dans

une séquence prédéterminée d'instructions. Les premiers ordinateurs commandaient directement les périphériques et perdaient beaucoup de temps en attente improductive. Le mécanisme de l'interruption participe à la puissance et à l'universalité des ordinateurs actuels.

Le traitement d'une interruption implique plusieurs choses:

* il faut sauvegarder tout ce qui est nécessaire pour pouvoir reprendre plus tard le processus interrompu;

* il faut savoir quelle nouvelle séquence d'instructions doit être exécutée;

Ce traitement est effectué par le matériel (par un traitement câblé, ou par le microcode en XA). Nous étudierons plus loin le mécanisme de l'interruption.

Notons que le processeur peut se permettre d'ignorer certaines interruptions (on dit qu'il les "masque") quand il est en train d'exécuter certaines routines "critiques" qui doivent absolument se terminer sans risquer d'être arrêtées en cours de déroulement (par exemple les routines de traitement d'interruptions ne doivent pas être interrompues, sous peine d'aboutir à des cascades indéfinies d'interruptions). On parle alors de mode "disable" (non interruptible). Les interruptions qui surviennent peuvent être momentanément ignorées pour être traitées plus tard, à la sortie du mode disable (de retour au mode "enable", interruptible). Signalons que la majorité des instructions ne sont pas interruptibles, et que l'interruption survient donc entre deux instructions consécutives. Certaines interruptions ne peuvent être masquées: les interruptions SVC, certaines interruptions erreur programme et erreur machine.

LES ÉTATS DU PROCESSEUR

En architecture 370 le processeur fonctionne dans 2 états différents: l'état "problème" (problem program) qui le limite à un certain jeu d'instructions (supposées être inoffensives pour le système lui-même) et l'état "superviseur" dans lequel toutes les instructions sont permises (c'est un privilège réservé au système ou aux programmes "autorisés").

On parle parfois aussi de mode "maître" (ou "contrôle") et de mode "esclave". C'est une protection à la fois contre les erreurs involontaires des programmeurs ou des utilisateurs, et aussi contre la malveillance...

LE MODE D'ADRESSAGE

Nous avons vu que la version SP1 de MVS traitait des adresses virtuelles de données sur 24 bits, tandis que SP2, SP3 et SP4 pouvaient aller jusqu'à 31 bits. Ces dernières versions ont conservé par compatibilité la possibilité de ne prendre en compte que 24 bits, comme en SP1. On parle donc de mode d'adressage à 31 bits ou à 24 bits, ce dernier empêchant l'accès aux données ou aux programmes d'adresses virtuelles supérieures à FFFFFF hexadécimal (adresse maximale qu'on peut coder sur 24 bits, on parle ainsi de "barre des 16 Mégaoctets"). A tout moment le processeur fonctionne dans l'un ou dans l'autre de ces modes d'adressage.

LE DAT (dynamic address translation)

Nous avons distingué les adresses virtuelles, "vues" par le programmeur, des adresses réelles, évidemment nécessaires, en dernier lieu, pour accéder aux zones de mémoire requises. Le DAT est le mécanisme qui traduit les adresses virtuelles en adresses réelles.

Un programme fonctionne habituellement en utilisant des adresses virtuelles: on dit qu'il tourne en mode V = V (virtuel = virtuel). Cependant il pourrait très bien ne pas requérir la traduction d'adresse par le DAT s'il utilisait exclusivement des adresses réelles, en mode V = R (virtuel = réel), en démarrant avec le paramètre ADDRSPC = REAL. Certaines routines du système d'exploitation sont d'ailleurs prévues pour fonctionner sans le DAT ("DAT-off" coding) pour le cas où celui-ci serait exceptionnellement inopérant.

TCB et SRB

Il s'agit des deux espèces d'unité de travail (ce terme -"unit of work"- désignant une demande de travail adressée au processeur) que peut traiter le processeur en MVS. Le TCB (Task Control Block) représente une

tâche liée à un espace-adresse, c'est le mode habituel d'exploitation du processeur. Le SRB (Service Request Block) est une demande de service système de haute priorité pouvant concerner tous les utilisateurs et dont le système peut demander l'exécution dans un espace-adresse donné.

Une tâche, représentée par le bloc de contrôle TCB, est créée par la macro-instruction ATTACH et détruite par DETACH (ces macro-instructions génèrent des interruptions SVC). Une tâche peut elle-même créer une autre tâche; l'environnement de gestion d'une tâche en MVS est assez lourd, et chaque espace-adresse est constitué d'au moins 4 tâches, la première étant la RCT (region control task), la tâche "mère".

Un SRB est un bloc de contrôle pratiquement indépendant du reste du système, de petite taille, préparant l'appel à une routine très courte et peu coûteuse en ressource. La création d'un SRB s'effectue par la macro-instruction SCHEDULE, l'annulation par PURGEDQ. Une routine SRB tourne en mode superviseur et ne peut émettre de SVC. Elle est interruptible, mais elle a le privilège de reprendre le contrôle après le traitement de toute interruption survenant pendant son exécution (on dit qu'elle est "non-preemptible"). Nous donnons en annexe un exemple de programme de création de SRB. Le SRB est utilisé comme moyen de communication entre espaces-adresses, et de nombreux composants du système ou produits annexes en font un usage abondant; le traitement des interruptions d'entrée-sortie donne lieu aussi à la création systématique de SRB destinés aux espaces-adresses demandeurs.

Le MOT d'ÉTAT PROGRAMME (PSW, program status word)

Le mot d'état programme PSW est une sorte de registre du processeur qui caractérise l'état de la routine qui est en cours d'exécution sur le processeur. Le PSW réunit les informations capitales qu'il faut sauvegarder pour le processus en cours s'il vient à être interrompu. Les 64 bits du PSW ont chacun une signification bien précise, entre autres:

- le 6e bit indique si le DAT doit être utilisé pour traduire les adresses (il vaut 1 pour oui, 0 pour non);
- les 7e, 8e, 14e bits indiquent si le processeur est interruptible (enable) respectivement pour les interruptions d'entrée-sortie, externes et d'erreur machine (1 pour oui, 0 pour non);
- le 15e bit indique si le processeur est en attente (wait state): c'est le cas si Ie bit vaut 1 (le processeur n'a pas d'instructions à traiter);
- le 16e bit donne l'état du processeur (1 pour l'état problème, 0 pour l'état superviseur);
- les 17e et 18e bits correspondent au mode secondaire (cross-memory) et au mode AR (architecture ESA/370) ;
- le 33e bit donne le mode d'adressage en vigueur pour le programme en cours de traitement (1 pour l'adressage à 31 bits, 0 pour celui à 24 bits);
- les 9e à 12e bits constituent les 4 bits de la clé d'accès-mémoire du programme. Nous verrons dans le chapitre suivant que cette clé que possède le programme lui "ouvre" l'accès à des zones mémoire de même clé;
- les 19e et 20e bits constituent le code-condition, qui est évalué après de nombreuses instructions: il peut être testé par exemple après une instruction de comparaison;
- les 34e à 64e bits constituent l'adresse à 24 ou 31 bits de la prochaine instruction du programme à exécuter (adresse virtuelle ou réelle selon la valeur du 6e bit du PSW). On devine l'intérêt de cette information: ce sera l'adresse de reprise du processus s'il vient à être interrompu;
- enfin plusieurs bits permettent de masquer certains types d'interruption programme (dépassements de capacité dans les calculs).

Nous allons voir par la suite le rôle fondamental du PSW dans le traitement des interruptions.

B) Les interruptions

Une interruption, rappelons-le, est un arrêt automatique de l'exécution du programme en cours, causé par un événement bien déterminé (interne ou externe au système) au profit d'une séquence d'instructions censée traiter cet événement. Une interruption n'est prise en compte que si le processeur est interruptible pour ce type d'interruption.

Le mécanisme de traitement d'une interruption est celui-ci:

1) On sauvegarde le PSW courant dans une zone fixe de la mémoire (fonction du type d'interruption): on parle d'ancien PSW (old PSW). Puis on charge le PSW qui traite ce type d'interruption (new PSW). Il y a ainsi 6 couples possibles ancien PSW/nouveau PSW correspondant aux 6 types d'interruptions de l'architecture IBM 370. L'échange entre les PSWs est effectué par le matériel (câblage ou microcode). Les 6 nouveaux PSWs possibles sont invariants et créés au démarrage du système.

2) Le nouveau PSW branche le processeur vers la routine de traitement d'interruption de premier niveau particulière à ce type d'interruption (on a donc 6 FLIHs: first level interrupt handlers). Cette routine est résidente en mémoire (elle fait partie du "noyau" du système). Les registres sont sauvegardés en mémoire centrale par cette routine de façon à pouvoir être restaurés plus tard avec les valeurs qu'ils avaient lors de l'interruption: on pourra ainsi reprendre le processus interrompu exactement au point où il en était avant l'interruption (à noter que d'autres systèmes utilisent deux jeux de registres, ce qui évite sauvegarde et restauration). La sauvegarde des registres s'effectue dans les 4 premiers K-octets de la mémoire virtuelle (de façon temporaire - elle sera faite ensuite de manière plus définitive en mémoire virtuelle de l'espace-adresse interrompu). Nous verrons que cette zone des 4 premiers K-octets (la PSA) n'est pas commune à tous les processeurs, mais que chacun a la sienne (ce qui est nécessaire pour un traitement en parallèle). D'autres informations sont stockées dans cette portion de mémoire, notamment un code interruption identifiant plus précisément l'interruption.

Le processeur fonctionne nécessairement jusqu'ici en mode superviseur ininterruptible; il ne pourrait prendre en compte une nouvelle interruption sans risque pour le processus dont il traite l'interruption (I'interruption SVC ne peut être masquée, mais elle ne peut survenir pendant le traitement du FLIH, celui-ci n'effectuant jamais de SVC). Une fois la sauvegarde effectuée, il peut revenir (éventuellement) en mode interruptible, et, après l'analyse de l'interruption, passer le contrôle à une routine spécifique.

3) Une routine de traitement d'interruption de second niveau (SLIH, second level interrupt handler) peut alors s'exécuter pour répondre exactement à l'interruption. Selon le type d'interruption, il pourra s'agir :

- d'une routine de l'IOS (input/output supervisor, superviseur d'entrée-sortie) pour une interruption d'entrée-sortie;

- d'une routine de RTM (recovery termination manager, composant responsable de la reprise en cas d'erreur) pour les interruptions de restart, d'erreur machine et d'erreur programme;

- d'une routine spécifique pour l'interruption externe;

- d'une routine de trace pour certains "program checks";

- d'une routine SVC pour l'interruption SVC. Certaines routines SVC peuvent elles-mêmes émettre des instructions SVC (elles sont donc interruptibles, ce qui n'est pas le cas général des routines SLIH et FLIH): en prévision de cela, le "SVC FLIH" construit préalablement un bloc de contrôle SVRB (SVC request block) qui servira à sauvegarder le contexte d'exécution de la routine SVC.

4) Après le traitement de l'interruption, le contrôle n'est pas forcément rendu à la routine interrompue: ce n'est le cas que pour des SRBs (de façon systématique) ou pour des TCBs de type "non-preemptible" (c'est le cas pour

certaines routines SVC). Le contrôle est le plus souvent rendu à un composant du système que nous étudierons plus loin: le distributeur.

Plusieurs interruptions peuvent survenir simultanément. Cependant, compte tenu du mécanisme simple d'échange de PSWs ancien/nouveau (sans gestion de pile d'interruptions comme dans d'autres systèmes), ne peuvent être prises en compte simultanément par le système que des interruptions de type différent: le système ne peut donc avoir à traiter simultanément que 6 interruptions au maximum. En réalité le matériel peut avoir généré beaucoup plus d'interruptions; parmi des interruptions de même type, ne sera "présentée" au système que celle de priorité maximale, les autres étant différées. La priorité des interruptions au sein d'un même type est, par exemple, fonction de l'unité pour une interruption entrée-sortie (il y a 8 sous-classes possibles, de O à 7), ou fonction du genre d'interruption pour une interruption externe (l'interrupt key étant la plus prioritaire et le "service signal" la moins). Les 6 types d'interruption sont traitées dans l'ordre suivant (en l'absence d'erreur machine grave):

- SVC
- "program-check"
- "machine-check"
- interruption externe
- entrée-sortie
- "restart"

Si par exemple surviennent en même temps 6 interruptions de type différent (le maximum que puisse traiter le système), surviendront alors 6 échanges successifs ancien PSW/nouveau PSW dans l'ordre inverse des priorités, en commençant par le "restart" et en terminant par le "SVC" (le nouveau PSW d'un type devenant l'ancien PSW d'une autre interruption): ainsi, par branchement au dernier "nouveau PSW", sera traitée l'interruption la plus prioritaire, puis celles de priorités inférieures.

C) Le distributeur (dispatcher)

1) DESCRIPTION

Le distributeur est un composant de MVS (il fait partie du noyau) dont le rôle est de rechercher le travail le plus prioritaire pour lui fournir la ressource processeur dont il a besoin. Le distributeur est appelé en particulier après le traitement d'une interruption: on voit que son intervention est le plupart du temps aléatoire.

Les espaces-adresses présents en machine sont représentés vis-à-vis du distributeur par un ASCB (address-space control block). Chaque espace-adresse a une priorité de dispatching (DPRTY) valeur binaire valant de 0à 255, déterminée par les paramètres du système, mais modifiable dynamiquement par le système (pour l'équilibrage par le composant SRM - étudié dans un chapitre postérieur - à l'aide de la macro CHAP), ou par l'opérateur par une commande RESET. Un certain nombre de TCBs ou de SRBs sont liés à un espace-adresse et peuvent requérir la ressource processeur. Voici l'ordre que respecte le distributeur pour l'attribution du processeur:

- les routines "special exits", si elles existent, ont les premières le contrôle. 11 s'agit de routines appelées dans des cas particuliers qui touchent à la disponibilité du processeur en environnement multi-processeur (ACR - alternate cpu recovery- qui invoque RTM en cas de problème sur un processeur, Vary Cpu - qui rend disponible ou indisponible un processeur -) ou à la synchronisation des TOD clocks des processeurs (Timer recovery);

- la file des SRBs "globaux" est la plus prioritaire (quel que soit l'espace-adresse auquel ils sont rattachés); il s'agit de services urgents pouvant impacter tous les utilisateurs du système et qui ne peuvent souffrir de retard;

- s'il n'y a pas de SRBs globaux, on considère l'ASCB de plus grande priorité: la file des SRBs "locaux" rattachés à cet ASCB est traitée d'abord, puis, s'il n'a pas de SRB à prendre en compte, la file des TCBs (les TCBs d'un espace-adresse peuvent avoir eux-mêmes différentes priorités). Privilégier la plus haute priorité laisse peu de ressource processeur aux processus de faible priorité, qui ne seront guère actifs qu'une fois que

tous les processus de priorité supérieure seront inactifs (en attente de résultat d'entrée-sortie ou d'autres événements). En pratique, en mettant de côté les espaces-adresses que l'on estime très importants et auxquels on affectera une forte priorité, on équilibrera l'utilisation du processeur en accordant la meilleure priorité aux travaux effectuant beaucoup d'entrées-sorties et la plus basse aux travaux très "gourmands" en ressource processeur (grands calculs par exemple);

- s'il n'y a aucun travail à traiter (aucun n'étant prêt à s'exécuter) le processeur charge un PSW d'attente interruptible (le bit "wait" du PSW est positionné): il existe en réalité un ASCB fictif de wait.

Le contrôle est passé au SRB ou au TCB choisi par l'instruction LPSW (load PSW) après que le contexte du processus (ses registres) a été restauré. Le distributeur, appelé après une interruption, sauve le contexte du TCB ou du SRB; il maintient dans l'ASCB les compteurs de temps d'utilisation du processeur en mode TCB et SRB (sauf pour cette tâche particulière qu'est la RCT) ainsi que les temps écoulés depuis le démarrage du travail (temps "elapse").

A titre de comparaison, le distributeur du système VM travaille sur une base plus égalitaire: chaque machine virtuelle reçoit une tranche de temps ("time-slice") d'execution sur le processeur. La liste des machines "éligibles" est composée de 4 listes EO E1 E2 E3, chaque liste correspondant à un taux de consommation différent, une machine pouvant passer d'une liste favorisée à une liste moins favorisée si son travail n'est pas encore terminé. Il n'y a pas de priorité de dispatching explicite, mais on peut favoriser une machine en lui accordant plus de ressource processeur que les autres (commande SET SHARE). Chaque processeur a sa propre file de machines virtuelles à dispatcher (14 machines au maximum en VM-XA SP) mais le processeur maître peut traiter les files des autres processeurs s'il n'a plus rien à faire. De nombreux systèmes orientés vers le temps partagé (time sharing) ont aussi un distributeur qui alloue le processeur par tranches de temps à chaque processus, un à un (système du tourniquet). Cela implique qu'aucun processus n'est plus prioritaire qu'un autre (le travail du distributeur
est alors assez limité). Ce n'est pas le choix qui a été fait pour MVS. MVS utilise un principe de priorités dynamiques (nous verrons dans le chapitre sur SRM comment sont déterminées et éventuellement modifiées les priorités).

2) LE MEMORY SWITCH

Le mécanisme du distributeur MVS tel que nous l'avons décrit n'assure pas la mise à disposition immédiate du processeur (ou d'un processeur) à un processus de haute priorité: ce processus devrait attendre l'arrivée (aléatoire) d'une interruption pour qu'une fois celle-ci traitée le distributeur le prenne en compte .

Un autre mécanisme résoud le problème: le "memory switch" (commutation mémoire), qui est invoqué à chaque fois qu'un SRB (par la macro SCHEDULE) ou qu'un TCB (par la macro POST qui met fin à un wait) est prêt à utiliser le processeur. Il est invoqué aussi pour tout changement d'état d'une tâche vis-à-vis du processeur (changement de priorité par la macro CHAP, mise en état "dispatchable" par la macro STATUS). Le memory switch n'est cependant vraiment intéressant que dans une configuration multi-processeur.

Le memory switch, quand il est invoqué, s'assure d'abord que l'unité de travail TCB ou SRB est mise dans la file d'attente du distributeur; puis il recherche un processeur inactif (en "wait"). S'il en trouve un, il émet une instruction SIGP à destination de ce processeur. Ce processeur reçoit le SIGP comme une interruption externe (on pourrait dire qu'il est ainsi "réveillé"), ce qui le conduit (après traitement de l'interruption) à invoquer le distributeur et à prendre en compte le TCB ou le SRB. Si le memory switch ne trouve pas de processeur en attente, il considère les priorités de dispatching des TCBs ou SRBs en cours de traitement sur les différents processeurs, il interrompt par SIGP le premier processeur qu'il trouve en train de traiter un TCB ou un SRB de priorité inférieure au TCB ou SRB qui est cause du memory switch. Ainsi sera prise en compte l'existence du nouveau-venu, plus prioritaire. On se prémunit aussi de la sorte contre le risque de voir un processus "très gourmand" en ressource processeur ("CPU-bound") monopoliser la machine.

Pour résumer, les différents composants qui invoquent le distributeur sont: les routines de traitement d'interruption (entrée-sortie et externe uniquement), SRM (algorithmes de gestion des priorités), et toutes les routines qui changent l'état d'un processus (CHAP, POST) ou en créent de nouveaux (SCHEDULE SRB).

3) LA FILE D'ATTENTE DU DISTRIBUTEUR

Un processus peut être dans un des états suivants:

* actif: il s'exécute sur un processeur;

* prêt: il est prêt à être activé mais la ressource processeur ne lui est pas accordée (un processus de plus haute priorité est exécuté);

* en attente: il ne peut s'exécuter car il attend un ou plusieurs événements (une entrée-sortie qu'il a demandée, une faute de page, une ressource indisponible, la réponse du pupitreur à un message, etc.).

Il y a eu, au cours des différentes releases de MVS, plusieurs façons de traiter la file d'attente (ou plutôt les files, car il y a des files séparées pour SRB globaux, locaux et ASCBs) du distributeur. Petit à petit on a optimisé l'emploi du distributeur en allégeant cette file. Les premières versions de MVS maintiennent une file de tous les espaces-adresses par ordre de priorité avec un numéro de séquence. Plutôt que d'examiner systématiquement si l'espace-adresse de plus haute priorité a du travail à faire (SRB ou TCB), on maintient dans la PSA du processeur un champ qui représente l'espace-adresse de priorité immédiatement inférieure à celui qui est actif (PSAANEW), et qui sera en principe le suivant à être dispatché. Le "memory switch", en agissant sur ce champ, pourra toujours faire un "rappel à l'ordre" si un espace-adresse de priorité supérieure vient à être "dispatchable". Avec MVS-SP2 (XA) le distributeur ne s'intéresse plus qu'aux travaux présents en mémoire (swapped-in queue), actifs ou non.

Quand le distributeur ne trouve aucun travail à dispatcher, avant de charger un PSW d'attente, il se met en mode interruptible et re-inspecte toute la file (cela ne coûte guère plus cher que de se mettre en "wait"
tout de suite). En MVS-SP213, il se dispense d'inspecter la file dès qu'il trouve des travaux "vidés logiquement" (inactifs, mais restés en mémoire, voir chapitre suivant). Enfin, à partir de MVS-SP217, une autre file d'attente apparaît: la TRQ (true ready queue, file des processus "réellement prêts"), qui contient, triés par priorité, les travaux prêts à s'exécuter. Cette file est plus petite que la file des versions précédentes; le premier de la file est le travail à dispatcher en priorité, et la zone "PSAANEW" ne sert plus à cet effet. Le deuxième "tour" du distributeur est supprimé: dès que la TRQ est vide on se met en attente. On résout aussi le problème des travaux de même priorité, qui avaient tendance à garder la même position relative (fonction de leur heure de démarrage respective) quand on ne leur imposait pas l'algorithme de rotation des priorités de SRM. A partir de MVS-SP217, le premier travail prêt à s'exécuter précède dans la TRQ ceux de même priorité qui sont prêts "plus tard" et arrivent après dans la file.

D) Les multi-processeurs

1) TYPES DE MULTI-PROCESSEURS

Les configurations multi-processeurs apportent un parallélisme dans l'exécution des travaux (n processus pouvant être simultanément actifs sur n processeurs), une puissance processeur globalement plus grande et une sécurité accrue (la défaillance d'un processeur n'entraînant pas l'arrêt du système). En contrepartie, le système doit savoir gérer plusieurs processeurs, et nous verrons les problèmes supplémentaires que cela entraîne. MVS-SP supporte au plus deux processeurs et MVS-XA peut en supporter jusqu'à 16.

A partir de la configuration la plus simple, le monoprocesseur (UP, uniprocessor), on peut obtenir les configurations suivantes:

le processeur attaché (**AP**, attached processor): au processeur de base, qui seul est connecté aux canaux, est rattaché un processeur qui apporte de la puissance supplémentaire, sans pouvoir accéder aux périphériques. La mémoire est commune aux 2 processeurs.

le **MP** (multi-processor, on parle aussi de "dual processors" pour les 4381): deux processeurs UP ont une mémoire commune mais chacun a son propre jeu de canaux (channel set). Si l'un des processeurs est défaillant, l'autre peut "reprendre" son jeu de canaux alternativement avec son propre jeu ("channel set switching" en MVS-SP1). Par exemple: le 3033MP.

Avec MVS-XA est apparue la configuration **dyadique**; un composant supplémentaire apparaît: le "system controller" qui sert d'interface entre la mémoire, les canaux (channel subsystem, ou channel control element pour les 3090) et les deux processeurs (pour un 3081, un 3090-200) ou le processeur (pour un 3083 ou les premiers 3090 monoprocesseurs). Un dispositif semblable existait déjà sur le 3033 MP(le "multisystem communication unit"). Les canaux ne sont pas dédiés à un processeur ou à un autre: chaque processeur est "symétrique" pour les entrées-sorties (ce qui permet à MVS-XA de faire l'économie d'un "channel set switching").

Existent enfin les multi-processeurs dyadiques (3084, 3090-400), obtenus par couplage de processeurs dyadiques, qui peuvent fonctionner dans 2 modes:

* le mode partitionné qui permet d'avoir plusieurs systèmes différents ("sides"), avec chacun sa mémoire et son jeu de canaux;

* le mode image unique ("single image"), obtenu par connexion des "system controllers", qui donne une seule mémoire et un jeu de canaux commun.

IBM parle traditionnellement de TCMP (tightly coupled multiprocessing),couplage "étroit", pour désigner toutes ses configurations multi-processeurs (une mémoire unique, plusieurs processeurs pouvant communiquer directement entre eux et régis par un système d'exploitation unique), par opposition au LCMP (loosely coupled multiprocessing),couplage "lâche" entre plusieurs ordinateurs (et plusieurs systèmes),dont nous reparlerons avec JES3, et qui n'est pas vraiment à notre sens une configuration multi-processeur (différents systèmes connectés par une liaison canal à canal).

2) LE PRÉFIXAGE

Le multi-traitement implique le partage d'une même mémoire réelle entre plusieurs processeurs: s'y trouvent les programmes et les données de processus susceptibles de s'exécuter sur l'un ou l'autre des processeurs. Cependant il existe un cas bien précis où ce partage de la mémoire pourrait poser un problème .

La PSA (prefix storage area) est la zone des 4 premiers K virtuels/réels(adresses 0 à 4 095) utilisée par le processeur (matériel et logiciel)quand il traite une interruption (voir plus haut). Le parallélisme des opérations implique que chaque processeur doive avoir sa propre PSA, pour traiter les interruptions qui lui sont adressées. Comme chaque PSA est "vue" à l'adresse zéro (ceci, par convention, pour simplifier la programmation des routines systèmes), on utilise un artifice qui permet de maintenir en mémoire réelle plusieurs PSA. Chaque processeur est pourvu d'un registre préfixe (prefix storage register) contenant l'adresse en mémoire réelle de sa PSA (établie une fois pour toutes au démarrage et obtenue dans une autre zone système, la SQA). A toute adresse réelle comprise entre 0 et 4 095 (soit les 20 premiers bits de l'adresse à zéro) est ajouté le contenu du registre préfixe, ce qui permet l'accès à la PSA du processeur (l'adresse "réelle" n'en étant plus une dans ce cas bien particulier, on parle d'adresse "absolue").L'accès à l'adresse absolue zéro (où se trouve une PSA non préfixée, contenant des données communes aux processeurs) s'effectue inversement par une adresse réelle augmentée préalablement du contenu du registre préfixe du processeur (reverse prefixing).

3) LA COMMUNICATION ENTRE PROCESSEURS

Outre les fonctions purement matérielles telles que la synchronisation des horloges (TOD clocks) de chaque processeur, en cas de problème sur un processeur ou la synchronisation mémoire en cas de traitement concurrent d'une même zone, le système, par logiciel, permet aux processeurs de communiquer entre eux. L'instruction SIGP permet à un processeur de s'adresser à un autre pour lui demander de réaliser une fonction: son arrêt (stop), son redémarrage (restart), ou pour demander son état (sense) en vue par exemple de son

initialisation (MVS ne démarrant d'abord qu'avec un seul processeur actif, les autres étant activés ensuite par ce processeur). Il s'agit dans ces cas-là de SIGP "direct". Nous avons vu aussi avec le "memory switch" qu'un processeur pouvait interrompre un autre processeur inactif pour lui signaler l'arrivée de travail (IBM emploie l'expression pittoresque de "shoulder tapping", tape sur l'épaule). Dans ce dernier cas, il s'agit de SIGP "à distance" (remote) dont le but est de provoquer dans le processeur visé, par création d'une interruption externe, I'exécution d'une routine particulière: distributeur pour le memory switch, traitement de recouvrement quand il s'agit d'un SIGP d'alarme ("emergency signal") émis par un processeur défaillant.

4) LE VERROUILLAGE

La mise en commun de la mémoire entre plusieurs processeurs, typique TCMP, rend accessible à tout processeur les programmes et les données des processus actifs. Encore faut-il veiller à ce que les processeurs ne fassent pas la même chose en même temps, ou pire, qu'ils ne fassent pas en même temps des opérations contradictoires (un processeur ignore ce qu'un autre fait). Autrement dit: il faut que l'accès à certaines ressources système critiques (représentées par divers blocs de contrôle) soit sérialisé. Il ne faut pas, par exemple, qu'un processeur modifie la file d'attente du distributeur pendant qu'un autre la lit, ou que plusieurs processeurs la modifient simultanément ; de même pour de nombreux blocs de contrôle, tels que l'UCB (unit control block) qui représente une unité périphérique. Le risque est au moins l'incohérence, au plus le blocage du système. On peut vouloir aussi sérialiser non pas l'accès à une ressource, mais l'appel d'une routine du système (par exemple, les algorithmes d'optimisation de SRM).

On convient d'associer à toute ressource critique un mot mémoire (appelé un verrou, "lock", on emploie aussi dans un sens proche le terme de "sémaphore"), à un endroit fixe, qui aura une valeur donnée si la ressource est disponible, une autre sinon. Tout processeur qui voudra accéder à la ressource représentée par le verrou devra examiner la valeur du verrou; si la ressource est libre, le processeur pourra "mettre le verrou" puis le retirer une fois l'opération sur la ressource effectuée. Il ne s'agit là que d'une convention, mais elle doit obligatoirement être respectée par tous les processeurs (et le code de MVS est fait pour cela). Si le verrou est mis (la ressource est donc indisponible) le processeur qui la requiert pourra soit attendre sa disponibilité en bouclant sans interruption sur le test du verrou (spin lock) soit inactiver le processus qui demande le lock pour passer à un autre travail (suspend lock).

Apparemment on n'a fait que reporter le problème des accès concurrents sur ces mots mémoire verrous, qui pourraient être lus ou modifiés simultanément par plusieurs processeurs qui les trouveraient libres en même temps. Il n'en est rien, car l'accès au verrou est effectué par une instruction qui lit et en même temps modifie (selon la valeur lue) le verrou, qui pendant ce temps-là est inaccessible aux autres processeurs. Il s'agit de l'instruction CS (compare and swap) qui porte sur un mot (l'instruction TS, test and set, rend le même service au niveau d'un octet). Voici le fonctionnement du CS:

```
CS  R1,R2,VERROU
```

Si VERROU contient la même valeur que R1 ("verrou libre") alors on transfère dans VERROU la valeur du registre R2 (on met le verrou). Le code condition du PSW vaut zéro. Sinon le verrou n'est pas modifié(verrou mis, indisponible), on récupère dans R1 la valeur de VERROU (ce qui permet d'identifier le détenteur du verrou) et le code condition est mis à 1. D'autres systèmes qui n'ont pas cette instruction à leur disposition dans le matériel réalisent la même fonction par logiciel ("sections critiques"). Les routines système emploient souvent l'instruction CS (ou CDS qui opère sur un double mot) pour manipuler des pointeurs ou incrémenter des compteurs, outre l'emploi des verrous.

Dans le cas d'un verrou "spin lock" qui rend un processeur ininterruptible, on a intérêt à ce que le verrou soit très rapidement rendu disponible par le processeur qui le détient, la boucle d'attente de libération du verrou étant improductive (même si on parle parfois d'"attente active"...). Par précaution, le processeur en boucle ouvre de temps en temps une "fenêtre" qui lui permettrait de traiter une interruption externe (de type "malfunction alert" ou "emergency signal") émise par le processeur détenteur du verrou qui deviendrait défaillant. Enfin, au bout d'un certain temps d'indisponibilité du verrou, le processeur avertit l'opérateur (voir le chapitre 11.A). Le verrou du type "spin lock" est plutôt utilisé pour les ressources système (il y a un peu plus de 15 sortes de "spin locks" en MVS). Le mot verrou contient dans ce cas un numéro identifiant le processeur.

Signalons qu'un verrou n'est pas forcément associé à une ressource système ("global lock"), mais qu'il peut être associé à une ressource au niveau d'un espace-adresse ("local lock"): on sérialise ainsi à ce niveau plusieurs tâches du même espace-adresse utilisant concurremment les processeurs. Cette sérialisation entre tâches est d'ailleurs opérationnelle même en configuration mono-processeur. Les verrous de type local sont aussi de type "suspend": en cas de non obtention, la tâche est inactivée.

Un autre problème doit cependant être résolu. En effet un "inter-blocage" peut survenir lorsque deux processeurs détiennent chacun un verrou différent et que chaque processeur attend d'obtenir le verrou que détient l'autre. Cette situation d'"étreinte fatale" (deadlock, interlock, deadly embrace) provoquerait une mise en attente indéfinie des deux processeurs (chacun attendant l'autre). Une solution (très sévère) serait de limiter à un seul le nombre de verrous que peut détenir un processeur. La solution choisie par IBM a été de créer une hiérarchie dans les verrous et de ne permettre l'obtention que de verrous supérieurs dans la hiérarchie aux verrous que l'on détient déjà. L'étreinte fatale ne peut ainsi survenir. C'est au composant gestionnaire de verrou (lock manager) qu'il revient de vérifier que le verrou demandé par un processeur est bien de hiérarchie supérieure aux verrous déjà détenus. Un lock se demande ou se libère par la macro-instruction SETLOCK (obtain ou release).

Voici quelques exemples de verrous utilisés en MVS (VM a aussi un système de verrous semblable):

verrou SALLOC: synchronise l'accès à la table des pages en mémoire réelle (en MVS-SP1);

verrou DISP: sérialise l'accès à la file du distributeur et à la table des espaces-adresses; les routines du distributeur peuvent ainsi s'exécuter simultanément (et sans risque) sur plusieurs processeurs;

verrou IOSUCB: un verrou par UCB représentant une unité d'entrée-sortie (pour sérialiser les E/S sur une unité);

verrou CPU: sérialise l'accès au processeur (permet en fait de se rendre ininterruptible). Ce verrou "spin" est différent des autres verrous: il y en a un par processeur (dans sa PSA), il peut être détenu par plusieurs tâches indépendamment des autres verrous "spin" qu'elles détiennent.

E) La sérialisation

Nous avons étudié, avec les verrous, le moyen de sérialiser l'accès à des ressources susceptibles d'être accédées simultanément par plusieurs processeurs Il existe un mécanisme qui sérialise des ressources susceptibles d'être partagées par plusieurs tâches sur une certaine période de temps (et pas forcément simultanément), l'exemple le plus courant étant le cas d'un fichier accédé en lecture et en écriture : l'absence de sérialisation permettrait des accès concurrents et ne garantirait pas la validité des données qu'il contiendrait. Ce mécanisme a un intérêt indépendamment du nombre de processeurs de la configuration, il est indispensable parce qu'une tâche qui a commencé à modifier une ressource peut être interrompue et voir son travail dégradé ou annulé par une autre tâche accédant à la même ressource.

On utilise un mécanisme logiciel semblable à celui des verrous, lui aussi tout à fait conventionnel, qui consiste à représenter une ressource par un nom symbolique. Un demandeur éventuel de la ressource émet la macro ENQUEUE sur la ressource (en fait sur le nom symbolique),en précisant s'il souhaite un accès partagé (shared) ou un accès exclusif (exclusive). L'accès partagé est accordé s'il n'y a pas déjà un détenteur exclusif de la ressource; l'accès exclusif est accordé si personne ne détient la ressource. Dans tous les cas, si l'accès ne peut être accordé, le demandeur est mis en attente et inactivé jusqu'à la libération de la ressource. La ressource une fois accordée est libérée par un DEQUEUE. Le mécanisme est plus souple que celui du verrou: il est disponible pour tous les demandeurs (système ou utilisateurs) et ne bloque pas le processeur par une boucle ininterrompue quand la ressource est indisponible. Il est utilisable sur de longues périodes de temps(par exemple tout le temps que dure la mise à jour d'un fichier). En revanche sa gestion par le système est assez lourde: il faut mémoriser toutes les demandes et tous les accès, comptabiliser le nombre de demandeurs et de détenteurs, respecter des conventions bien plus compliquées que pour les verrous. De plus, le mécanisme d'ENQUEUE/DEQUEUE ne permet pas d'éviter l'étreinte fatale entre plusieurs tâches: il n'y a pas de hiérarchie ni de limitations sur le

nombre d'ENQUEUEs qu'on peut détenir. Cependant cela ne bloque que les tâches en question (et non pas un processeur !) et l'étreinte fatale est facile à détecter. L'ENQUEUE peut être utilisé pour sérialiser l'accès à n'importe quelle ressource (fichier, zone mémoire, commande exécutable). Un type particulier d'ENQUEUE peut être utilisé par le système quand il s'agit d'une ressource critique dont la modification doit s'effectuer sans interruption sous peine de laisser cette ressource dans un état incohérent: il s'agit de l'ENQUEUE SMC (set must complete), qui se termine par un DEQUEUE RMC (reset must complete).

Le nom symbolique qui représente la ressource se décompose en deux parties:

- le "**qname**" (ou major name) d'au plus 8 caractères, qui est censé représenter un type de ressource: par exemple SYSDSN représente une allocation de fichier, SYSVSAM une allocation de fichier type VSAM, SPFEDIT un fichier utilisé sous l'éditeur de ISPF, SYSIKJUA un utilisateur connecté à TSO, SYSIEWLP un fichier en cours de mise à jour par l'éditeur de liens (link-edit), SYSVTOC la table des fichiers contenus par un volume disque (VTOC), etc.

- Ie "**rname**" (ou minor name) de 1 à 255 caractères qui identifie complètement la ressource (nom de fichier, de volume, etc.).

Il faut de plus, au moment où est demandé l'ENQUEUE, préciser la portée de cette sérialisation:

- à un niveau local (SCOPE = STEP): pour sérialiser les tâches d'un même espace-adresse (si bien qu'un même nom symbolique peut représenter des ressources différentes quand les demandeurs sont dans différents espaces-adresses) .

- au niveau du système (SCOPE = SYSTEM): la ressource est commune à tous les utilisateurs du système MVS.

- à un niveau inter-systèmes (SCOPE = SYSTEMS): la ressource est commune à plusieurs ordinateurs. Cette portée n'a de sens qu'avec un logiciel tel que GRS (global resource serialization) et une liaison physique entre ordinateurs qui permette de propager l'ENQUEUE: liaison canal-à-canal (adapter), disque partagé. Sinon elle équivaut à une portée "système" .

Le "RESERVE" est un ENQUEUE particulier qui "réserve" au système émetteur un volume disque tout entier partagé entre plusieurs ordinateurs: on empêche ainsi tout accès physique à ce disque par les autres systèmes.

F) La synchronisation

Un système MVS pouvant supporter de nombreuses tâches s'exécutant en concurrence, il est nécessaire de disposer d'un mécanisme qui assure la synchronisation de tâches interdépendantes. Un exemple typique nous est fourni par les entrées-sorties: une tâche qui a lancé un ordre d'entrée-sortie ne pourra traiter les données demandées avant d'être assurée que les périphériques et les canaux aient bien amené en mémoire les informations requises. De même une tâche peut déclencher un tri de fichier s'exécutant alors de manière asynchrone et poursuivre son exécution, jusqu'à l'instant où elle aura besoin des données triées et devra donc s'assurer que le tri est terminé, et attendre sa fin si ce n'est pas le cas.

On emploie un dispositif analogue à celui du verrou: un mot mémoire, positionné initialement à zéro, l'ECB (event control block), qui représente l'événement dont on attend la fin. Le premier bit du mot(wait bit) indique, s'il est positionné, qu'une tâche s'est mise en attente de l'événement; le bit suivant (complete bit) indique, s'il est positionné, que l'événement attendu est survenu. Le reste du mot peut être utilisé par la tâche qui "réalise l'événement" pour passer un code retour à la tâche en attente. On aura donc deux opérations possibles qui mettent en jeu le distributeur:
* WAIT sur l'ECB: le système est informé que la tâche ne peut continuer son traitement tant que l'événement représenté par l'ECB ne sera pas accompli. Une tâche peut opérer aussi un WAIT sur plusieurs ; elle ne reprendra alors le contrôle que lorsque tous les événements seront survenus.

* POST sur l'ECB: signale au système l'arrivée de l'événement représenté par l'ECB. Le distributeur peut reprendre en compte la tâche qui a effectué le WAIT. Le POST peut passer un code retour dans l'ECB pour signaler une erreur à la tâche en attente. Dans certains cas on utilise un "quick POST" pour mettre directement à 1, par l'instruction COMPARE and SWAP, le bit "complete"; on évite ainsi le passage par les routines du distributeur, mais une telle opération n'est possible que si le bit de WAIT n'est pas positionné, sans quoi la tâche en WAIT ne serait jamais informée de l'arrivée de l'événement. Si le bit de WAIT est à 1, on doit recourir à la macro POST.

Le POST peut être destiné à une autre tâche dans un espace-adresse différent (cross-memory POST); on doit préciser dans la macro l'ASCB de l'espace-adresse visé. Les routines SRB utilisent très fréquemment ce type de POST.

MVS fournit une macro-instruction EVENTS plus souple que la macro WAIT: avec EVENTS une tâche peut se mettre en attente de plusieurs événements et reprendre le contrôle dès qu'un événement est survenu. La tâche peut ainsi savoir quels événements sont arrivés et dans quel ordre.
Il existe en MVS 3 moyens classiques de faire exécuter des programmes de façon asynchrone:

- le SRB: moyen facile à mettre en oeuvre (si on est autorisé) mais assez limité;
- le DIE, routine qui se déclenche dans un intervalle de temps donné (voir paragraphe suivant);
- les routines "exit asynchrone", qui permettent de forcer l'exécution d'un programme sous un TCB donné. La mise en oeuvre nécessite 3 temps:
    - création d'un IRB (interruption request block) qui précise notamment le nouveau PSW et le point d'entrée de la routine;
    - scheduling d'un IQE linterrupt queue element) qui donne l'adresse du TCB que l'on vise;
    - l'IRB est rattaché à la tâche et la routine sera exécutée dès que le TCB sera dispatché par le distributeur.

G) Le contrôle du temps

Le contrôle du temps (timer supervision) est une fonction importante qui permet de disposer d'une horloge, d'entreprendre des actions à une heure donnée ou après un certain temps: algorithmes réguliers de SRM, contrôle du temps alloué à un travail, planification à l'avance de travaux, etc. Les moyens matériels sont l'horloge "TOD clock", le "clock comparator" et le "CPU timer" (voir chapitre 1). La mesure du temps est importante pour la comptabilisation des consommations de ressources (SMF), le suivi de l'activité et des événements (SYSLOG), l'utilisation des fichiers, etc.

Il y a habituellement deux "temps" de référence : celui qu'IBM appelle le temps "GMT" (qu'il conviendrait de désigner plutôt par "UTC", temps universel coordonné, référence pour les fuseaux horaires), et l'heure locale, obtenue à partir du temps GMT par décalage d'un certain nombre d'heures selon la position du site par rapport au méridien de Greenwich (paramètre PARMTZ de la SYSGEN ou membre CLOCKxx de la PARMLIB). Le temps local est le seul qui soit utilisé couramment et qui serve de référence à tous les utilisateurs de l'informatique à un endroit donné ; le temps GMT fournit une base de comparaison pour des travaux envoyés vers un autre site distant (par NJE) sur un fuseau horaire différent (cas fréquent aux Etats-Unis).

Le registre TOD clock utilisé par le système est initialisé à l'IPL par le pupitreur ou automatiquement à partir d'une horloge matérielle qui contient le temps GMT. Le temps local peut être modifié à tout moment par commande MVS (SET CLOCK=..., TIME=...), tandis que le temps GMT ne peut être modifié que par IPL. En général, avancer l'heure locale (cas de l'heure d'été en France depuis 1976) ne pose pas de problème ; la reculer (cas de l'heure d'hiver) peut entraîner quelques anomalies dans certains composants à cause d'un horodatage en double (SMF, IMS, JES).

Les macros TIME, STIMER et TTIMER permettent respectivement d'obtenir la date et l'heure, de positionner un chronomètre et de le tester. Les fonctions de timing reposent sur la gestion de blocs de contrôle TQE (timer queue element) représentant les requêtes de contrôle du temps, et mis dans une file d'attente où ils sont classés dans l'ordre d'expiration des intervalles, de sorte que celui qui est en tête correspond à l'intervalle d'expiration le plus petit. Le CPU timer régulièrement décrémenté provoque une interruption externe quand l'intervalle est écoulé; l'interruption est gérée par le SLIH affecté aux fonctions de timing, qui retire de la file le premier TQE et déclenche l'action prévue à l'expiration.

Une fonction intéressante de DIE (disabled interrupt exit) permet de donner le contrôle à une routine après un intervalle de temps donné; la routine est appelée directement par le SLIH et fonctionne en mode superviseur, clé 0 et ininterruptible. L'utilisateur (autorisé) de cette fonction doit construire lui-même un TQE et préciser sa demande par un "set DIE". On peut de cette façon entreprendre de façon très précise certaines actions à des intervalles de temps réguliers. Le DIE est utilisé par la plupart des moniteurs (RMF par exemple).

Les applications pratiques des fonctions de timing sont nombreuses : serveurs se "réveillant" à intervalles réguliers pour examiner si une demande leur est adressée; commandes passées à une heure donnée (possible avec JES2); planification de travaux à heure fixe.

H) SYSPLEX

Sysplex est un composant de MVS/ESA SP4 qui permet un couplage lâche de systèmes MVS (jusqu'à 16) connectés par canaux à fibre optique ESCON ou par unités 3088 et synchronisés par une horloge système unique (une unité externe ETR, ou "Sysplex timer"). Les bénéfices escomptés pour les plus grands centres sont les suivants :

* une augmentation de la puissance totale au-delà du millier de MIPS, impossible à obtenir avec un seul ordinateur dans l'état actuel de la technique ;

* une amélioration de la disponibilité des systèmes (reprise sur un autre MVS d'une application défaillante) ;

* un suivi des opérations plus simple (plusieurs MVS pupitrés depuis une seule console).

Le composant XCF (cross-system coupling facility) de MVS gère les connexions inter-systèmes pour fournir tous les services de Sysplex, exploités par des produits tels que GRS, l'APPC/MVS, TSO, OPC/ESA ou CICS (avec XRF).

Le membre COUPLExx de la PARMLIB paramètre XCF en imposant les adresses des CTCs (voies XCF), des "classes" de transport, etc. ; de nouvelles commandes MVS permettent une gestion dynamique (SETXCF, VARY XCF). Le membre CONSOLxx peut être adapté pour profiter du fait que
toute console du complexe peut accéder à tout système.

Le futur de Sysplex tel que le supputent certains observateurs est une image logique unique du centre de traitement et l'accès transparent à une très grande puissance de traitement, avec une possibilité accrue de "tolérance de pannes". Les membres du Sysplex pourront partager des bases de données en mise à jour ; toute défaillance d'un ordinateur sera palliée par basculement sur une autre machine (hot standby mainframe) ; la charge de traitement sera répartie entre les systèmes beaucoup plus finement qu'en organisation lâche de type JES3, grâce au partage de la MAP qui permettra par swap aux différents MVS de mettre en commun la "work queue".

3. LA RESSOURCE MEMOIRE

A) Historique

La façon d'utiliser la mémoire centrale des ordinateurs a beaucoup évolué dans le temps. Le problème à résoudre a toujours été le suivant: comment utiliser au mieux la mémoire dont on dispose (limitée et chère) étant donné le nombre toujours croissant d'utilisateurs, et sachant que chacun a besoin d'un espace mémoire linéaire pour installer en contigu ses données et ses programmes ? La solution employée par les systèmes d'exploitation modernes est la mémoire virtuelle (qui permet de se libérer de la limitation de la taille de la mémoire centrale et de la nécessité d'y avoir des zones de mémoire contiguës pour chaque travail) et son corollaire la pagination dynamique: la mémoire est découpée en "pages" de taille identique allouées aux différents processus mais gérées dynamiquement par le système (qui peut les transférer sur disque au besoin, ce qui fait que les zones référencées par une instruction ne sont pas forcément présentes en mémoire centrale).

MVS provient de l'OS (Operating System) des années 60 après différentes mutations... Le primitif OS-PCP (programme de contrôle primaire) ne supportait qu'un seul utilisateur à l'instar du DOS-PC. Avec l'OS-MFT (multiprogrammation avec des tâches fixes) la mémoire réelle pouvait être divisée jusqu'en 15 "partitions" fixes pour 15 utilisateurs. L'adresse et la taille de chaque partition étaient fixes et déterminées au démarrage du système, ce qui n'optimisait pas l'occupation de la mémoire, un petit travail pouvant être amené à s'exécuter dans une partition trop grande pour lui (fragmentation interne). Une légère amélioration fut apportée par OS-MVT (multiprogrammation avec des tâches variables) avec des partitions de taille variable, appelées "régions": l'adresse d'une région était déterminée au début de l'exécution du travail et la taille d'une région était variable. Les régions n'étaient cependant pas translatables en mémoire, ce qui aurait permis, comme dans certains systèmes, de récupérer, par "tassement" des régions, les trous d'espace libre laissés par les travaux terminés (fragmentation externe). Les tailles de mémoire centrale de cette époque seraient aujourd'hui considérées comme ridiculement petites (64K sur les 360). Avec OS-VS1 (OS virtual storage version 1), obtenu en ajoutant un superviseur de pagination à OS-MFT, on commença à se libérer de la limitation de la mémoire centrale: une mémoire virtuelle unique de 16 Méga-octets pouvait être divisée en 15 partitions pour 15 utilisateurs simultanés. La pagination (avec des pages de 2K) était supportée par une mémoire auxiliaire avec une correspondance figée 1 page virtuelle/1 case sur disque. Cependant il y avait toujours cette limitation sur le nombre d'utilisateurs simultanément en mémoire, due à la nécessité d'assurer à chacun un espace mémoire d'un seul tenant (le système DOS-VSE, basé lui aussi sur des partitions de mémoire, rencontre toujours ce genre de limitation). OS-VS1 appliquait le principe de la mémoire virtuelle à OS-MFT, OS-VS2 release 1 (appelé aussi SVS, single virtual storage) l'appliquait à OS-MVT.

Enfin apparut OS-VS2 (release 2), qui était déjà MVS, et qui donnait cette fois à chaque utilisateur (le nombre d'utilisateurs devenant quasi illimité) un espace virtuel de 16 Méga-octets (avec une taille de page de 4K). Les versions MVS-SP2 et SP3 ont encore accru cette taille de mémoire virtuelle. En effet les 16 Méga-octets initiaux n'étaient pas disponibles en un seul morceau: une partie de plus en plus grande était occupée par les zones système communes; les zones privées étaient utilisées par des applications elles aussi de plus en plus grandes, avec le développement de systèmes "online" gérant de grandes bases de données et des centaines de terminaux connectés. Des artifices de programmation furent inventés: avec la segmentation (overlay) on ne gardait en mémoire (sur décision préalable du programmeur) que les portions de programme les plus actives. Enfin on dut accroître substantiellement l'espace virtuel (et corrélativement la mémoire réelle qui le supporte). Les figures 1 et 2 montrent l'évolution qui a conduit à MVS depuis les années 60.

B) La mémoire "réelle"

On peut établir une hiérarchie entre les différents supports-mémoire de l'ordinateur en partant des plus rapides, qui sont aussi les plus limités en taille, jusqu'aux plus lents, plus amples (et moins chers). MVS profite pleinement des différents supports, en réservant les supports les plus rapides aux zones de mémoire les plus accédées.

Une notion qu'on rencontre fréquemment est celle de "cache" ou d'antémémoire: I'idée est d'extraire d'un support-mémoire les zones les plus utilisées pour les conserver sur un support plus rapide. On améliore ainsi les performances, le prix en est la gestion de la duplication des zones et la détection des zones les plus accédées (ou l'anticipation). On peut ainsi "cacher" la mémoire centrale ou la mémoire disque.

Chaque processeur dispose d'une antémémoire (HSB, high speed buffer) de quelques dizaines de Koctets qui contient un sous-ensemble des zones les plus accédées de la mémoire centrale. Ceci implique une communication entre les processeurs quand il advient que leurs HSBs référencent les mêmes zones...

Plus bas dans la hiérarchie des mémoires se trouve la mémoire centrale de l'ordinateur et les tampons TLB des processeurs, utilisés pour les traductions d'adresse (ce sont aussi des antémémoires). Les temps d'accès à la mémoire centrale sont communément de 80 nanosecondes (sur les 3090) et diminueront encore dans l'avenir (65 ns sur les ES/9000).

Un peu plus bas, la mémoire d'arrière-plan (MAP, extended storage) dont sont pourvus certains ordinateurs (3090, ES/9000). D'une taille pouvant aller jusqu'à 2 Méga-octets, elle est utilisée pour la pagination, comme intermédiaire avant le disque, ou pour stocker des données en ESA. Les temps d'accès à la MAP sont à peine supérieurs à ceux de la mémoire centrale (120 nanosecondes). Pour cette raison la MAP depuis son apparition a suscité un certain intérêt et IBM en a tiré beaucoup d'applications en la substituant tantôt à la mémoire centrale (toujours trop limitée !) tantôt au support magnétique (toujours trop lent !). On ne peut cependant exécuter un programme directement depuis la MAP, le transfert en mémoire centrale est toujours indispensable...

Enfin on trouve les disques (qui peuvent avoir une antémémoire, selon le type de contrôleur 3880 ou 3990), puis les bandes et les autres supports. Le temps d'accès disque est de l'ordre de quelques dizaines de millisecondes ou de quelques millisecondes dans le meilleur des cas.

Les capacités de ces différentes mémoires s'échelonnent entre plusieurs dizaines ou centaines de millions d'octets pour la mémoire principale et quelques milliards d'octets pour certains disques.

C) La mémoire virtuelle: notions de base

1) PAGE ET CLE; SUBPOOL

L'unité de base de la mémoire virtuelle est la **page**, bloc de mémoire de 4K. La page virtuelle est supportée par la mémoire réelle, découpée par construction en blocs de 4K (cadres de page: "frame") ainsi que par la mémoire auxiliaire (sur disque) découpée en cases (slots) de 4K elle aussi. Cette taille de 4K résulte d'un compromis: une taille de page trop grande risque d'entraîner un gâchis d'espace à l'intérieur de la page (fragmentation interne), une taille trop petite oblige à gérer de très nombreuses et grosses tables de pages pour la correspondance pages virtuelles/cadres de page réels. D'autres systèmes utilisent une taille de 2K (c'est le cas de OS/VS1, du DOS et des S/36).

A chaque cadre de page est associée une clé de protection mémoire (non adressable) d'un octet qui est constituée:

* d'une clé de 4 bits de valeur 0 à 15;
* d'un bit de protection lecture (fetch protect bit);
* d'un bit signalant que la page a été lue (reference bit);
* d'un bit indiquant si la page a été modifiée (change bit);

On ne permet l'accès en écriture à une page mémoire que si le PSW du processus qui effectue l'accès a une clé égale à la clé de 4 bits du cadre de page. L'accès en lecture est permis dans tous les cas, sauf si le bit de protection lecture est mis et que la clé du PSW diffère de la clé mémoire. Le système dispose de la clé particulière de valeur 0 qui lui ouvre l'accès en lecture ou en écriture à tous les cadres. La clé du PSW peut être modifiée (macro MODESET par exemple). La clé d'un cadre en revanche est assignée lors de son allocation par

le système et prend la valeur de la clé du PSW, pour ne plus être changée dans la "vie" du travail (sauf cas très particuliers avec l'emploi d'une macro CHANGKEY).

La notion de clé de protection mémoire est une survivance (utile) de l'OS-VS1: chacun des 15 utilisateurs du même (et unique) espace virtuel de 16 Méga-octets avait une clé différente (de valeur 1 à 15): il se protégeait ainsi du risque de voir ses données mémoire malencontreusement modifiées par un autre utilisateur (protection "verticale"). Le superviseur avait la clé privilégiée de valeur 0. Avec OS-VS2 et MVS, ce risque n'existait normalement plus: à un instant donné un seul espace virtuel d'adresses est actif sur un processeur, seules ses pages virtuelles lui sont accessibles et il ne peut modifier les données dans la mémoire virtuelle d'un autre: la mémoire virtuelle entraîne une protection "horizontale" entre utilisateurs. Cependant l'espace-adresse virtuel contient des données et programmes systèmes communs à tous les espaces-adresses (voir le paragraphe D.), et ces zones, accessibles à tous, sont en un seul exemplaire en mémoire réelle. On évite le risque de modification erronée de ces zones par la protection verticale par clés. La valeur des clés est associée aux grands composants du système:

- 0 pour le système MVS
- 1 pour le scheduler et JES
- 2 pour VSPC (virtual storage personal computing)
- 3 et 4 sont "réservés" (parfois utilisés par certains composants)
- 5 pour la gestion des données
- 6 pour la gestion du réseau de communications (TCAM, VTAM)
- 7 pour le gestionnaire de bases de données IMS
- 8 est la clé par défaut de tous les utilisateurs en mémoire virtuelle (une seule valeur suffit pour tous à cause de la protection horizontale)
- les valeurs de 9 à 15 sont réservées aux utilisateurs en mémoire réelle (V = R, pas de traduction d'adresse). Le V = R est pratiquement inutilisé en MVS, étant donné la nécessité de "dégager" pour l'utilisateur V = R un espace mémoire contigu (ceci aux dépens des autres utilisateurs).

L'intérêt des bits "page lue" et "page modifiée", qui complètent la clé, sera vu plus loin.

Une page virtuelle peut ne pas être en mémoire réelle (pas de cadre correspondant): elle a pu être "volée" par le système (page-out) pour désencombrer la mémoire réelle. Tout accès à cette page provoquera une interruption programme de type "faute de page" qui amènera la page en mémoire (page-in) à partir de la copie en mémoire auxiliaire (dans une case). MVS peut s'interdire le vol d'une page s'il a "fixé" la page en mémoire (macro PGFIX); plusieurs composants peuvent fixer la même page: un compteur est maintenu. Il y a une autre manière de gérer la mémoire centrale, qui consiste à vider (swap) complètement sur disque les zones privées d'un espace-adresse.

Les pages sont regroupées pour MVS en "subpools". Un subpool est un ensemble de pages de mêmes caractéristiques, telles que: une clé de protection commune, un emplacement sous ou sur la "barre" des 16 Méga-octets, la faculté d'être ou non paginable, vidable, à usage système ou utilisateur. Il n'y a pas de notion de contiguïté ni de taille limite. Un subpool est identifié par un numéro de 0 à 255 (la plage utilisateur est 0-127, 0 par défaut). Il s'agit uniquement d'un regroupement logique géré par MVS (alors que page et clé sont liés à l'architecture).

Un découpage plus fin permet d'avoir dans les subpools des "pools" de mémoire; le programmeur peut construire ou détruire un "pool", obtenir du pool ou lui rendre des blocs de mémoire, les "cellules". La figure 3 représente un pool de cellules créé dans le subpool 0. Ceci permet d'éviter la pénalisation qui résulte de nombreuses demandes (par GETMAIN) de petits blocs de mémoire .

Trois composants gèrent la mémoire en MVS: RSM (real storage manager) qui s'occupe de la mémoire réelle: il gère les tables de page et les mouvements de pages de mémoire; ASM (auxiliary storage manager) gère la mémoire auxiliaire: il amène en mémoire le contenu des cases (slots) lors d'un "page-in" ou copie une page sur une case pour un "page-out"; VSM (virtual storage manager) gère la mémoire virtuelle: il alloue ou désalloue l'espace virtuel (macros GETMAIN ou FREEMAIN) et garde trace des zones allouées et disponibles dans les "subpools" des espaces-adresses.

2) LA TRADUCTION D'ADRESSE

La traduction adresse virtuelle -> adresse réelle assurée par le DAT repose sur des tables de correspondance propres à chaque espace-adresse (chaque poste décrivant le cadre de page réelle associée à une page virtuelle de l'espace-adresse). Le format d'un tel poste (PGTE, page-table entry) est le suivant en architecture 370:

* 12 bits (page-frame real address) donnant l'adresse du cadre en mémoire réelle (les 12 bits manquants pour constituer l'adresse réelle à 24 bits d'une zone de mémoire virtuelle représentent le déplacement dans la page et découlent immédiatement des 12 derniers bits de l'adresse virtuelle);

* 1 bit (page-invalid bit) qui vaut 0 si le cadre correspond bien à une page virtuelle, 1 sinon (la page virtuelle a été "volée" par le système et transférée sur disque, le cadre de page est donc disponible);

* les 2 bits suivants peuvent compléter les 12 premiers pour donner finalement une adresse réelle à 26 bits au lieu de 24 (car MVS-SP1.3 peut supporter une mémoire réelle jusqu'à 64 Méga-octets, au lieu des 16 Méga-octets résultant d'une adresse réelle à 24 bits).

En architecture 370-XA, le format du poste est un peu différent:

* un bit à 0 inutilisé;

* 19 bits donnant l'adresse réelle du cadre (les 12 bits du déplacement venant les compléter pour donner l'adresse réelle à 31 bits, MVS-XA pouvant supporter des mémoires réelles de 2 Giga-octets = 2 puissance 31 octets);

* un bit à 0 inutilisé;

* le bit "page invalide";

* un bit (page-protection bit) qui empêche les accès en écriture sur le cadre s'il vaut 1. Une telle protection au niveau cadre de page n'existe pas en 370 pur. Elle est utile pour les programmes systèmes installés une fois pour toutes (pour ne plus être modifiés) au démarrage de MVS;

* les bits restants sont disponibles.

Chaque poste de table mesure 2 octets (en 370) ou 4 octets (en 370-XA). Un calcul élémentaire montre qu'une telle table de pages pour toutes les pages possibles d'un espace-adresse, occuperait un grand espace de mémoire, avec une contrainte de contiguïté et de non-pagination. Aussi utilise-t-on un second niveau de pagination avec le segment (ou hyperpage), constitué de 16 pages en 370 ou de 256 en 370-XA. On maintient une table des segments de l'espace-adresse, chaque poste de 4 octets (SGTE) contenant l'adresse réelle de la table de toutes les pages du segment ("page table origin"). Un bit "segment invalide" existe comme pour les pages: il indique que la table des pages du segment n'est pas encore construite (ou n'est pas en mémoire). Une table de segments comporte 256 postes en 370 ou 2048 en 370-XA (autant que de segments et donc de tables de pages pour l'espace-adresse), ce qui porte sa taille à seulement 1024 ou 8192 octets. Chaque table de pages du segment est ramenée à 16 ou
256 postes. Seule la table des segments doit être en mémoire. Une table de pages est construite lors de la première référence à son contenu (faute de page de la première référence), et non pas lors du GETMAIN demandeur d'espace virtuel.

La traduction d'adresse se fait donc en passant par les deux niveaux. Une adresse virtuelle de 24 ou 31 bits est décomposée en 3 parties:

- les 8 premiers bits (370) ou les 11 premiers (370-XA) constituent le numéro de segment (S);
- les 4 suivants (370) ou les 8 suivants (370-XA) donnent le numéro de page dans le segment (P);
- les 12 derniers donnent le déplacement dans la page virtuelle (D).

Le poste numéro (S) de la table des segments nous fournit l'adresse réelle de la table des pages du segment (S). Dans cette table le poste numéro (P) nous donne l'adresse réelle de la page. L'information référencée sera dans cette page au déplacement (D). Voir la figure 4.

Une table des pages est de longueur variable et, à l'inverse de la table des segments, elle n'est pas forcément en mémoire quand l'espace-adresse est actif. Quand cela arrive, le DAT ne peut continuer sa traduction, et une interruption "faute de segment" survient. La table des pages absente est soit construite soit ramenée en mémoire si elle a été elle-même paginée. Une autre table (SLT, second level table) est nécessaire pour cela (elle conserve les clés mémoire des pages contenant ces tables de pages). Une "faute de page" survient de même quand le bit "page invalide" (page volée) est positionné dans un poste de la table des pages, ou quand le poste n'est pas encore créé dans la table.

Une page contenant une table de pages est traitée par MVS de la même façon que toute autre page. Cependant, les tables de pages courantes, qui sont paginables (lower page tables), sont contenues dans des pages elles-mêmes décrites dans des tables de pages particulières (higher page tables) construites les premières lors d'une faute de segment. Ces dernières ne peuvent être paginées vers la mémoire auxiliaire (sauf si elles sont invalides). Ainsi le vol d'une page contenant une table de page de premier niveau ne pose pas de problème. La figure 5 décrit ces niveaux entre les tables de pages. De même les tables des zones communes à tous les utilisateurs restent toujours en mémoire.

Le registre de contrôle 1 (CR 1), appelé aussi STOR (segment table origin register) pointe sur la table des segments propre à l'espace-adresse. Ainsi se fait la distinction entre les différents espaces de mémoire virtuelle, une même adresse virtuelle pouvant correspondre à différentes zones dans différents espaces-adresses (sauf s'il s'agit des zones dites communes). Quand le processeur traite un espace-adresse, il suffit que le CR1 soit correctement initialisé pour que les adresses virtuelles soient automatiquement décodées.

La traduction d'adresse par le DAT avec ces tables de segments et de pages n'est pas spécifique à MVS, mais est disponible sur toutes les machines IBM 370.

Une telle recherche avec 2 niveaux de tables est relativement lourde (plusieurs accès mémoire sont nécessaires avant d'accéder à la donnée requise), et risquerait d'être inutilement répétée plusieurs fois de suite (pour donner comme résultat la même page réelle) pour des zones virtuelles "voisines" (situées dans la même page). Aussi toute traduction d'adresse est stockée dans une autre "table", un tampon matériel de petite taille (selon les machines, depuis 8 postes jusqu'à une centaine tout au plus), le TLB (translation lookaside buffer), qui est une mémoire associative (adressable par son contenu) d'accès très rapide (on n'y effectue pas de recherche séquentielle, tous les postes peuvent être comparés en même temps) qui fait correspondre un numéro de segment et de page avec l'adresse du cadre correspondant. La traduction d'adresse par le DAT n'est effective que si aucune entrée n'existe dans le TLB (TLB miss); dans ce cas le TLB est remis à jour. Le TLB est nettoyé de ses postes obsolètes par une instruction PTLB (purge TLB).

Avec MVS/ESA le mécanisme de traduction se complique avec l'apparition d'un troisième niveau de tables pour accéder aux nouveaux espaces de données. Avec le mode AR (access register), les registres généraux référencent la mémoire virtuelle des espaces-adresses identifiés par les registres AR associés aux registres généraux. Ainsi, dans une instruction L R1,0(R3) exécutée depuis un espace-adresse EA1, le registre AR3 détermine quel espace-adresse EA2 est référencé, par consultation d'une "access list", propre à EA1 ou à la tâche en cours d'exécution, qui dresse la liste des espaces-adresses ou dataspaces accessibles; le registre général 3 donne l'adresse virtuelle de la zone visée dans EA2. Il y a donc une traduction supplémentaire (access register translation), dont le but est de trouver la bonne table de segments, celle de EA2, avant de procéder à la traduction habituelle par le DAT.

D) Les zones virtuelles

Un espace-adresse est l'espace virtuellement alloué à un utilisateur; sa taille est de 16 Méga-octets en 370 ou de 2 Giga-octets en 370-XA. Cet espace contiendra tout ce qui est nécessaire au déroulement du travail: les données et programmes "utilisateurs" ainsi que les données et programmes de service du système. L'espace-adresse se composera de différentes zones qui auront des rôles particuliers et une évolution différente dans la

durée de vie du travail. Certaines sont propres à ce travail (zones privées), d'autres seront communes et partagées par tous. Examinons d'abord les zones privées.

La "**system region**" est formée par les 4 premières pages de la zone privée: elle est utilisée par la RCT (region control task), tâche de plus haut niveau dans un espace-adresse, qui contrôle le vidage du travail.

La **LSQA** (local system queue area) contient les blocs de contrôle (TCBs, blocs décrivant les subpools, etc.) et les tables (tables de segments et de page par exemple) propres à l'espace-adresse. Elle peut être vidée (avec l'espace-adresse) mais non paginée (les pages sont fixées en mémoire réelle). La seule exception, et elle est largement justifiée en XA, est pour les tables de pages décrivant la zone privée au-dessus de la barre des 16 Méga-octets (extended private). En XA, on met dans une seule page une table de pages (avec 256 entrées PGTEs et XPTEs). L'ensemble des tables de pages de la zone privée étendue pouvant occuper jusqu'à 2048 pages de 4K, soit 8 Méga-octets, on a préféré rendre cet espace paginable en "extended LSQA". Les tables des zones privées en dessous de la barre des 16 Méga-octets ne sont pas paginables, le gain ayant été jugé négligeable.

La **SWA** (scheduler work area) contient des informations qui proviennent du langage de contrôle (JCL) du travail après son interprétation: il s'agit surtout d'informations relatives aux allocations de fichiers (cartes DD du JCL ou allocations dynamiques). La SWA est vidable et paginable et contient des pages de clé 1.

Les subpools 229 et 230 (**AUK**, authorized user key), paginables, contiennent des données propres à l'utilisateur (zones de travail pour JES, blocs DEBs des fichiers ouverts), qui peuvent être protégées en lecture par le "fetch bit" avec la clé de l'utilisateur. Les 3 zones (LSQA, SWA, AUK) sont allouées selon le besoin dans la zone privée en partant du haut de cette zone et varient en taille.

La zone proprement réservée à l'utilisateur (**user region**) s'étend entre la "system region" et les zones LSQA/SWA/AUK. Sa taille maximale, variable d'une installation à une autre, dépend de la taille des zones communes (en MVS-SP1 il n'était pas rare qu'elle ne soit que de 4 ou 5 Méga-octets sur les 16 de l'espace-adresse...). Elle dépend aussi de la place que prennent LSQA, SWA et AUK (variable d'un travail à l'autre). On peut la limiter par le paramètre REGION du langage de contrôle. Elle s'alloue à partir des adresses basses, ce qui fait qu'en cas de demande mémoire trop importante (et de paramètre REGION trop grand), il y a risque de "collision" avec la LSQA qui s'alloue à partir des adresses hautes (un tel genre d'incident existe aussi en VM).

Les zones communes figurent dans tous les espaces-adresses et sont en un seul exemplaire en mémoire réelle. Elles sont "alimentées" par le système ou par les utilisateurs quand il est nécessaire de faire connaître certaines informations à tous les travaux qui pourraient en avoir besoin. Certaines zones peuvent être paginables, mais aucune n'est "vidable" dans son entier.

La **PSA** (prefixed save area, ou prefixed storage area), évoquée au chapitre précédent, contient les données propres à un processeur. Elle occupe les 4 premiers K de la mémoire virtuelle, et aussi de la mémoire réelle pour un mono-processeur (pour les multi-processeurs le registre préfixe dirige chaque processeur vers une PSA différente, qui n'est pas dans les 4 K de la mémoire basse, mais en SQA; le PSA de la mémoire basse sert à initialiser les PSAs des différents processeurs). Les 512 premiers octets sont protégés vis-à-vis du logiciel pour n'être utilisables que par le matériel.

Le **noyau** (nucleus) contient les routines superviseur de base (traitement des interruptions, dispatcheur, gestion des entrées-sorties, de la pagination, programmes de reprise en cas de panne), la table des cadres de page (PFTEs) dont nous verrons la fonction plus loin, les blocs de contrôle d'accès aux librairies système, les UCBs des unités périphériques et toutes les "primitives", fonctions de base s'exécutant en mode superviseur, invoquées pour demander tel ou tel service. Le terme de superviseur est réservé au composant qui gère les tâches. Le noyau contient aussi la CVT (communication vector table), bloc de contrôle fondamental en MVS car il contient les pointeurs vers presque tous les autres blocs de contrôle du système. Le noyau est de clé 0 et constamment résident en mémoire (non paginable). Il ne s'agit pas là d'un vague souci de performance, mais les fonctions du noyau qui s'exécutent en mode ininterruptible (disabled) ne peuvent par là-même supporter une faute de page (qui
est une interruption). Une partie du noyau est protégée en écriture (read-only nucleus).

En MVS-SP1 le noyau est en V = R (adresse virtuelle = adresse réelle), ce qui le dispense du mécanisme de traduction d'adresse. Au contraire en MVS-XA, il est en V = F, fixé en mémoire mais à des adresses virtuelles (il occupe donc des cadres de page non forcément contigus); il a donc besoin du mécanisme de traduction d'adresse (DAT-on nucleus; par précaution, il existe un noyau qui fonctionne sans le DAT, DAT-off nucleus, et qui n'est pas en mémoire virtuelle). Le noyau, en XA, est placé à cheval (juste en dessous et juste en dessus) de la "barre" des 16 Méga-octets, alors qu'en SP il est aux adresses basses. Sa taille est variable d'une installation à l'autre, et dépend de divers facteurs: la taille de la mémoire réelle (à cause de la table des cadres), le nombre d'unités périphériques (blocs UCB), la taille des routines SVCs qui y figurent.

La **PLPA** (pageable link pack area) contient la plupart des routines du système: SVCs, méthodes d'accès aux fichiers, etc. Les pages de la PLPA ne sont pas modifiables (nous verrons ce que cela implique quant à l'écriture des programmes en PLPA); elles peuvent être volées par le système en cas d'inutilisation sans qu'il ait besoin à chaque fois d'en maintenir une copie en mémoire auxiliaire (pas de "page-out", la copie est faite une fois pour toutes au démarrage de MVS). La PLPA est créée par chargement au démarrage des modules du fichier système SYS1.LPALIB et des fichiers "concaténés" (liste LPALSTxx en XA); sa taille ne varie plus ensuite. Ce chargement, étant assez long, est accéléré quand on prend au démarrage les cases des modules conservées sur les disques de pagination (warm start, quick start) plutBt que sur les LPAlBs (cold start). La taille de la PLPA est uniquement fonction de la taille des LPAlBs. Les autres systèmes 370 ont l'équivalent de la PLPA: SVA en DOS, segments partagés en VM.

La **FLPA** (fixed link pack area) contient des modules qui pourraient être en PLPA, mais qu'on veut fixer en mémoire pour des raisons de temps de réponse. On évite le risque de faute de page qu'encourent normalement les modules de la PLPA. La FLPA est considérée en MVS-SP1 comme une extension du noyau (dont elle fait partie) ou comme une extension de la PLPA en MVS-XA.

La **MLPA** (modified LPA) est utilisée pour tester des modules avant de les mettre en PLPA (un module en MLPA est pris en compte de préférence à une autre version du même module en PLPA quand on l'invoque). Comme pour la PLPA et la FLPA, les pages de la MLPA ne sont pas modifiables, sauf si on le demande (paramètre MLPA = NOPROT dans la PARMLIB en XA). La FLPA et la MLPA sont des zones optionnelles, de taille variable selon les listes de modules à y charger précisées en PARMLIB (IEALPAxx pour la MLPA, IEAFlXxx pour la FLPA). Elles ne sont chargées au démarrage que si on le demande explicitement, elles ne sont pas conservées d'un démarrage à l'autre.

La **CSA** (common service area) est utilisée par le système et tous les utilisateurs pour stocker des données qui doivent être accessibles à tous. Les pages peuvent être fixées ou paginables, et de diverses clés.

La CSA a longtemps été une zone système "fourre-tout" où l'on mettait de nombreux blocs, des zones de travail, et même des programmes. En SP, son utilisation devait être soigneusement suivie pour éviter son remplissage total, notamment par certains logiciels qui "oublient" de libérer les zones qu'ils ont acquises. De fréquents cycles d'acquisition/libération de mémoire en CSA provoquent parfois un problème de fragmentation de la mémoire virtuelle: à un moment donné, bien qu'il semble que l'espace libre en CSA soit globalement suffisant, on ne peut plus obtenir en CSA un bloc de mémoire de taille convenable; en fait, cet espace libre est "gruyérisé", constitué de nombreux petits morceaux épars libérés par différentes tâches. Le problème est moins crucial en XA avec la CSA étendue.

La **SQA** (system queue area) contient des blocs de contrôle et des tables propres au système dans son ensemble (clé 0), par exemple les tables de segments et de pages des zones communes, les blocs ASCB, la table des bibliothèques autorisées, les blocs de service VSAM (VSRBs), etc. La SQA est fixée en mémoire et s'alloue par blocs de 64 K. La taille de la SQA et celle de la CSA sont paramétrables (paramètres CSA et SQA), et inchangées une fois MVS démarré. Quand le système n'a plus de place libre en SQA, il déborde en CSA. En VM, l'équivalent de la SQA est le "free storage".

D'autres zones existent en MVS-SP1: les BLDLs (fixe et paginable), zones optionnelles, qui contiennent des pointeurs vers les modules les plus utilisés de la link-list (programmes systèmes qui résident sur disque). En MVS-XA ces tables ont été créées systématiquement et mises dans un espace-adresse spécifique, la LLA.

En MVS-XA, pratiquement toutes ces zones ont une partie au-dessus de la "barre" des 16 Méga-octets et une partie en dessous, par compatibilité avec SP1. On parle ainsi de "private" et d'"extended private", de CSA et E-

CSA, etc. (voir le schéma). Selon leur "RMODE" (residency mode) les programmes sont stockés dans l'une ou l'autre partie. Voir les figures 6 et 7.

Certaines zones du système n'apparaissent pas en mémoire virtuelle: outre le noyau "DAT-off" de XA, il y a aussi la HSA (hardware storage area) qui contient le microcode, les blocs de contrôle UCW (unit control word) pour les entrées-sorties, éventuellement le microcode PR/SM pour le partitionnement logique de l'ordinateur. La HSA n'est directement accessible que par le matériel .

E) La pagination

1) LE VOL DE PAGE

La pagination est le moyen par lequel le système MVS contrôle la bonne utilisation de la mémoire réelle par les espaces-adresses virtuels. Il y a deux principaux phénomènes:

* le système veille à ce qu'il y ait toujours assez de cadres de mémoire réelle disponibles: il maintient une réserve de cadres (AFQ, available frame queue). Sur décision de SRM, composant d'optimisation de MVS, il "volera" (transférera sur disque) les cadres les moins utilisés, sans chercher à savoir s'ils appartiennent à de gros ou de petits consommateurs de mémoire: c'est une "pagination contre le système" (et non contre les processus les plus gourmands, ce qui se fait dans d'autres systèmes). Ce vol n'est pas fait à intervalles réguliers, mais uniquement en cas de besoin.

* le système doit permettre aux travaux actifs de pouvoir travailler dans de bonnes conditions: il ne vole pas les pages les plus utilisées et garantit ainsi au travail un espace vital minimum (working set); il ramène en mémoire les pages absentes (faute de page) à partir de la copie qui en est faite sur disque (page-in, ou "demand paging").

Le vol de page, "arme" principale du système, est fondé sur une mesure du "vieillissement" des cadres de page en mémoire. Nous avons vu que la clé mémoire associée à chaque cadre comportait un bit "page référencée" (c'est- à-dire page lue ou modifiée). Ce bit est positionné par le matériel pour toute lecture ou modification d'une donnée dans le cadre. Par ailleurs MVS maintient une table des cadres (page frame table) dans le noyau (ne pas confondre avec la table des pages propre à chaque espace-adresse virtuel). Chaque poste (PFTE, 16 octets en SP, 32 en XA) contient les caractéristiques du cadre: notamment le numéro (ASID, address-space identifier) de l'espace-adresse qui le détient éventuellement, I'adresse virtuelle de la page contenue par le cadre, et un compteur UIC (unreferenced interval count: compteur d'intervalles de non-référence) de taille un octet, contenant une valeur de 0 à 255 qui mesure l'"âge" du cadre.

Le compteur UIC, propre à chaque cadre, est maintenu ainsi: MVS (par une instruction RRB ou RRBE, reset reference bit) inspecte les "reference bits" à intervalles de temps réguliers; quand le bit vaut 0 (page non référencée), I'UIC est augmenté de 1; quand le bit vaut 1 (cadre lu ou modifié) il est remis à 0 (et le bit aussi). Un UIC de 0 indique ainsi que le cadre a été accédé dans le dernier intervalle. Une valeur de n différente de zéro indique que dans les n derniers intervalles le cadre n'a pas été accédé. Plus l'UIC sera grand, plus la page correspondante sera candidate pour être volée, étant donnée son inactivité. Une valeur limite (high UIC), qui mesure la "contrainte" sur la mémoire réelle, est en pratique maintenue par MVS au-dessus de laquelle toute page est volée (sauf exceptions que nous étudierons...). La mise à jour des UlCs des cadres de page est faite à intervalles réguliers, en fonction de la valeur du "high UIC"; I'intervalle est de 1 à 6 secondes en MVS-SP, de 1 à
20 en MVS-XA.

Cet algorithme de rotation des pages, le vol des pages les moins utilisées sur une période récente (LRU, least recently used), suppose que l'avenir ressemblera au passé: il estime (ou espère) que les pages peu utilisées finiront par ne plus l'être du tout (d'où l'intérêt de les voler), et de même que les pages très utilisées risquent de l'être encore, et doivent donc rester en mémoire. D'autres systèmes n'utilisent pas d'UlC (ou équivalent) et se contentent de voler les pages non référencées. Certains algorithmes tiennent compte de la fréquence d'utilisation des cadres pour voler les moins fréquemment utilisés (LFU, least frequently used): la gestion est

simple (incrémentation d'un compteur à chaque accès à une page) mais défavorise à tort les pages récemment chargées et donc non encore utilisées. D'autres algorithmes encore plus simples consistent à voler les pages les plus anciennement chargées par ordre d'arrivée en FIFO (first-in, first-out: premier arrivé, premier sorti) indépendamment de leur plus ou moins grande utilisation, ou même de voler aléatoirement les pages. L'algorithme idéal consisterait à voler les pages dont on est sûr qu'elles ne seront pas utilisées pendant la plus longue période de temps. Une telle prévision est impossible, et l'algorithme LRU (et sa mise en oeuvre dans MVS -par l'UIC- comme dans de nombreux autres systèmes paginés) extrapole le futur proche à partir du passé immédiat. L'expérience montre que cette approximation est correcte.

L'algorithme LRU a une propriété intéressante qui est celle-ci: toutes choses égales par ailleurs, une augmentation de la taille de la mémoire centrale ne risque pas d'augmenter le nombre de fautes de pages; cela paraît être le minimum exigible d'un algorithme de remplacement de pages, mais certains algorithmes (le FIFO) n'y obéissent pas (anomalie de Belady). En revanche le coût d'un algorithme LRU n'est pas négligeable, et met en jeu autant le logiciel (maintenance et exploration de tables de cadres) que le matériel (qui maintient les "reference bits"). Le système VM utilise aussi l'algorithme LRU avec une mise en oeuvre comparable à MVS.

Le vol de page n'est pas une tare des systèmes paginés ni un événement dramatique: c'est l'élément essentiel d'une gestion dynamique des mémoires virtuelles. Ce sont les fautes de pages qui peuvent poser des problèmes de performance quand elles ne restent pas dans des limites raisonnables.

2) LE SUPPORT PHYSIQUE

Le support auxiliaire de la pagination est constitué de fichiers sur disque (page data sets) découpés en cases de 4K sur lesquelles sont transférés les cadres de page. Il n'y a pas une correspondance fixe (comme en OS-VS1) entre un cadre et une case. Pour répondre à toute sortie ou demande de page, le composant ASM maintient une table de correspondance (XPTE, external page table) entre les cadres de mémoire (PFTE) et les cases qui les supportent sur les fichiers de pagination. Ces fichiers sont au minimum au nombre de trois (il peut y en avoir jusqu'à 256 en XA): le fichier PLPA supporte la zone PLPA, ce fichier n'est accédé qu'en lecture puisque la PLPA une fois construite n'est pas modifiable; le fichier COMMON supporte les zones communes modifiables: CSA, MLPA. Enfin le ou les fichiers LOCAL supportent les zones privées des espaces-adresses.

Les fichiers de pagination sont déterminés au démarrage, mais on peut ajouter ensuite dynamiquement des fichiers LOCAL (on ne peut cependant alors plus les désaffecter, sauf en MVS-ESA...). Une option consiste à maintenir une copie des pages sur des fichiers de pagination DUPLEX: on prévient ainsi le risque d'une erreur de lecture physique sur un fichier de pagination. Cette option est cependant quasiment inutilisée à cause du ralentissement qui en résulte (et aussi de la grande fiabilité des disques).

Étant donné l'importance de la pagination dans un système à mémoire virtuelle, le maximum est fait pour accélérer les opérations d'entrée-sortie sur les disques de pagination (elles sont prioritaires par rapport aux opérations sur les autres disques). ASM essaie d'équilibrer la charge de chaque disque de pagination. Par ailleurs, les opérations d'entrée-sortie sur les disques de pagination utilisent une fonction particulière, le "suspend/resume" (suspension/reprise du programme canal) qui permet à ASM de monopoliser le disque et d'améliorer le temps d'accès. Il est recommandé de ne pas destiner les disques de pagination à une autre utilisation.

On évite cependant chaque fois que possible l'accès disque: ainsi du vol de pages non modifiées (le contenu de la case disque est alors encore valable, il est donc inutile de recopier sur disque la page avant de la voler), ou de demande d'une page (suite à faute de page) qui a été volée (bit "page invalide") mais qui se trouve encore en mémoire parce que le cadre de page correspondant n'a pas été encore utilisé (il contient donc encore intactes les données paginées) ou parce que l'opération d'entrée-sortie (page-out) n'est pas terminée; la page est alors "réclamée" (page reclaim) sans accès disque et l'exemplaire en mémoire est réutilisé.

Des améliorations notables sont apportées par divers dispositifs adaptés spécifiquement à la pagination, tels que:

* les unités de contrôle à cache pour pagination: 3880 modèles 11, 21 avec des disques 3350 (les 3880 modèle 23 ou les 3990 modèle 3 étant des unités cache de données);

* la mémoire d'arrière-plan (MAP, "expanded storage") présente sur certains ordinateurs ES/9000, 3090 ou 4381, d'accès très rapide, qui est en cela une véritable extension de la mémoire réelle, si bien qu'une faute de page pour un cadre résidant en MAP est synchrone (résolue sans attente, au même titre qu'une page "réclamée") et n'entraîne pas une suspension du processus comme lorsque la page manquante est sur la mémoire auxiliaire traditionnelle, le disque. Ceci permet d'ailleurs d'avoir en MVS-ESA des processus ininterruptibles qui peuvent supporter des fautes de page, quand les pages manquantes se trouvent en MAP (disabled reference storage).

On transfère en MAP les pages les plus utilisées de préférence au support disque. Cette mémoire est gérée elle aussi par un algorithme LRU; en cas de nécessité (encombrement ou mise "offline") RSM déplace les cadres de mémoire étendue vers les cases de la mémoire auxiliaire traditionnelle, en les faisant passer auparavant par la mémoire réelle (page migration).

3) L'ÉCROULEMENT ET L'ESPACE VITAL

La pagination, support de la mémoire virtuelle, permet d'obtenir une excellente utilisation de la mémoire réelle par un nombre indéfini d'utilisateurs. En revanche elle nécessite une gestion complexe avec de nombreuses tables et de nombreux algorithmes. Elle a aussi ses bornes, et on aurait tort d'oublier le "réel" (limité) qui est le soubassement du "virtuel" (presque illimité conceptuellement). Les limites sont atteintes quand il y a en mémoire trop de processus occupant (de préférence) trop de mémoire. La mémoire étant surutilisée, le vol de page est très grand et empêche les processus de travailler correctement à cause des fautes de pages qu'ils subissent. Le point extrême est atteint quand les processus passent plus de temps à paginer qu'à utiliser le processeur. Tandis que le taux de fautes de pages et de "page-reclaim" est très élevé, et le support auxiliaire de pagination très sollicité, le processeur est sous-utilisé (ce qui, dans les systèmes "primitifs", portait le système à accepter de nouveaux processus et à aggraver le phénomène en emballant le système...). Il faut en toute hâte diminuer le nombre d'utilisateurs (degré de multiprogrammation) pour soulager la mémoire. Un SRM convenablement paramétré suffit à éviter d'en arriver là...

On évite un trop fort taux de fautes de page en assurant aux travaux une fois en mémoire la disposition de leurs pages les plus utilisées: on constitue ainsi un "espace vital" variable qui évolue en fonction du comportement du travail et de la disponibilité de la mémoire réelle. Sauf cas très particuliers, on profite d'une propriété de "voisinage": un programme qui a le contrôle a tendance à référencer des zones voisines dans une "fenêtre" déterminée, éventuellement à sauter d'une "localité" à une autre. Une "localité" constitue à un moment donné un espace vital qu'il faut assurer au processus pour qu'il puisse s'exécuter dans de bonnes conditions.

4) L'ISOLATION MÉMOIRE (STORAGE ISOLATION)

L'algorithme LRU traite de la même façon tous les espaces-adresses (LRU global) et leur vole des pages sans distinction. Cependant, certains sous-systèmes importants, notamment les sous-systèmes transactionnels ou interactifs (IMS, CICS, DB2, TSO), risquent de voir leur temps de réponse diminuer fortement s'ils doivent subir de nombreuses fautes de page. Le problème est patent quand on passe brusquement d'une période de relative inactivité pendant laquelle beaucoup de pages ont été volées à une période de pleine charge. Quelques dizaines de fautes de page peuvent suffire à diminuer d'une seconde le temps de réponse, ce qui devient sensible à l'utilisateur quand s'y ajoute le temps de transfert sur le réseau.

Pour cette raison, on peut souhaiter garder en mémoire un minimum de pages, même si elles sont peu référencées. Ce minimum doit être raisonnable, sans quoi le vol de page habituel sera bien plus sévère sur les autres travaux, qui s'en trouveront tous pénalisés. On peut préciser aussi une valeur maximum, au-dessus de laquelle le vol de page sera permis; cette valeur elle aussi doit être bien adaptée, car le vol de page s'exerce de préférence sur ce surplus de pages. La possibilité d'isolation mémoire offre presque les mêmes avantages que le V = R, la souplesse en plus: les pages "isolées" ne seront volées qu'en cas de nécessité, en dernier recours. Les

zones communes paginables, sujettes aussi au vol de page, peuvent être aussi protégées par une isolation mémoire .

F) Le vidage (swapping)

Le vidage (swap-out) est l'action de transférer sur mémoire auxiliaire (disque par exemple) les pages mémoire d'un espace-adresse devenu inactif (en attente d'un événement: montage de volume, entrée de données au clavier d'un terminal, etc.) et qui ne requiert plus pour l'instant le processeur. L'opération inverse (swap-in) consiste à ramener ses pages quand il redevient actif. Ces opérations peuvent aussi être déclenchées par le système s'il juge adéquat pour les performances d'inactiver ou de réactiver un espace-adresse.

L'intérêt est évidemment de pouvoir désengorger d'un seul coup la mémoire en vidant toutes les pages devenues inutiles, plutôt que de les voler petit à petit une à une selon l'algorithme LRU. De plus il est bien plus rapide de faire une seule entrée-sortie de n pages que n d'une page. Alors que la pagination ne s'effectue qu'en cas de nécessité (vol de pages pour maintenir le "coussin" de cadres, transfert à la demande pour résoudre une faute de page), le vidage et le retour en mémoire résultent de décisions a priori du système (SRM). Le vidage peut aussi être provoqué occasionnellement par un manque critique de cadres de page qui n'a pu être résolu par les moyens habituels (frame shortage) ou par un manque de place en mémoire auxiliaire.

La préhistoire du swap peut sans doute se situer avec l'option de roll-in / roll-out de MVT, qui vidait sur disque une région. Le swap en MVS fut d'abord restreint à la LSQA, ce qui soulageait d'autant les fichiers de pagination. Il fut généralisé en MVS-SP aux autres pages de la zone privée avec le vidage étendu (extended swap). Le vidage ne s'applique pas actuellement à toutes les pages de la zone privée; il est limité à la LSQA, aux pages fixées et à un espace vital réduit (trim swap) constitué par les pages récemment référencées. En font partie au moins les pages d'UlC nul (elles seules dans les premières versions de MVS-SP). La "réduction" de l'espace vital dépend de la contrainte sur la mémoire, mesurée par la valeur maximum des UIC (high UIC). Les pages changées mais non référencées (un tel cas peut se produire après un "reset" du reference-bit par RRB) sont volées et ne reviendront en mémoire qu'à la demande. Enfin les pages ni changées ni référencées sont rendues disponibles pour un autre emploi sans transfert sur mémoire auxiliaire. Le mécanisme du vidage met en oeuvre divers composants de MVS (déjà rencontrés) dans la séquence suivante:

SRM invoque la RCT (region control task) de l'espace-adresse, qui y stoppe toute activité et appelle RSM; ce dernier composant demande à ASM le transfert (vol) des pages qui ne font pas partie de l'espace vital (trim swap); il demande ensuite de sortir de la file du distributeur l'espace-adresse; enfin ASM effectue le vidage proprement dit. Le retour en mémoire (swap-in) est effectué par les mêmes composants et toujours provoqué par SRM.

Quand la mémoire est abondante ou sous-utilisée le vidage physique est injustifié, et coûterait des entrées-sorties inutiles. L'espace-adresse est alors considéré comme "logiquement" vidé (logical swap): les pages de son espace vital sont encore présentes en mémoire, au moins pour un certain temps (qui est un "think time", temps de réflexion moyen, pour un utilisateur TSO) fonction de la contrainte sur la mémoire (toujours selon l'UIC global), après lequel un vidage physique peut survenir ("detected long think time swap"). A noter d'ailleurs que, comme pour un vidage physique, il y a auparavant vol des pages peu référencées (trim swap), ce qui peut poser en l'absence de MAP certains problèmes de performance lors du "swap-in" logique, avec de nombreuses fautes de page (sauf s'il y a isolation mémoire, car l'espace vital est alors plus grand). Les pages d'un espace-adresse vidé logiquement ne sont pas soumises à la mise à jour régulière de leur UIC (elles ne sont pas référencées). Elles peuvent tout de même être volées. L'algorithme de vol de page de MVS essaie de contrebalancer les avantages accordés parfois trop généreusement à certains espaces-adresses: avant de voler des pages aux travaux "normaux" et aux zones communes, il vole des pages aux espaces-adresses isolés en mémoire et utilisant plus de cadres que leur maximum, puis à ceux qui sont swappés logiquement et dépassent leur espace vital.

Le swap est une opération "sévère", qui pénalise, certes non sans bien fondé, le travail qui le subit; le swap n'existe pas dans certains systèmes, tel VM (sauf VM-HPO); il n'existait pas en OS-VS1. On peut protéger certains travaux ou sous-systèmes importants en les déclarant "non-swappables". Cela ne les empêche pas évidemment de subir le vol de page, comme tous les autres travaux en mémoire...

La mémoire auxiliaire peut être la MAP des 3090 ou des fichiers sur disque: les fichiers swap. Le transfert entre mémoire réelle et auxiliaire s'effectue par groupes ("swap-sets") de 12 pages affectés à des cases disques contiguës. Les fichiers swap sont facultatifs (et pas recommandés la plupart du temps, surtout si on bénéficie de la MAP); en leur absence le vidage s'effectue sur les fichiers de pagination. La MAP optimise le swap en recueillant les pages écartées de l'espace vital lors du vidage. Lors d'un vidage physique la MAP recueille au minimum un espace vital primaire constitué de la LSQA, des pages fixées et d'une page par segment contenant des pages "récemment" référencées; l'espace vital secondaire, constitué des autres pages récemment référencées ira aussi sur la MAP mais ne sera pas systématiquement ramené en mémoire au swap-in (il le sera à la demande, par faute de page). Ce redécoupage de l'espace vital permet de minimiser l'impact d'une migration éventuelle des cadres de la MAP: l'espace vital primaire est migré en une seule fois, tandis que le reste l'est une page à la fois.

Les différentes causes de swap sont les suivantes:

* diverses mises en attente: entrée de commandes au terminal pour les utilisateurs du temps partagé ("terminal input"), attente d'affichage d'un résultat à l'écran ("terminal output", un terminal de 24 lignes ne peut en afficher 100 d'un seul coup...), toute attente reconnue "longue" par le système ("long wait", plus de 0,5 seconde), une attente non signalée au système mais détectée par SRM ("detected wait") suite par exemple à l'absence de réponse d'une unité périphérique: "missing interrupt", positionnement d'une bande (la bande tourne jusqu'à atteindre le fichier désiré, ceci peut prendre plusieurs dizaines de secondes), disque inaccessible car réservé par un autre système, etc.

* les swaps par lesquels SRM optimise l'utilisation des ressources: unilatéral quand on décide le vidage ou le retour en mémoire d'un travail; d'échange quand on vide un travail actif et recharge un travail "out"; enfin le swap "enqueue exchange" qui fait revenir en mémoire un travail qui détient (par ENQ) une ressource demandée par d'autres, cela dans l'espoir qu'il la libère et permette aux autres de travailler.

* les swaps dus aux "situations de crise" pour soulager la mémoire réelle ou la mémoire auxiliaire (storage shortage); on vise les plus gros consommateurs de mémoire, si la pénurie provient d'une trop forte pagination, ou ceux qui occupent beaucoup de pages fixées.

* enfin des cas très particuliers: le "request swap" qui vide un travail suite à l'inactivation (mise en "offline") des cadres qu'il occupe; le swap de transition survenant quand un travail devient non-swappable.

G) Le "cross-memory"

La mémoire virtuelle entraîne une séparation des espaces-adresses entre eux: à un instant donné uniquement la table de segments de l'espace-adresse traité par le processeur est utilisée pour décoder les adresses. Il serait pourtant intéressant de faire communiquer entre eux les espaces-adresses, pour mettre en commun des données ou des programmes autrement que par les zones communes, toujours limitées. De même l'emploi de SRBs est peu commode: leur déroulement asynchrone (avec invocation du distributeur) en parallèle avec la tâche émettrice impose une resynchronisation (par "cross-memory post") avant d'être sûr d'obtenir les données de l'espace-adresse visé. Ces données ne sont pas obtenues immédiatement, mais après transfert par les zones communes. Enfin uniquement les étapes de travaux autorisées peuvent émettre des SRBs, ce qui permet difficilement de généraliser ce mode de communication à tous les utilisateurs.

Les machines d'architecture 370 ont été pourvues de nouvelles instructions qui permettent à un espace-adresse l'accès direct (synchrone) aux données d'un autre espace-adresse ou l'appel d'une routine qui s'y trouve. Ce mode de communication, le cross-memory, a été de plus en plus utilisé par les différentes versions de MVS pour créer des espaces-adresses contenant du code ou parfois uniquement de grands volumes de données; on a pu ainsi économiser de la place dans les zones communes en créant des espaces-adresses spécialisés: GRS, SMF, CATALOG, TRACE, ALLOCAS, etc. En même temps la relative isolation qu'offre la mémoire virtuelle est plus sûre que l'emploi des zones communes. Les grands sous-systèmes de MVS utilisent aussi très fréquemment le cross-memory. Les espaces-adresses impliqués dans le cross-memory doivent de préférence être non vidables pour que la communication soit possible.

Le matériel supporte le cross-memory grâce aux dispositifs suivants: de nouvelles instructions de transfert de contrôle ou de transfert de données, un registre de contrôle (le CR7) pour pointer sur la table de segments d'un autre espace-adresse, un bit du PSW ("secondary space mode") pour indiquer si l'on utilise le CR1 habituel ou le CR7 pour accéder à la table des segments, ce qui détermine quel espace-adresse est référencé (espace-adresse "primaire" avec le CR1, "secondaire" avec le CR7). Les instructions qui supportent le cross-memory sont les suivantes:

* PC (program call): permet de transférer le contrôle de façon synchrone à un autre espace-adresse. Les opérandes du PC indiquent un jeu de tables (linkage table et entry table) pour déterminer l'espace-adresse visé, l'adresse de branchement, des paramètres et les valeurs de clé (authorization key mask) que doit posséder l'appelant (ce qui permet plus de souplesse que la clé zéro nécessaire pour les SRBs). Quand le PC provoque un changement d'espace (space switch) il y a échange entre CR1 et CR7 pour que le CR1 pointe sur la table de segments de l'espace invoqué.

* PT (program transfert): utilisé inversement pour revenir à l'espace-adresse primaire.

* SSAR: prépare le cross-memory en chargeant dans le CR7 la table des segments de l'espace secondaire que l'on veut atteindre.

* SAC (set address space control): positionne le bit du PSW pour indiquer à quel espace on accède (quand ce bit vaut 1, on accède à l'espace secondaire avec le CR 7). Par cette instruction un programme peut accéder alternativement aux données de deux espaces-adresses (ce programme doit résider dans les zones communes pour que ses instructions puissent être accessibles dans tous les cas).

* MVCP (move to primary): transfère des données depuis l'espace secondaire vers le primaire.

* MVCS (move to secondary): transfère des données du primaire vers le secondaire. MVCP et MVCS, pour bien fonctionner, doivent avoir été précédés d'un SSAR qui a déterminé l'espace secondaire.

Une protection est établie pour éviter qu'on utilise abusivement les instructions SSAR et PT. L'espace appelant dispose d'un AX (authorization index) qui indique l'autorité de cross-memory dont il dispose. Cet AX est un index dans la table d'autorisation (AT) de l'espace visé. Par convention un index de 0 (valeur par défaut) indique que l'appelant n'est pas autorisé; un index de 1 permet d'effectuer un SSAR ou un PT vers tout espace actif. Les valeurs supérieures donnent ou non l'autorité nécessaire selon les postes de la table AT.

Le cross-memory étant un mode d'exécution assez particulier ne peut utiliser tous les services de MVS, notamment les SVCs.

Nous invitons le lecteur intéressé à voir en annexe un exemple de programme effectuant du transfert de données par cross-memory.

H) Les tendances actuelles

Au cours des dernières années on a pu constater, pour la mémoire virtuelle, l'évolution suivante:

* l'augmentation de la taille de la mémoire virtuelle qui passe de 16 Méga-octets (avec l'adressage à 24 bits de SP) à 2 Giga-octets (avec l'adressage à 31 bits de XA); avec MVS-ESA cette taille passe à 16 Téra-octets (jusqu'à 8192 espaces de 2 Go). La portion de mémoire virtuelle en dessous des 16 Méga-octets est peu à peu désencombrée d'une release à l'autre, par mise au-dessus de la "barre" de nombreux programmes et blocs de contrôle.

* la multiplication des espaces-adresses en remplacement des zones communes: on échappe ainsi à la contrainte "verticale" de la mémoire virtuelle en exploitant sa dimension "horizontale".

* la séparation des données et du code: on améliore la sécurité en séparant des choses dissemblables. Le code réentrant, par nécessité de multi-programmation, effectue une séparation à un premier niveau entre le code, figé, et les données dynamiques. L'étape suivante consiste à utiliser des espaces-adresses spécifiques pour stocker des données accessibles par cross-memory. Enfin, avec MVS-ESA, on peut créer des "dataspaces" de 2 Giga-octets complètement réservés aux données (ou à des programmes exécutables considérés comme des données à stocker) et accessibles par mécanisme matériel. L'isolation est cette fois maximale.

* le remplacement des entrées-sorties disques par de la pagination ou des accès mémoire, pour obtenir de meilleures performances et éliminer ce goulet d'étranglement traditionnel que sont les entrées-sorties. Cette tendance est ancienne et plus marquée encore à présent qu'existent de grandes capacités de mémoire centrale ou de mémoire étendue (expanded storage des 3090). Les dispositifs suivants en témoignent:

- le "virtual fetch" permet de stocker de manière permanente des programmes en mémoire et d'éviter un accès disque à chaque appel.

- le VIO (virtual I/O) permet de créer des fichiers temporaires en mémoire ou sur les fichiers de pagination plutôt que sur disque.

- le DIV (data in virtual) apparu avec MVS-XA 220 permet de traiter des fichiers VSAM de type "linéaire" comme s'ils résidaient en mémoire virtuelle. Ces fichiers sont constitués de blocs de 4K correspondant à la taille d'une page de mémoire; seuls les blocs lus sont amenés en mémoire, seuls les blocs modifiés sont recopiés en fin de traitement. Le SGBD DB2 utilise intensément le DIV. Le "data windowing" de MVS-ESA est une extension du DIV qui est utilisable par les langages évolués (alors que le DIV à l'origine nécessite un code assembleur).

- les dataspaces de MVS-ESA, déjà évoqués, permettent d'isoler les données pures du code exécutable et de les mettre à la disposition de un ou plusieurs espaces-adresses; les mécanismes habituels de gestion de la pagination permettent aux données les plus actives de rester en mémoire (ou de profiter de la mémoire étendue) et aux autres d'aller en mémoire auxiliaire. Ils subissent le vidage en même temps que l'espace-adresse propriétaire. Ils sont conçus uniquement pour stocker des données: on ne peut exécuter directement un programme dans un dataspace. On peut aboutir à une extension logique de la CSA grâce aux dataspaces de type "partagé".

L'accès aux dataspaces est analogue au cross-memory, avec notamment l'instruction SAC, vue au paragraphe précédent, qui permet de quitter le mode primaire pour se mettre dans un mode nouveau: le mode AR (access register), qui permet d'accéder aux dataspaces par les instructions habituelles (sans MVCP ni MVCS). Voir un exemple de programmation en annexe.

- les hiperspaces (high-performance spaces) constituent un autre apport de MVS-ESA. Il s'agit d'espaces de stockage résidant en mémoire auxiliaire ou en mémoire étendue (jamais en mémoire centrale). On peut exiger qu'ils restent en mémoire étendue (expanded storage only), la mémoire auxiliaire fournissant le dernier support quand la mémoire étendue est surutilisée. Pour être traitées, les données sont copiées en mémoire centrale par pages de 4K. Une application des hiperspaces est fournie par VSAM pour ses tampons, par le tri (DFSORT) pour accélérer le traitement, et par l'hiperbatch depuis MVS/SP313 (qui consiste à stocker dans des hiperspaces les fichiers QSAM ou VSAM les plus accédés par les travaux batch). Avec les dataspaces la MAP peut être considérée comme une extension de la mémoire centrale, tandis qu'avec les hiperspaces, il s'agit d'une extension de la mémoire externe qui correspond bien à cette tendance, évoquée plus haut, à éliminer les entrées-sorties pour les grands
systèmes.

- le VLF (virtual lookaside facility) utilise les dataspaces pour stocker des données très accédées (programmes, membres de PDS, Clists, ou même catalogues) à la manière du virtual fetch pour les modules. La LLA (linklist lookaside) de MVS-XA a été améliorée pour stocker d'autres répertoires que ceux de bibliothèques de modules et pour placer les objets les plus actifs dans un espace VLF.

HIPERBATCH : une application d'ESA

Hiperbatch (high performance batch) met en oeuvre le concept de "données en mémoire" : l'idée consiste à intercepter certaines E/S disque et à les satisfaire à partir de la mémoire d'arrière-plan. La MAP joue ainsi le rôle d'une mémoire cache rapide et de grande taille, ce qui rend les opérations de lecture très rapides (les E/S d'écriture, elles, opèrent à la fois sur le disque et en MAP, par précaution). Les travaux qui utilisent l'Hiperbatch de façon intense voient leur temps "elapse" diminuer d'au moins 50% au prix d'un léger accroissement du temps CPU.

Les fichiers concernés par Hiperbatch sont les fichiers séquentiels accédés en QSAM et les fichiers VSAM accédés en NSR et de CI de taille 4096 (ou un multiple de 4096). A l'ouverture, le composant DLF détermine si le fichier est éligible pour une gestion Hiperbatch : il devient dans ce cas un "objet DLF". Les blocs de données lus sont conservés en MAP (dans un hiperspace de type ESO) avec une gestion de type MRU (most recently used), les blocs retrouvés le plus récemment étant les moins susceptibles d'être redemandés dans un avenir proche. On distingue les objets "retenus" et "non-retenus" : ces derniers sont supprimés en MAP dès que plus aucune tâche ne les demande, alors que
les premiers quitteront la MAP sur une commande DELETE classique ou sur passage d'un utilitaire COFMSTCN.

Le paramétrage de DLF repose sur le membre COFDLFxx de la PARMLIB (taille de l'hiperspace) et sur des profils RACF liés à la classe DLFCLASS pour le choix des fichiers éligibles (pour les anciennes versions de RACF on doit passer par un exit COFXDLF1). Les clients qui disposent d'OPC/A peuvent déclarer les fichiers éligibles comme des "ressources spéciales OPC/A" et employer l'exit CSYDLFX.

L'aspect suivi des performances est pris en charge par un moniteur COFDMON et par l'examen des enregistrements SMF de type 14, 15 et 64.

Hiperbatch s'applique idéalement aux fichiers batch séquentiels "inter-steps" ou "inter-chaînes" ou à des fichiers très utilisés (fichiers tables en VSAM). Il n'y a pas besoin de modifier les JCLs existants, à l'inverse d'autres apports d'ESA tels que le batch LSR. Cependant les utilitaires qui emploient directement l'EXCP ou BSAM plutôt que QSAM ou VSAM ne peuvent profiter de l'hiperbatch. Les exploitations "traditionnelles" qui privilégient encore le séquentiel au détriment des SGBD peuvent optimiser sensiblement la largeur de leur "fenêtre batch" grâce à Hiperbatch.

4. La RESSOURCE ENTREE-SORTIE

A) Le matériel

Une opération d'entrée-sortie (E/S) consiste en un transfert d'information entre la mémoire du processeur et les périphériques, unités d'entrée-sortie; l'entrée est le transfert entre le périphérique et la mémoire, la sortie est l'opération inverse. Ces transferts s'accompagnent d'un changement physique de nature de l'information (mais non de signification) puisque l'information est représentée différemment en mémoire et sur les supports périphériques.

Ces opérations, lentes au regard de la vitesse de traitement des instructions par le processeur, sont désynchronisées au maximum par rapport à ce dernier; ceci augmente le "débit" de l'ordinateur et lui permet de faire de la multi-programmation (voir chapitres précédents).

De nombreux composants matériels bien spécifiques entrent en jeu dans une opération d'entrée-sortie; I'exemple proposé est celui d'une opération d'E/S disque dans un environnement XA ou ESA. Les trois composants essentiels sont:

* **le sous-système canal** (channel subsystem): connecté aux unités par les canaux (channel paths), ce composant de l'ordinateur décharge le processeur de la gestion des E/S. Vis-à-vis du processeur les unités d'E/S sont représentées par un sous-canal (subchannel); le lancement d'une E/S par le processeur aboutit en dernier lieu à une instruction SSCH (start subchannel) en XA ou SIO (start I/O) en 370 par laquelle le processeur remet au matériel la responsabilité des opérations; il peut poursuivre d'autres traitements jusqu'à ce que le sous-système canal lui signale la fin de l'opération par une interruption.

Un canal peut être considéré en lui-même comme un processeur, avec ses registres et le programme qu'il est chargé d'exécuter: le "programme canal", constitué de mots de commande canal (CCW, channel command words) qui décrivent la demande d'entrée-sortie. Un CCW correspond à une opération élémentaire et décrit le type d'opération d'E/S (lire, écrire, positionner le bras d'accès disque, rembobiner la bande, etc.), L'adresse de la zone mémoire pour le transfert, la longueur des informations. Les CCWs sont souvent regroupés entre eux par chaînage pour effectuer plusieurs opérations consécutives: dès qu'une opération est terminée, l'unité le signale au canal ("device end") et traite le CCW suivant; quand toutes les opérations sont terminées, le canal le signale au processeur par interruption ("channel end") .

La notion de sous-canal, apparue avec l'IBM 360, donne l'apparence d'un processeur indépendant traitant un programme canal. Selon le niveau de concurrence entre programmes canaux, on parlera de :

- canal "selector" : 1 seul sous-canal est prévu, ainsi une seule opération est permise avec une seule unité à la fois. Le canal est en liaison avec l'unité jusqu'à ce qu'il n'y ait plus de commande canal à traiter pour elle (exemple du contrôleur 3803 avec les 3420).

- canal multiplexeur : il y a jusqu'à 256 sous-canaux, chacun pouvant exécuter un programme-canal. Pour des unités disques, on verra ainsi un sous-canal avec un programme canal par bras d'accès. Contrairement au cas précédent, le programme canal peut être interrompu entre deux transferts de blocs (cas du canal multiplexeur par blocs) : le canal n'est monopolisé par le programme canal que le temps d'un transfert, il est "relâché" lors des mouvements de bras (RPS).

* **une unité de contrôle** (ou "contrôleur"): cet élément interprète les signaux envoyés par les canaux et les transmet aux unités en les "personnalisant" au besoin en fonction du type d'unité; il a aussi une fonction de stockage intermédiaire. Le contrôleur décharge le canal en se chargeant de certaines fonctions auxiliaires trop particulières pour lui (qui ne connaît que les CCWs) et qui dépendent du type d'unité: le positionnement des bras d'accès de disque, le rembobinage des bandes magnétiques, etc. Le système ne communique pas ses ordres directement aux unités, mais aux contrôleurs.

Le contrôleur est adapté aux unités d'entrée-sortie auxquelles il est connecté: dans notre exemple il s'agit d'un contrôleur 3880 pour les disques 3380. Certaines unités ne nécessitent pas de contrôleur ou comportent cette fonction de façon intégrée: CTC, disques 3380 CJ2, etc. Le contrôleur est dirigé par un programme microcodé chargé lors de la mise en fonction depuis une disquette ou un disque (IML).

* **une unité d'entrée-sortie**, chaînon final (dans notre exemple il s'agit de 16 unités de disques 3380). Les unités (I/O devices) servent de moyen de stockage (unités disque ou bande) ou de communication (écran, imprimante). MVS distingue différentes grandes classes d'unités, selon leur utilisation: unités en accès direct (disques); unités à bandes magnétiques (bandes ou cassettes); unités de visualisation (terminaux); unités de communication (par exemple contrôleurs 37X5); les unités (unit record) telles que les imprimantes, les lecteurs de cartes; les unités CTC qui permettent des liaisons "canal à canal" entre ordinateurs; enfin les unités "lecteurs de caractères".

Une unité peut être connectée à plusieurs contrôleurs eux-mêmes connectés à plusieurs canaux: on peut ainsi disposer de nombreux "chemins" d'accès et bénéficier d'une plus grande sécurité et d'une meilleure performance: si un contrôleur ou un canal est occupé ou même pose problème, on emprunte les chemins disponibles restants pour arriver à l'unité. De même, certaines unités (en XA) peuvent choisir le chemin de retour de l'E/S (reconnexion dynamique) .

Les composants matériels remplissent chacun leur fonction en parallèle, avec la plus grande désynchronisation possible: ainsi, avec la fonction RPS (rotational position sensing), une unité disque en cours de positionnement du mécanisme d'accès ("seek") libère le contrôleur et le canal pendant le temps de l'opération, et ne se "reconnecte" que pour le transfert ("search"), quand il estime que la tête de lecture/écriture arrive au-dessus des données (pour cela une piste est divisée en secteurs égaux).

L'exemple d'E/S disque est relativement complexe (nous n'avons d'ailleurs pas évoqué la "tête de string" qui a aussi une fonction de contrôleur). Un autre exemple pourrait être celui d'une opération d'E/S à destination d'un terminal éloigné ("remote", par opposition aux terminaux "locaux" présents physiquement sur le site et directement connectés à l'ordinateur). Elle mettrait en jeu les éléments suivants (cas d'une sortie): un contrôleur de communication, puis un modem qui transforme les données numériques en signaux analogiques, une ligne de communication (ligne téléphonique par exemple) qui les transmet à destination, un modem qui reconvertit les données à l'arrivée, un contrôleur de terminaux, enfin un terminal qui reçoit à l'écran les données.

ESCON et les opérations asynchrones.

ESCON (évoqué au chapitre 1, A.3) apporte un changement radical dans l'architecture d'E/S : le "channel subsystem" est tout à fait différent sans que les méthodes d'accès de MVS soient impactées.

Avec ESCON et les canaux à fibre optique, les données sont transmises en série bit par bit, alors qu'elles étaient envoyées en parallèle (les 8 bits d'un octet en même temps) dans les canaux en cuivre qui ont précédé ESCON ; il y avait des câbles "bus" (pour les données) et des câbles "tag" (pour les informations de contrôle), alors que les "frames" de données ESCON véhiculent maintenant aussi bien des informations de contrôle et de routage que les données proprement dites. Les canaux parallèles souffraient d'un risque d'affaiblissement du signal à mesure qu'on s'éloignait de l'ordinateur, ce qui limitait la distance unité de contrôle - ordinateur à 400 pieds (122 m). Leur remplacement par la fibre optique intéressera surtout les responsables de salle machine (ESCON n'améliore pas les performances, ce qui est gagné en temps de transfert canal étant perdu par augmentation du temps de protocole).

L'ESCD (ESCON director), genre de commutateur très rapide, et son logiciel, l'ESCM, constituent la plaque tournante des opérations d'E/S. L'ESCD permet de connecter plusieurs unités de contrôle au même canal en évitant les connexions "en guirlande" qui prévalaient jusque là. Il apporte la possibilité de "basculer" sur une configuration de secours par d'autres voies d'accès (type unité 3814). Il permet enfin d'effectuer des opérations asynchrones, pourvu que l'unité de contrôle s'y prête (3490, 3390-3, 3174-22L) et effectue elle-même la bufferisation : le canal n'est plus alors utilisé que pour une opération d'E/S, il n'est plus sollicité comme auparavant quand la tête de lecture arrive près de l'enregistrement recherché (search). Il peut fonctionner à son plein rythme, plutôt qu'à celui du disque, pourvu que la bufferisation soit suffisante.

B) Les composants logiciels

Plusieurs composants logiciels entrent en jeu dans l'exécution d'une opération d'entrée-sortie, et on peut les décrire en couches plus ou moins proches du matériel. Les interactions entre ces composants seront examinées au paragraphe D.

1) L'IOS

L'IOS (input/output supervisor, dans d'autres systèmes on parle d'lOCS: I/O control system) est le composant de MVS qui est chargé au niveau le plus bas de la gestion des E/S: déclenchement des E/S, traitement des interruptions d'E/S, signal de fin d'opération au programme demandeur. L'IOS a été concu différemment en mode 370 pur et en 370 XA ou ESA. En 370 le composant matériel de sous-système canal est méconnu de MVS, et l'IOS traite complètement l'E/S en choisissant lui-même le chemin d'accès à l'unité (path) et en gérant les conditions de chemin occupé (busy). Le sous-canal est un chemin physique de communication entre un canal et une unité; l'IOS choisit un sous-canal pour l'opération et ce chemin sera le même au retour de l'E/S. Un canal est dédié à un processeur et une interruption d'E/S est présentée uniquement au processeur émetteur de l'E/S. En XA, l'IOS lance l'opération d'E/S sur l'unique sous-canal logique représentant l'unité visée; le sous-système canal fait
dynamiquement le choix du chemin d'accès physique (dynamic path selection); un canal n'est plus dédié à un processeur, toute unité est accessible pour tout processeur et tout processeur est apte à traiter une interruption d'E/S. Ainsi une partie des fonctions de l'IOS a été déportée vers le sous-système canal (et donc vers le matériel et son microcode), ce qui améliore très nettement les performances en E/S des systèmes XA.

2) L'IOS DRIVER

L'IOS est le FLIH (gestionnaire d'interruption de premier niveau) pour les E/S. Il n'a qu'un rôle d'interface avec le matériel: il émet les instructions SSCH (ou SIO en 370) et "récupère" en retour les interruptions. Il gère une file d'attente d'opérations d'E/S pour chaque unité.

Un IOS driver "pilote" I'IOS en préparant l'environnement nécessaire à l'E/S; il fixe en mémoire les pages contenant les buffers, car le sous-système de canaux ne connaît ni faute de page, ni DAT, ni mémoire virtuelle; il traduit en adresse réelle les adresses des zones d'E/S que comportent les CCWs; enfin, quand l'IOS lui signale la fin d'une opération, il passe le contrôle au distributeur (voir chapitre 2) car la fin de l'opération permettra à nouveau à l'utilisateur émetteur (peut-être très prioritaire) de reprendre son travail. Le rôle de pilote de l'IOS apparaît dans la possibilité qu'ont certains drivers de contrôler l'exécution de l'E/S par des exits ("appendages" des méthodes d'accès) et même d'influer sur l'E/S au cours de son exécution en permettant aux méthodes d'accès de modifier le programme canal (PCI, program controlled interrupt).

L'IOS driver le plus courant est l'EXCP (execute channel program), appelé par la macro EXCP (qui génère une SVC), mais certains sous-systèmes ou méthodes d'accès ont leur propre driver: ainsi pour ASM (gestionnaire de mémoire auxiliaire), VSAM, JES3, VTAM, etc. L'EXCP peut être invoqué directement par un programme utilisateur qui code ses CCWs (voir un exemple en annexe), mais les utilisateurs habituels de l'EXCP sont les méthodes d'accès.

3) LES METHODES D'ACCES

Une méthode d'accès est une routine directement appelée par un programme utilisateur pour accéder à une unité; elle correspond tant à une organisation des données sur le support qu'à une technique pour y accéder. Elle libère le programme du détail de l'E/S en construisant les blocs de contrôle nécessaires et surtout les CCWs du programme canal. Elle communique avec un IOS driver qui l'informe de la fin de l'opération.

4) DFP

Une fonction importante du système est de rendre l'accès aux données par l'utilisateur aussi indépendant que possible du support et du chemin physique emprunté; cette transparence permet à l'utilisateur d'obtenir cet accès grâce à quelques fonctions logiques: le traitement d'un fichier consistera à se l'allouer, l'ouvrir, le lire, l'écrire, le fermer, le désallouer. Un programme écrit pour traiter un fichier fonctionnera de la même façon que le fichier soit sur disque ou sur bande. Les composants de MVS assurant la gestion des données à tous les niveaux ont été regroupés pour donner un produit sous licence inclus en standard dans MVS: DFP (data facility product), qui assure les fonctions suivantes:

* gestion des unités (définition et utilisation);

* gestion des données (méthodes d'accès, catalogues, utilitaires);

* gestion des programmes (création, chargement).

Les logiciels de la famille "DF" (DFDSS, DFHSM, DFSORT, etc.), dont nous reparlerons, sont optionnels.

C) Fichiers et volumes

1) GÉNÉRALITES

Une grande partie des unités d'E/S d'un site informatique (surtout en gestion d'entreprise) est consacrée au stockage des données. Les données sont organisées en fichiers; un fichier est pour l'informaticien une unité logique d'information; pour le système c'est une unité de stockage (sur support magnétique le plus souvent), avec certaines caractéristiques, un début, une fin, une organisation, éventuellement un nom. Le volume est le support de stockage lui-même, disque, bande ou cassette, qui doit être monté sur une unité d'E/S pour être opérationnel.

Un fichier est constitué d'un certain nombre d'enregistrements et de blocs regroupant plusieurs enregistrements contigus; le bloc est l'unité de transfert entre le support et la mémoire, tandis que l'enregistrement est vis-à-vis du programmeur une unité logique de traitement. L'enregistrement est un découpage logique respecté par le système quand il restitue les données; les blocs, eux, sont physiquement séparés par des "gaps".

Un fichier peut être décrit par une étiquette (label) placée sur le volume qui le contient. Le label fournit les renseignements nécessaires au système pour accéder au fichier: taille d'enregistrement et de bloc, organisation, nom, espace alloué pour le cas de fichier disque, etc. Pour certains fichiers bande, il n'y a pas d'étiquette de fichier (bande "no label") et le système doit se fier aux informations du programme ou du langage de commande pour connaître les caractéristiques du fichier. Dans d'autres cas, comme pour les fichiers VSAM, le label (pour les fichiers sur disque il s'agit d'un DSCB, data set control block) est insuffisant et les informations sont prises dans un catalogue, qui répertorie ainsi tous les fichiers qu'on veut bien lui faire connaître. Les volumes disposent eux-mêmes d'étiquettes utilisées par le système pour les reconnaître et les monter sur les unités d'E/S adéquates.

2) ALLOCATION

Le fichier est mis à la disposition d'une tâche par une fonction d'allocation qui, en identifiant le volume où réside le fichier et l'unité où doit se trouver ce volume, prépare l'accès aux données. On contrôle la disponibilité du fichier, de l'unité d'E/S, des volumes qu'il faut éventuellement monter sur les unités. A ce niveau, il n'y a pas forcément accès physique aux unités, sauf par exemple en cas de création de nouveaux fichiers sur disque. La fonction d'allocation s'achève par la création de blocs de contrôle; la zone SWA reçoit les JFCBs (job file control blocks) qui identifient chaque fichier alloué; la table TIOT (task input/output table) dresse la liste des JFCBs et des UCBs des unités à allouer.

L'allocation des fichiers est réalisée au niveau de chaque étape d'un travail, en fonction des indications du langage de commande JCL. Elle peut aussi être effectuée n'importe quand par un travail au cours de son exécution, grâce à la fonction d'allocation dynamique, appelée par la SVC 99, avec un paramétrage très souple bien qu'un peu touffu (les "text units").

La fonction de désallocation remet à la disposition du système les ressources utilisées: fichier, volume, unité et décide de conserver ou non chaque fichier, selon les souhaits de l'utilisateur ("disposition").

3) OUVERTURE

L'opération d'ouverture d'un fichier (OPEN), qui suit normalement l'allocation, consiste à identifier les caractéristiques physiques du fichier ainsi que la méthode d'accès qui permettra par la suite la lecture ou l'écriture. Le DCB (data control block) du programme est complété par les informations du JCL (le JFCB) et les données du label du fichier (DSCB pour les fichiers disques) pour obtenir une description complète du fichier; la méthode d'accès est identifiée et son adresse est indiquée dans le DCB. L'équivalent du DCB est l'ACB (access method control block) pour les méthodes d'accès VSAM ou VTAM. Le DEB (data extent block) est un autre bloc associé au DCB qui comporte des informations physiques supplémentaires (extents de fichier sur le disque), pointe sur le bloc UCB de l'unité d'E/S et indique les adresses de certaines routines (appendages) qui traitent des cas d'erreur ou qui interviennent au cours de l'E/S; par précaution, ce bloc est protégé en mémoire (en AUK) pour éviter qu'il soit utilisé par un accès "pirate". Après l'ouverture, les données du fichier deviennent accessibles.

Inversement l'opération de fermeture (CLOSE) écrit les derniers enregistrements, la marque de fin de fichier, de nouvelles informations dans le DSCB du fichier disaue, et libère les blocs créés à l'ouverture (DEB).

D) L'accès aux données

Nous donnons ici le détail du traitement d'une entrée-sortie en XA ou ESA; il s'agit de l'accès aux données d'un fichier (mais la description peut être généralisée à tout type d'E/S).

Le fichier alloué à l'espace-adresse a été "ouvert" par la macro OPEN, qui a indiqué dans le DCB du fichier l'adresse de la routine méthode d'accès. Chaque ordre de lecture (GET) ou d'écriture (PUT) provoque un branchement à cette routine.

La méthode d'accès est chargée de construire les CCWs du programme canal; elle prépare l'appel à l'EXCP en construisant un ECB (voir chapitre 2F) et un bloc IOB (input/ouput block), qui répertorie toutes les données nécessaires à l'E/S (DCB, UCB, programme canal, ECB). L'EXCP est invoqué par la macro EXCP qui génère une SVC 0.

L'EXCP (ou un autre IOS driver) est un interface entre la méthode d'accès et l'IOS: il contrôle la validité de la demande adressée par la méthode d'accès; il traduit les adresses virtuelles en adresses réelles et fixe les tampons (buffers) d'E/S. Il crée un bloc IOSB (I/O supervisor block) qui décrit la demande à l'IOS ainsi qu'un SRB. La demande est enfin adressée à l'IOS par une macro STARTIO, qui provoque un SCHEDULE du SRB.

Le contrôle est ensuite rendu à la méthode d'accès, qui peut décider de se mettre en attente du résultat de l'E/S par un WAIT sur l'ECB (la tâche devient ainsi inactive) ou de repasser directement le contrôle au programme appelant qui s'en chargera lui-même (méthode d'accès basique). Au niveau de l'espace-adresse, la demande d'E/S est complètement formulée.

L'IOS met la demande dans la file d'attente correspondant à l'UCB de l'unité visée (UCB IOQ), la décrit par un ORB (operation request block) et lance l'opération par l'instruction SSCH (start subchannel) si le sous-canal représentant l'unité est disponible (autrement dit: il n'y a pas d'E/S en cours sur l'unité). A ce point du traitement, MVS en a terminé avec la demande d'E/S. Le matériel (le sous-système canal) gère une file d'attente des demandes par LCU (unité de contrôle logique); il choisit un chemin d'accès disponible, lance

l'E/S, effectue le transfert des données, signale le bon déroulement de l'opération dans le SCHIB (subchannel information block) qui décrit l'état de l'unité, et dans l'IRB (interrupt response block), et finalement signale la fin de l'opération en interrompant un processeur.

L'interruption est traitée par le gestionnaire d'interruption de premier niveau pour les E/S (I/O FLIH), puis par celui de second niveau (I/O SLIH). Ce dernier analyse le résultat de l'E/S et charge (par SRB) le composant IOS POST STATUS de signaler à l'EXCP la fin du traitement. S'il y a encore des demandes d'E/S pour l'unité, le SLIH les lance tout de suite (redrive). Enfin il teste la présence de nouvelles interruptions d'E/S par une instruction TPI (test for pending interrupt) de manière à les traiter tout de suite, sans retour au mode interruptible (nécessaire, en principe, pour les détecter). Le TPI a été introduit en XA pour pouvoir traiter une interruption d'E/S tout en restant en mode "disable", ce qui permet une certaine économie d'instructions.

A plusieurs reprises au cours du traitement de l'interruption, il y a possibilité de passer le contrôle à des exits de l'EXCP (qui peuvent eux-mêmes "remonter" à des "appendages" de la méthode d'accès) pour des traitements supplémentaires .

La partie "post status" de l'IOS rend le contrôle à l'EXCP, qui signale la fin d'E/S à la méthode d'accès par un POST ECB.

La figure montre l'interdépendance des composants qui prennent en charge l'E/S autant que le parallélisme des opérations, avec l'emploi des SRBs et le passage fréquent par le distributeur (lors de WAIT, POST, SCHEDULE SRB).

E) Les méthodes d'accès

1) GENERALITES

La méthode d'accès est l'interlocuteur que choisit un programme pour accéder aux périphériques; elle permet au programme de disposer d'une interface d'emploi aisé et indépendante du support externe.

Nous nous intéresserons plus particulièrement aux méthodes d'accès pour fichiers et nous laisserons de côté les méthodes d'accès pour les télécommunications (QTAM, BTAM, TCAM et surtout VTAM) qui mériteraient un autre ouvrage à elles seules.

Dans les méthodes d'accès pour fichiers on peut distinguer de prime abord les méthodes "conventionnelles", les plus anciennes, héritées de l'OS, et VSAM (virtual storage access method) apparu avec la mémoire virtuelle (OS/VS1) en 1972.

Une méthode d'accès est définie par la combinaison:

* d'une technique d'accès;

* d'une organisation de fichier.

Deux techniques d'accès existent:

* **la technique "basique"** (basic access): la plus simple, qui n'anticipe pas sur la séquence future de traitement des enregistrements. Le fichier est lu ou écrit par blocs (macros READ, WRITE), à charge pour le programme de définir le "blocage" (le nombre d'enregistrements à mettre dans un bloc). La méthode d'accès rend le contrôle au programme avant que l'E/S soit terminée; le programme doit vérifier la fin de l'E/S par une macro WAIT ou CHECK. Exemples de méthodes basiques: BSAM, BDAM, BISAM, BPAM pour les fichiers; BTAM pour les télécommunications.

* **la technique "queued"** (queued access): le système anticipe les traitements et lit ou écrit plusieurs blocs par opération d'E/S. Cette technique est adaptée aux lectures ou écritures séquentielles, pour lesquelles il y a de

fortes chances que le (n + 1) ième enregistrement soit traité une fois que les n premiers l'ont été. Les macros utilisées sont GET et PUT; le programme reçoit le contrôle une fois l'E/S terminée. QSAM et QISAM sont des exemples de méthodes utilisant la technique "queued".

Les organisations de fichiers possibles sont les suivantes:

* **séquentielle**: on ne peut accéder au (n+1)ième enregistrement avant d'avoir lu les n premiers. Une telle organisation est la seule possible sur des supports physiques tels que les bandes ou les cassettes. Cette organisation est adaptée aux traitements par lots traditionnels dans lesquels on accède à de gros volumes de données et pour lesquels l'ordre de traitement des enregistrements importe peu. Exemples: QSAM, BSAM, le VSAM ESDS.

* **directe**: on peut accéder directement à un enregistrement si l'on connaît son emplacement sur le support d'accès direct (disque). Il faut qu'une telle relation entre le "contenu" et le "contenant" puisse exister, ce qui est loin d'être évident. On peut maintenir une table de référence ou employer une technique d'adressage indirect: chaque enregistrement est repéré par une "clé" unique à partir de laquelle une relation mathématique plus ou moins complexe fournit l'emplacement disque. L'emplacement de l'enregistrement sur le disque peut être défini relativement au début du fichier: par le numéro d'ordre du bloc, ou par le "TTR" (nombre de pistes et d'enregistrements qui précèdent celui qu'on recherche); avec l'organisation RRDS de VSAM il suffit de connaître le numéro relatif d'enregistrement. L'emplacement peut être aussi défini par une adresse disque absolue "CCHHR" (numéro de cylindre, de piste et d'enregistrement); cependant cela rend le programme dépendant du type de support disque et de l'emplacement du fichier sur le disque (le fichier devient "unmovable"). Voir le paragraphe G pour la question de l'adressage des données sur disque.

* **séquentiel indexé**: les enregistrements sont repérés par des "clés". L'accès à l'enregistrement voulu se fait par consultation d'un "index", qui est une table de correspondance clé/emplacement ordonnée par clé et maintenue par la méthode d'accès. L'index est consulté de manière séquentielle ou plus directement par des niveaux d'index supérieurs.

* **l'organisation partitionnée** est une organisation particulière de fichier sur disque à mi-chemin entre le séquentiel et le direct: le fichier est constitué de "membres" stockés les uns après les autres dans l'ordre d'arrivée; le début du fichier est occupé par un répertoire (directory) qui est un index de correspondance nom de membre/TTR du membre permettant d'accéder directement au membre souhaité. Le membre en lui-même est ensuite traité comme un fichier séquentiel.

Le tableau suivant indique les méthodes d'accès fichier utilisées en MVS:

	queued access	basic access	VSAM
séquentiel	QSAM	BSAM	ESDS
séq. indexé	QISAM	BISAM	KSDS
direct	-	BDAM	RRDS
partitionné	-	BPAM	-

Nous nous intéresserons par la suite aux plus courantes: QSAM, BPAM, ISAM (d'un intérêt plutôt historique) et VSAM.

2) TECHNIQUES UTILISEES

Les méthodes d'accès emploient certaines techniques pour améliorer les performances des opérations d'E/S.

a) l'emploi des tampons (buffers) comme zones de travail temporaires durant les E/S permet d'optimiser les traitements de fichiers séquentiels. Un tampon est une zone de mémoire virtuelle dont la taille correspond à celle des blocs du fichier. En QSAM, 5 tampons par défaut sont disponibles pour chaque fichier. Avec l'emploi

des tampons et le chaînage des CCWs associés à ces tampons (chained scheduling), on peut lire ou écrire en une seule opération autant de blocs que de tampons.

BSAM (qui, lui, n'anticipe pas les demandes d'E/S) emploie un chaînage dynamique obtenu en reprenant le contrôle par un appendage lors de l'interruption d'E/S .

b) Les méthodes d'accès basiques rendent le contrôle au programme émetteur de l'E/S avant que celle-ci soit terminée; cela permet au programme d'utiliser le processeur pour d'autres traitements en attendant le résultat de son E/S. En pratique, on réalise le recouvrement (overlap) du temps processeur et du temps d'E/S en ayant toujours "deux fers au feu": une E/S terminée et dont on exploite les résultats, une autre en cours d'exécution pendant ce temps .

c) QSAM peut effectuer plusieurs lectures de fichiers en parallèle: un seul ordre GET suffit pour obtenir un enregistrement de chaque fichier. On peut ainsi "rapprocher", comparer, ordonner en même temps n fichiers différents.

d) La méthode d'accès VSAM offre de nombreuses options de performance, plus liées à l'organisation des données sur le disque qu'à la technique d'accès elle-même.

e) ASM, gestionnaire de mémoire auxiliaire, emploie la technique particulière du suspend/resume pour accélérer les E/S de pagination. Le positionnement du bit "suspend" du CCW indique un arrêt (mais non la fin) du programme canal; le CCW peut ensuite être modifié et l'envoi d'une instruction "resume subchannel" (resume I/O en 370) relance l'exécution du programme canal. Cette technique permet à ASM de relancer une E/S plus rapidement sur le disque, sans passer par l'IOS. Ce bénéfice est perdu si d'autres fichiers sont accédés sur le même disque: I'E/S "suspendue" doit alors être resoumise de la façon habituelle (macro STARTIO).

IBM a annoncé en 1989 une nouvelle méthode d'accès d'E/S "parallèles" (parallel l/O access method) destinée aux installations pour lesquelles le temps de réponse E/S est crucial (applications temps réel par exemple); avec cette méthode les enregistrements consécutifs d'un fichier sont lus ou écrits en parallèle sur n disques (255 au maximum); d'une telle dispersion des informations, inédite en MVS, découlent des améliorations significatives des performances.

3) ÉTUDE DE QUELQUES ORGANISATIONS

a) Les fichiers séquentiels

Le format d'un fichier correspond à la façon de stocker physiquement les enregistrements; les formats possibles sont F (fixe), V (variable), U (indéfini) et D (données ASCII).

En format fixe, les enregistrements ont tous la même longueur et un bloc contient un nombre constant d'enregistrements (le "facteur de blocage").

En format variable les enregistrements et les blocs sont de longueur variable; l'intérêt est d'économiser de l'espace magnétique en tenant compte de la longueur réelle des données stockées. Chaque enregistrement est précédé d'un RDW (record descriptor word) constitué d'un demi-mot indiquant la longueur de l'enregistrement, suivi d'un demi-mot à zéro binaire. De même chaque bloc est précédé d'un BDW (block descriptor block) dont le premier demi-mot indique la longueur du bloc.

Le format variable spanné permet de stocker un enregistrement "à cheval" sur plusieurs blocs; on peut ainsi outrepasser la limitation portant sur la taille du bloc (pas plus de 32 760) et employer des tailles d'enregistrements supérieures. La portion d'enregistrement contenue par un bloc s'appelle un segment, et le RDW devient un SDW (segment descriptor word). Le second demi-mot du SDW est utilisé pour indiquer s'il s'agit du premier segment, du dernier, d'un segment intermédiaire ou de l'unique segment qui constitue l'enregistrement.

Avec le format indéfini le système ne se soucie pas de la taille des enregistrements et ne voit plus que des blocs.

Avec le format D on peut traiter des données stockées à l'origine sur bande dans le code à 7 bits ASCII; la traduction en EBCDIC est faite de façon transparente par la méthode d'accès.

Un fichier séquentiel est le plus souvent lu ou écrit en une seule fois, enregistrement après enregistrement; toutefois une possibilité de mise à jour "en place" existe (sur support à accès direct), ainsi qu'une possibilité d'extension d'un fichier existant par ajouts en fin de fichier (DISP=MOD ou OPEN avec option EXTEND).

QSAM et BSAM emploient tous deux la bufferisation des données dans des pools de tampons. En outre QSAM peut copier l'enregistrement dans la zone indiquée par le programmeur (move mode, data mode) ou seulement indiquer l'adresse du prochain tampon dans le registre 1 (locate mode).

b) Les fichiers partitionnés

Un fichier partitionné (PDS, partitioned dataset) est constitué de 2 parties:

* **le répertoire** (directory) placé en tête du fichier, alloué en blocs de 256 octets; les entrées du répertoire sont classées par nom de membre et pointent sur les membres du PDS avec un adressage TTR (ce qui rend le PDS indépendant de son emplacement sur le support disque). Chaque entrée peut comporter des données utilisateur (user data) de 62 octets maximum; ces user data peuvent être utilisées pour stocker diverses informations: des statistiques de mise à jour de chaque membre (éditeur d'lSPF), des pointeurs vers les CSECTs pour un load-module. Un bloc de répertoire peut contenir une vingtaine d'entrées, bien moins si les user data sont utilisées (4 ou 5 avec l'éditeur d'lSPF par exemple);

* **les membres** du PDS sont stockés en séquentiel dans l'ordre d'arrivée, sans classement. Les ajouts de membres se font en fin de fichier, à la suite des autres membres, de même pour les modifications de membres existants (sauf pour le cas particulier de mise à jour en place), ce qui fait grandir toujours davantage le fichier puisque l'espace occupé par les anciennes versions de membres n'est pas récupéré automatiquement. Il faut à intervalles réguliers effectuer un "compress" du PDS par l'utilitaire IEBCOPY pour recopier les membres et éliminer l'espace inutilement occupé. C'est un inconvénient majeur de l'organisation, et il faut attendre DFSMS et DFP320 avec le PDS "extended" pour avoir enfin une bonne occupation de l'espace sans besoin de réorganisation. Entre autres contraintes, un PDS ne peut exister que sur un support à accès direct (disque) et ne peut s'étendre sur plusieurs volumes.

L'organisation partitionnée est une organisation semi-séquentielle combinant BPAM et BSAM: le répertoire est parcouru séquentiellement jusqu'à obtention de l'entrée décrivant le membre recherché, lequel peut être alors traité par BSAM (voir un exemple de programmation en annexe). La recherche dans le répertoire est moins performante que la recherche dans les index d'un fichier ISAM, et peut prendre un certain temps. L'allocation d'un membre de PDS demandée par la notation PDS (MEMBRE) permet de traiter le membre comme un fichier séquentiel ordinaire (par QSAM).

Les PDS représentent un type de fichier extrêmement utilisé en MVS, tant par le système (PARMLIB, PROCLIB, etc.) que par les utilisateurs (bibliothèques de sources de programmes, de JCLs, de modules, de textes, de paramètres, etc.).

Les fichiers partitionnés PDSE

Le PDSE, annoncé par IBM en 1989, lève enfin les nombreuses contraintes que subissait l'organisation partitionnée depuis son origine dans les années 60 : le besoin de récupérer l'espace inutilisé par COMPRESS, les données étant ajoutées systématiquement en fin de fichier (champ DS1LSTAR); la dépendance par rapport à l'architecture CKD, la géométrie du disque et le blocksize d'allocation (dont découle une plus ou moins bonne occupation de l'espace disque et tout l'aspect performances); la directory figée (impossible à étendre sauf en recréant un autre PDS) et traitée séquentiellement (le membre A sera toujours localisé plus vite que le membre Z); une taille limitée à 16 extents (comme tous les fichiers OS).

Un fichier PDSE est un fichier géré par SMS, d'une structure proche de celle du fichier VSAM linéaire (CI de 4K, allocation jusqu'à 123 extents). Les "trous" d'espace libre sont récupérés dynamiquement pour stocker les membres ; la directory est extensible et n'est plus localisée au début du fichier, comme pour le PDS.

Le PDSE bénéficie des apports d'ESA et de SMS (directory "cachée" par un dataspace, membres stockés dans un hiperspace, gestion par HSM). On peut envisager des mises à jour de membres simultanées.

La structure interne du PDSE est complètement différente de celle du PDS; elle n'est pas documentée autrement que par la mise en garde "IBM proprietary information". Cependant, il est intéressant pour le programmeur de savoir que le PDSE se comporte comme un PDS pourvu qu'on emploie les services standards BPAM ou les utilitaires IBM habituels (à part l'archaïque IEHMOVE). Le paramètre DSNTYPE=LIBRARY du JCL permet de créer un PDSE.

Le PDSE est un apport appréciable pour remplacer les PDS de très grande taille, subissant fréquemment des abends système x37 ou contenant de très nombreux membres, et provoquant des problèmes de performance. L'intérêt du PDSE s'est accru avec le support des modules exécutables ou "objets-programmes" par la partie gestion de programmes de MVS.

c) Les fichiers séquentiels-indexés

L'organisation ISAM a aujourd'hui beaucoup vieilli, mais elle est toujours supportée, bien qu'lBM recommande la conversion en VSAM. Elle a eu au moins le mérite de préfigurer ce qu'allait être l'organisation KSDS de VSAM.

Dans cette organisation, les enregistrements sont repérés par une clé et sont classés physiquement par clés croissantes. L'accès à un enregistrement passe par la consultation d'un index de piste qui indique la bonne piste en fonction de la clé fournie (chaque entrée de cet index repère la plus haute valeur de clé de chaque piste). Si la taille du fichier dépasse un cylindre, il y a création d'un index de cylindre qui remplit une fonction analogue à celle de l'index de piste (chaque entrée identifiant la plus haute clé connue dans un cylindre). On peut enfin employer un index maître pour consulter plus vite l'index de cylindre. De façon optionnelle, on peut réserver un certain espace de débordement ("overflow") pour l'ajout d'enregistrements qui ne pourraient plus tenir sur la piste destinataire; on évite de devoir réorganiser le fichier à cette seule fin.

Une interface particulière, l'IIP (ISAM interface program), permet à un programme écrit à l'origine pour traiter un fichier ISAM de pouvoir traiter un fichier VSAM de façon transparente.

ISAM, moderne dans les années 60, devrait de nos jours être proscrit : mauvaises performances (overflow traité en séquentiel), besoin de réorganisation fréquent, structure unmovable (adresses en CCHH).

d) L'organisation VSAM

Le besoin d'une méthode d'accès générale disposant d'options de performance, de partage s'est fait ressentir avec l'apparition de la mémoire virtuelle. Il fallait remédier à cette prolifération des méthodes d'accès OS qui aboutissait à créer des fichiers ne pouvant être manipulés que par certaines méthodes d'accès, d'autres par plusieurs, avec différents utilitaires à chaque fois. Cette méthode devait aussi assumer les fonctions des méthodes d'accès séquentielles, directes et indexées, tout en restant autant que possible indépendante du support physique, à la différence d'lSAM ou BDAM. Le fait que VSAM est la seule méthode d'accès disponible sur tous les systèmes 370 témoigne bien de son rôle "unificateur".

Voici de quelle façon les données sont organisées en VSAM:

* l'enregistrement logique est l'unité de base; sa taille est variable (il n'y a pas de RECFM en VSAM);

* le control interval (CI) est l'unité de transfert entre la mémoire et le disque; ce n'est pas l'unité de stockage de base, plus petite, qui reste le bloc physique (un CI peut être à cheval sur plusieurs pistes). Un CI est la combinaison de n enregistrements, d'un espace libre (free space) réservé pour les ajouts, d'informations de contrôle pour les enregistrements (RDF, record definition fields) qui permettent de trouver les longueurs des enregistrements et de réaliser après coup un découpage logique du CI, et enfin d'un CIDF (CI definition field) de 4 octets qui permet de retrouver la zone d'espace libre à l'intérieur du CI. Comme pour les fichiers traditionnels, on peut créer des enregistrements plus grands que le CI (spanned records). Muni de ses informations de contrôle, un CI est ainsi une unité cohérente nécessaire et suffisante au traitement de n enregistrements;

* la control area (CA) est un groupe de CIs. Sa taille représente un nombre entier de pistes mais ne dépasse pas un cylindre. L'existence de cette notion se justifie surtout pour les fichiers de type KSDS;

* un fichier VSAM est constitué d'un nombre entier de CAs;

* un cluster ("grappe") VSAM réunit les fichiers composants nécessaires au traitement: une partie "data" contenant les données brutes, une partie "index" permettant l'accès à la partie data (uniquement pour les KSDS);

* un "data space" (ne pas confondre avec les dataspaces d'ESA) est une aire de stockage VSAM sur disque destinée à contenir des fichiers ou des catalogues VSAM. Cette notion a disparu avec les catalogues ICF (voir chapitre 7).

Actuellement 4 types de fichiers VSAM existent:

* le **KSDS** (key-sequenced data set): organisation comparable à ISAM. Les enregistrements sont ordonnés par une clé située à un emplacement bien défini et qui est unique pour chaque enregistrement. Un index repère les plus hautes clés de chaque CI de données; comme pour ISAM on peut avoir plusieurs niveaux d'index pour accélérer la recherche, avec des pointeurs "horizontaux" et "verticaux" (voir figure 6). L'index de plus bas niveau, pointant immédiatement sur les données, est le "sequence set"; il est utilisé autant pour les accès directs (par clé) que pour les lectures séquentielles (il connaît seul la séquence logique des CIs). L'index ou les index figurent dans un fichier VSAM séparé, organisé de telle sorte que les entrées d'un CI décrivent les enregistrements d'un CA de la partie data.

L'espace libre existant dans le CI, qu'il résulte d'un choix avant le chargement du fichier, ou bien de destructions ou de raccourcissement d'enregistrements pendant leur mise à jour, est récupéré quand on insère de nouveaux enregistrements (dont la clé est dans la fourchette de clés du CI). Si un CI est plein, un ajout provoque un "split" (éclatement) du CI: le nouvel enregistrement et les données du CI de clé supérieure partent dans un CI libre du CA. S'il n'y a pas de CI libre dans le CA, le split de CI est précédé d'un split de CA pour dégager un CI libre dans le CA visé (par copie d'un certain nombre de CIs dans un CA inoccupé). Par cette manière de procéder on évite de gérer des zones de débordement comme en ISAM; cependant le nombre de splits de CI ou de CA doit être modéré pour ne pas impacter les performances (il s'agit de mouvements de données improductifs);

* **I'ESDS** (entry-sequenced data set): il s'agit d'une organisation séquentielle, sans clé (stockage dans l'ordre d'arrivée). Les ajouts ne peuvent se faire qu'en fin de fichier; les destructions d'enregistrements ne sont pas effectuées (la place n'est pas récupérable automatiquement); les modifications doivent se faire en place sans possibilité d'augmenter la taille d'un enregistrement;

* le **RRDS** (relative record data set): on accède directement à un enregistrement en précisant son numéro d'ordre dans le fichier; un enregistrement occupe une case prédéterminée (un "slot") de longueur fixe. Il n'y a pas de problème pour ajouter ou détruire des enregistrements: une case correspondant à un numéro d'ordre est libre ou occupée, c'est tout;

* le **fichier linéaire**, (LDS, linear data set): avec cette organisation récente, il n'y a plus de notion d'enregistrement. Le fichier est une longue chaîne d'octets découpée en CIs de taille 4096 (correspondant à une page de mémoire, ce qui n'est pas un hasard...) sans informations de contrôle. Un nouveau concept, le DIV (data in virtual) est apparu en MVS/XA version 220: le but est d'améliorer les performances en évitant les E/S

inutiles. Le DIV rappelle plus une technique de pagination qu'une méthode d'accès classique. Le fichier LDS est considéré en quelque sorte comme un fichier de pagination "privé"; il est traité comme s'il était déjà entièrement en mémoire virtuelle: pour le programmeur les lectures ou écritures se ramènent à des instructions MVC. Il n'y a pas de tampons; seuls les CIs lus ou modifiés peuvent être présents en mémoire; seuls les CIs modifiés peuvent être recopiés sur disque en fin de traitement. Étant donné qu'il n'y a ni clé ni numéro relatif (en cela le
fichier linéaire est proche de l'ESDS), le programmeur est responsable de l'organisation des données et de la façon d'y accéder directement, s'il y a lieu.

Quelques exemples peuvent illustrer les différences entre les types de VSAM: un fichier "personnel" d'une entreprise pourrait être un KSDS (accès par nom de personne ou matricule); tout fichier dont la "clé" serait un numéro pourrait être un RRDS (par exemple: fichier des départements français); enfin tout fichier pour lequel un accès séquentiel est suffisant peut être un ESDS. DB2 et DL/1 emploient respectivement les fichiers linéaires et les ESDS pour leurs structures de bases de données, en gérant leurs propres informations de contrôle pour pouvoir réaliser tout de même des accès directs.

Un langage de commande est fourni pour la gestion courante des fichiers VSAM (certaines commandes conviennent aussi pour les non-VSAMs): il s'agit d'AMS (access method services) et de l'utilitaire IDCAMS. Les principales commandes sont: DEFINE (création ou catalogage de fichier), DELETE (destruction ou décatalogage), REPRO (copie d'un fichier sur un autre), PRINT (impression du contenu), LISTCAT (listage des informations du catalogue), ALTER (modification des caractéristiques d'un fichier), EXPORT et IMPORT pour transfert de fichiers entre systèmes.

Voici les points en lesquels à notre avis VSAM diffère essentiellement des autres méthodes d'accès:

* la gestion de l'espace disque: la réservation d'espace libre dans chaque CI et dans chaque CA permet l'insertion de nouveaux enregistrements dans un KSDS sans passer obligatoirement par le mécanisme du CI split ou CA split; I'espace occupé par l'index est réduit par une "compression" des clés qui élimine les caractères redondants. Par ailleurs un fichier VSAM peut occuper jusqu'à 123 extents sur un disque (contre 16 pour les fichiers ordinaires) et peut s'étendre sur des volumes "candidats" supplémentaires;

* les tampons et blocs de contrôle de VSAM peuvent presque tous résider au-dessus de la "barre" des 16 Méga-octets; ce n'est pas (encore) le cas pour de nombreuses autres méthodes d'accès;

* les options de performance pour les KSDS: chaque enregistrement de l'index peut être copié en autant d'exemplaires que possible pour occuper une piste entière (option REPLICATE): ainsi les délais de rotation du disque sont réduits quand l'index est consulté. Avec l'option IMBED, le sequence set, index de plus bas niveau, est copié en plusieurs exemplaires sur une piste précédant immédiatement le CA data qu'il décrit, ce qui réduit tant les mouvements de bras que le délai rotationnel. La répartition des données sur plusieurs disques améliore aussi les performances: on peut ainsi séparer les composants data et index; on peut encore répartir les data sur plusieurs volumes en fonction de fourchettes de valeurs de clés (key ranges).

Les enregistrements des KSDS et ESDS et les CIs des LDS peuvent être traités en accès direct par RBA (relative byte address): il s'agit d'un adressage en octets comptés à partir du début du fichier, qui est ainsi indépendant de l'emplacement du fichier sur le disque (comme le TTR).

Un fichier KSDS peut être traité selon une séquence autre que celle qui résulte de la clé de base (clé primaire). Un nouveau fichier, "alternate index" (c'est un KSDS) doit être créé pour refléter ce nouvel ordre sans introduire de redondance de données. Cet index secondaire peut être maintenu automatiquement par VSAM (upgrade set) quand le cluster de base est modifié; il peut aussi être introduit sur un ESDS de manière à avoir un accès direct sur une organisation séquentielle à l'origine.

Comme pour les méthodes d'accès basiques, une possibilité d'E/S asynchrone existe, avec l'emploi d'une macro CHECK pour vérifier la fin de l'E/S. Dans le même ordre d'idées, on peut traiter un fichier VSAM au niveau du CI plutôt que de l'enregistrement (CI access ou "improved CI access").

* les possibilités de partage sont les suivantes:

- partage des tampons et blocs de contrôle d'E/S: les possibilités sont au nombre de trois:

. Ie NSR (no shared resource) est l'option par défaut: pas de mise en commun de ces "ressources" entre plusieurs tâches;

. Ie LSR (local shared resources) permet un partage au niveau d'un espace-adresse: des pools de "ressources" pour les data et pour les index peuvent être créés. Les tampons de données sont consultés avant d'effectuer les E/S sur disque; les données écrites restent dans les tampons le plus longtemps possible pour être encore disponibles sans E/S. Le LSR est intéressant pour les traitements directs; CICS et les autres gestionnaires de hases de données en font un usage fréquent;

* Ie GSR (global shared resources) permet une mise en commun des ressources entre des espaces-adresses ayant une clé de protection système (0 à 7); un pool est créé en CSA à cette fin;

- partage d'accès: un programme peut accéder en même temps, de différentes façons à un fichier VSAM, et être "positionné" en plusieurs endroits du fichier ("multistring": la notion de "string" désigne un accès avec positionnement);

- le partage d'un fichier VSAM accédé indépendamment par plusieurs utilisateurs (cross-region) ou plusieurs systèmes (cross-system) est régi (et restreint) en fonction d'options de partage (share-options) qui interviennent à l'OPEN du fichier. L'option 1 assure une intégrité du fichier en lecture et en écriture: plusieurs utilisateurs peuvent lire le fichier tant qu'il n'y a pas d'accès en écriture; sinon un seul utilisateur peut effectuer des mises à jour. L'option 2 ne garantit qu'une intégrité en écriture (une seule écriture et plusieurs lectures permises). L'option 3 permet tout accès et ne garantit aucune intégrité (laissée à la charge de l'utilisateur). De même pour l'option 4, qui de plus entraîne un rafraîchissement des tampons pour les accès directs, ce qui procure une certaine intégrité en lecture (une intégrité plus complète est obtenue en DOS/VSE).

Pour le "cross-system" seules les options 3 et 4 existent; VSAM n'assure explicitement aucune intégrité. L'utilisateur en est complètement responsable; les méthodes à employer sont à base de macros RESERVE et de GRS (pour éviter appropriation de tout le volume disque). Un garde-fou est toutefois fourni en standard avec l'option de cross-system 4, avec laquelle VSAM interdit tout split de CA dans un KSDS et empêche l'extension du fichier.

Ces options de partage sont assez limitatives : rien n'est prévu pour partager en mise à jour (en gardant l'intégrité des données) un fichier entre plusieurs espaces-adresses. On doit avoir recours pour cela à des SGBD tels que IMS ou DB2. Il est plus aisé d'organiser ce partage entre sous-tâches d'un même espace-adresse (par exemple avec le LSR et l'option de partage 2, ou le NSR avec l'option 4).

"Optimiser VSAM" peut devenir nécessaire dans certains cas (habituellement on garde les options prises par défaut lors du DEFINE CLUSTER). Voici quelques idées directrices valables dans la plupart des cas :

- surveiller le nombre de CI- et CA-splits des KSDS : en cas d'excès, revoir la quantité d'espace libre (freespace), réorganiser les fichiers (REPRO);

- la taille de CI peut être choisie pour être selon les cas la plus grande possible (fichiers traités séquentiellement) ou au contraire assez petite (accès directs);

- disposer d'un grand nombre de buffers est le moyen de diminuer le nombre d'E/S : paramètres BUFND, BUFNI ou BUFSP pour le NSR séquentiel, LSR avec plusieurs pools bien taillés (CICS, IMS), "batch LSR" pour certains travaux effectuant beaucoup d'accès directs (par exemple SMP/E);

- l'architecture ESA a apporté plusieurs dispositifs de "data in memory" qui doivent être considérés : la possibilité d'allouer des tampons en hiperspace ; hiperbatch ; les fichiers linéaires et les services de "data windowing" ; les "data tables" de CICS (petits fichiers très sollicités en LSR), etc.

e) Les entrées-sorties virtuelles (le VIO)

Le VIO (virtual input/output) est un des premiers moyens inventés par IBM pour remplacer les E/S disque par des accès en mémoire (le concept DIM). Il permet d'exploiter des fichiers de travail (temporaires, dont la durée de vie ne dépasse pas le temps d'exécution) par une méthode de pagination plutôt que par des transferts de données sur disque. Le composant EXCP n'invoque pas l'IOS; les données passent dans un tampon de l'espace-adresse, la "fenêtre VIO". Il s'agit d'une zone de mémoire virtuelle d'une taille égale à une piste de disque (ce qui revient à 4 pages pour un équivalent de disque 2305 ou 3330). Quand cette "piste" virtuelle est pleine, les pages sont transférées sur le fichier de pagination par le mécanisme de page-out; elles reviendront à la demande dans la fenêtre quand elles seront référencées (faute de page). Le VIO offre d'excellentes performances par rapport à l'usage de fichiers temporaires habituels sur disque: on évite le passage par les routines d'allocation sur disque et on allège grandement les opérations d'E/S, une faute de page représentant bien moins d'instructions à exécuter qu'une E/S.

Le VIO est très recommandé pour les fichiers de travail de petite taille (utilitaires de compilation, de copie) ; la vocation des fichiers de pagination dans ce cas précis n'est pas de servir de disques de travail, mais d'améliorer les performances d'E/S (l'ingénieur-système peut par précaution empêcher le VIO d'utiliser certains fichiers de pagination pour ne pas impacter les performances des autres travaux). SMS permet d'allouer systématiquement en VIO (dans un storage group "VIO") tous les fichiers temporaires de taille inférieure à une limite fixée.

F) La définition de la configuration

Une configuration de périphériques consiste en un certain nombre de canaux, de contrôleurs et d'unités d'E/S; une description doit en être fournie au système. Avant même que MVS soit actif, l'ordinateur doit être capable d'identifier les unités disponibles (par exemple le disque système MVS et une console système pour démarrer MVS) et les voies physiques pour y accéder. De même MVS et son IOS ensuite pour diriger les E/S.

L'IOCP (input/output configuration program) est un utilitaire IBM destiné à copier cette description sur des fichiers spéciaux intégrés à l'ordinateur 308X ou 3090, les IOCDS (I/O configuration datasets). Ces fichiers seront lus lors de l'IML (initial microcode load) et chargés en zone HSA. Il y a pour les 308X et 3090 quatre IOCDS par side. L'IOCP existe en version autonome (stand-alone, obligatoire pour les 4381), ainsi que sous VM et sous MVS (programme ICPIOCP pour les 308X, IOPIOCP pour les 3090, IXPIOCP avec ESCON). Les principaux ordres de contrôle sont les suivants:

* CHPID: indique le numéro de chpid (channel path identifier); il s'agit du canal physique: par exemple 00 à 2F sur un 3090 à 48 canaux. En 370, on doit préciser de plus à quel canal logique (0 à F) on rattache le chpid (MVS/370 ne connaît que 16 "canaux") et de quel "channel set" il fait partie (il s'agit du groupe de canaux dédiés à un processeur: groupe 0 en uniprocesseur, 0 ou 1 en biprocesseur). A partir de XA, la notion de "canal" n'existe plus dans le sens qu'elle avait en 370, et il est plus correct de parler de "chpid".

On indique aussi le type de canal: "byte multiplex", pour lequel des périphériques "lents" (terminaux, imprimantes) sont choisis à tour de rôle pour effectuer un transfert de quelques octets; "block multiplex", adapté aux unités "rapides" (à disques ou bandes), qui restent en connexion avec le canal le temps de transférer un bloc de données (la vitesse de transfert sur de tels canaux atteint au minimum 3 ou 4,5 millions d'octets à la seconde). Avec ESCON apparaissent les canaux "ESCON convertor" (TYPE=CVC) ou les canaux purement ESCON (TYPE=CNC).

Enfin, avec le partitionnement logique (LPAR) de PR/SM, on peut indiquer à quelle partition logique de machine on rattache le chpid.

* CNTLUNIT: définit une unité de contrôle, son type, le ou les chpids de connexion, les numéros d'unités qu'elle "adresse" physiquement.

* IODEVICE: définit une unité, les contrôleurs auxquels elle est connectée, le modèle d'unité, l'adresse. L'adresse d'une unité est un numéro composé de 3 digits hexadécimaux (000 jusqu'à FFF) identifiant de manière unique une unité dans une configuration. En 370, le premier digit correspond au canal (0 à F), les deux suivants sont fonction du branchement sur l'unité de contrôle. En XA au contraire l'adresse est arbitraire;

en particulier, le premier digit ne correspond plus au canal; les deux derniers peuvent ne pas correspondre à l'adresse de branchement, qui dans ce cas doit être rappelée par un paramètre UNITADD.

L'adresse d'unité d'E/S en XA est un numéro conventionnel (device number), c'est celui de l'UCB (unit control block) en MVS. L'information véritablement utile vis-à-vis du matériel pour les E/S est le numéro de sous-canal (subchannel, voir paragraphe A). Au démarrage de MVS la correspondance est établie entre le numéro d'UCB à 3 digits, seul connu des pupitreurs, et le numéro de sous-canal, seul utilisé pour lancer l'E/S, à 4 digits. Ces dispositions permettent d'envisager des configurations plus vastes que celles dont on pourrait disposer en 370 pur.

Le numéro d'UCB a été limité à 3 octets plutôt qu'à 4 (comme en VM) uniquement pour distinguer les allocations par nom générique (UNIT = 3380) des allocations par adresse (UNIT=330, device number).

L'IOCP en XA établit des unités de contrôle logiques (LCUs). Une LCU correspond à un groupe d'unités de contrôle physiques de mêmes caractéristiques se partageant un certain nombre d'unités d'E/S. Comme indiqué au paragraphe D, le sous-système de canaux gère les demandes d'E/S par LCU; connaissant toutes les unités de contrôle conduisant à l'unité voulue, il peut choisir un chemin disponible pour répondre à la demande d'E/S. On peut indiquer dans l'IOCP un canal préféré pour accéder à une unité. MVS-XA, quant à lui, ne peut influencer le choix du chemin; tout au plus peut-il communiquer au matériel la liste (path mask) des chemins (chpids) à employer pour effectuer l'E/S (un chemin a pu être activé ou désactivé à cause d'une erreur matérielle ou d'une intervention du pupitreur par la commande MVS VARY PATH).

La configuration doit aussi être décrite à MVS, dans un processus d'lOGEN (génération d'E/S) ou de MVSCP (MVS configuration program). L'IOGEN est un procédé assez lourd de génération (voir chapitre 13) qui provoque une mise à jour de plusieurs tables ou modules sur le disque système (fichiers LPALIB et NUCLEUS notamment) et y installe les modules MVS supportant les unités de la configuration. Le MVSCP, qui remplace l'IOGEN à partir de MVS-XA 220, simplifie la tâche: il ne modifie pas le code de MVS et se contente de créer 4 tables dans le fichier NUCLEUS: IEANCTxx, table des consoles pour l'IPL; IOSUCBxx, table des adresses; IEFEDTxx, table des unitnames; IOSllTxx, table indiquant les modu- les de support d'unités à charger à l'IPL. "xx" est un numéro qui identifie la configuration; on peut disposer aisément de plusieurs configurations, le numéro pouvant être précisé en paramètre à l'IPL s'il n'a pas la valeur par défaut (00).

L'équivalent en VM de l'IOGEN et du MVSCP est le HCPRIO. Il y a, comme pour MVS, rapprochement à l'IPL des numéros d'unités (rdev, real device) et des numéros de sous-canaux.

Le source utilisé pour l'IOCP peut servir en principe à l'assemblage en MVS de l'IOGEN ou du MVSCP, avec quelques ajouts:

* un ordre CONSOLE décrit les consoles MVS, utilisées par le pupitreur pour communiquer avec MVS; il est recommandé de préciser des consoles alternées, utilisées par MVS en cas de défaillance de la console maître (master console). Aux consoles sont affectés des codes de routage (routcodes) liés aux grandes fonctions d'une exploitation informatique (gestion de bandes, de disques, de réseau, de la sécurité, etc.) ou à l'importance du message (pour action, pour information, etc.); ces codes permettent de limiter l'affichage des messages et de spécialiser chaque console;

* l'ordre IODEVICE est le même que pour l'IOCP, avec de plus un paramètre FEATURE pour indiquer notamment si l'unité est susceptible d'être partagée (SHARED) par plusieurs systèmes: ce paramètre est examiné par MVS pour déterminer s'il y a lieu de prendre certaines précautions pour accéder à une unité (cas des disques). Il s'agit du RESERVE qui sera examiné au paragraphe suivant;

* en 370 les canaux sont décrits par des macros CHANNEL indiquant le type et le numéro de canal (0 à F);

* les ordres UNITNAME enfin établissent des groupes d'unités et les désignent par des noms "ésotériques". On utilise ces noms dans le paramètre UNIT du langage de commande JCL pour diriger l'allocation des unités. On a la possibilité d'utiliser des noms plus "parlants" que les noms génériques d'unités (3380,3420, etc.), et de fixer ces noms une fois pour toutes, la relation entre les noms et les adresses d'unités qu'ils représentent étant à la charge de l'équipe système. On peut en même temps réaliser des groupements d'unités en fonction de leur

emploi: un groupe de disques pourra être réservé à l'exploitation, un autre au développement applicatif, etc. Des exemples courants de tels noms ésotériques suivent:

* TAPE pour les unités dérouleurs de bande ou lecteurs de cassettes.

* DASD ou SYSDA pour les unités disques.

* VIO pour les fichiers temporaires "virtual I/O". Pour le VIO, il faut "raccrocher" le nom ésotérique à une adresse d'unité disque factice (par exemple 2305 ou 3330) et indiquer VIO=YES.

Les unitnames peuvent être modifiés sans avoir à refaire une IOGEN ou un MVSCP complets: on parle alors d'EDTGEN (eligible device table generation).

G) La gestion des supports magnétiques

1) Le point sur les périphériques de stockage sous MVS

Le marché des matériels de stockage est au moins aussi important en valeur financière que celui des ordinateurs, avec un accroissement des besoins de 20 à 30 % l'an. Les décisions sont souvent difficiles à prendre si l'on veut tenir compte des offres compatibles, de l'évolution des techniques, des besoins de performance, de la surface au sol nécessaire, de l'obsolescence prévisible et de l'état du marché à un moment donné, etc. Un examen des matériels disponibles montre qu'à tout niveau de la "hiérarchie" des mémoires une offre existe pour des coûts, capacités et performances cohérents entre eux et répondant aux besoins exprimés.

* Les disques électroniques (SSD, unités à semi-conducteurs), introduits sur le marché vers 1976, sont des supports de stockage à mémoire le plus souvent non volatile (en cas de panne, les données
sont conservées par une batterie de secours ou même sauvegardées sur un disque magnétique). Le temps d'accès constant (autour de 1 ms) le destine idéalement à la pagination et à la conservation de fichiers de contrôle très accédés (checkpoint de JES, base RACF, etc.) ou même aux bases de données. Certaines unités permettent de stocker 1 GO avec un taux d'utilisation dépassant 1000 E/S par seconde. IBM ne vend pas de SSD sous le motif que les autres supports (MAP, contrôleur à cache) rendent un service équivalent.

* Les disques magnétiques sont basés sur la technologie Winchester, lancée par IBM en 1973 avec l'unité 3340. Les disques de référence aujourd'hui sont les 3380 et les 3390.

* Les contrôleurs de disques à cache appliquent le classique principe de localité pour conserver en mémoire les données des disques. La fonction de cache en lecture est complétée sur les contrôleurs récents (3990-3 d'IBM) par des "fonctions avancées" qui améliorent fortement les performances de traitement des fichiers sur disque. Le CFW (cache fast write), invoqué par exemple par le tri DFSORT, place les fichiers temporaires directement en mémoire cache ; le DFW (dasd fast write), cache en écriture, raccourcit les temps d'E/S en stockant les données dans une mémoire non volatile intermédiaire, la NVS (le transfert sur le disque étant déclenché ensuite de façon asynchrone) ; le dual copy et le fast dual copy accroissent la disponibilité des données par copie sur un disque miroir. Les fonctions du 3990-3 s'étendent avec la possibilité de sauvegarde online (concurrent copy), la gestion dynamique du cache, etc.

* Les cassettes de type 3480, 3490 et 3490E constituent un support robuste, bon marché, peu encombrant et de grande capacité (jusqu'à 2,4 GO avec IDRC, une plus grande longueur de bande et l'enregistrement sur 36 pistes au lieu de 18).

* Les disques optiques procurent une grande capacité de stockage accessible directement par un système de "juke-box" de faible encombrement avec un temps de réponse correct (à l'échelle humaine). Leur coût ne permettra vraisemblablement pas de remplacer le support bande ou cassette. L'application typique qui peut être envisagée avec ce support est la consultation de documents texte ou image en remplacement du papier ou des microfiches. L'accès aux disques optiques sous MVS s'effectue par un logiciel sous-système spécifique ("OAM"

pour IBM), ou par une émulation bande ou disque ; l'émulation disque est le moyen le plus commode (exemple du système optique IBM 3995-151 qui est "vu" comme des disques 3390-2).

* Les disques "en grappe", ou RAID (redundant arrays of independant disks), sont des moyens de stockage étudiés depuis 1987, à base de plateaux "standards" (par exemple 5,25") avec un grand nombre de têtes de lecture, offrant rapidité de transfert et tolérance de panne (des disques de parité permettent de reconstituer les données en cas de problème physique). Les données sont "éclatées" sur les différents disques par bit, octet ou bloc (selon les niveaux de RAID). La librairie ICEBERG de StorageTek est un exemple original de stockage de masse à base de disque RAID.

* Les mémoires de masse : ce terme désigne un ensemble assez disparate de moyens de stockage combinant automatisme et grande capacité. IBM lui-même avait ouvert la voie en 1975 avec le 3850, une unité vendue à quelques centaines d'exemplaires (abandonnée aujourd'hui) qui permettait de gérer un ensemble de cassettes de conception spécifique (chacune de capacité 50 MO) et stockées dans une structure en alvéoles ; le montage d'une cartouche et son transport à la station de lecture/écriture demandait quelques secondes, les données étant ensuite transférées sur disque (staging). En 1987 le constructeur Storagetek lance une bandothèque 4400, un "silo" qui stocke des
milliers de cassettes montées par un robot. IBM suivit tardivement avec sa bibliothèque 3495 à cartouches 3490E, d'une capacité totale allant jusqu'à 45 TO, et qui instaurait, à l'aide de DFSMS, une automatisation poussée du support de stockage cassette. D'autres réalisations intéressantes (telles que la machine "base de données" de Teradata) décentralisent complètement la gestion des données sur un un système spécialisé, en frontal de l'ordinateur.

2) LE SUPPORT BANDE

Le support bande est utilisé depuis longtemps en informatique: parlent en sa faveur son faible coût, une bonne capacité de stockage, un poids et un encombrement faible qui facilitent son transport et son rangement. Les inconvénients tiennent au temps d'accès, à la nécessité de manipulation manuelle (montage sur le dérouleur) et à sa limitation au stockage de fichiers séquentiels. Le support cassette (ou "cartouche"), apparu il y a quelques années (les 3480 ont été annoncés par IBM en 1984), a rendu son intérêt au support bande, avec une plus grande capacité (200 millions d'octets par cassette contre 160 pour les bandes), un plus faible encombrement et la possibilité de charger automatiquement plusieurs cassettes pour des fichiers de travail avec l'ACL (automatic cartridge loader).

Une bande ou une cassette est un support réutilisable, le critère de ré-emploi étant la date d'expiration du fichier qu'elle contient. Un système de gestion de bandothèque (ou cassettothèque) doit éviter l'écrasement de données non expirées et pouvoir fournir une liste des numéros de volume des bandes disponibles (bandes "scratch") utilisables à la demande de tout travail créant de nouveaux fichiers. IBM, hélas, ne fournit pas un tel système de gestion en MVS et oblige le client à se tourner vers la concurrence, à développer ses propres outils (examen des fichiers SMF, exits), ou à gérer un système "fermé" dans lequel une bande donnée est toujours utilisée dans le même travail pour le même fichier (allocation par nom de volume).

La création d'un fichier bande nouveau, dans un système "ouvert", se fait par allocation non spécifique (le numéro de volume n'est pas indiqué), qui se traduit à la console par un message de demande de montage de bande SCRTCH ("scratch", pour les fichiers de travail, la bande pourra être réutilisée en fin de travail) ou PRIVAT (pour les fichiers qu'on souhaite conserver). Avec JES3, dans tous les cas, le message n'indique que SCRTCH.

Par ailleurs, avec JES2, on peut monter une bande à l'avance sur un dérouleur et la faire connaître du système avant même qu'un travail ne la demande. Cette facilité est connue sous le nom d'AVR (automatic volume recognition); elle ne s'applique pas aux bandes scratch ou private (le nom du volume étant inconnu a priori).

Dans une exploitation moderne, le support bande est réservé aux gros fichiers peu utilisés, aux sauvegardes ou archivages de fichiers disque, aux sauvegardes périodiques de volume disque.

3) LE SUPPORT DISQUE

a) Description

Le disque est un support obligatoire en MVS (fichiers du système, fichiers de pagination, SPOOL de JES, fichiers VSAM, etc.). Pour une exploitation informatique il présente de nombreux avantages:

* performance: possibilité d'accès direct, et surtout rapidité d'accès, avec les dispositifs spéciaux tels que:

- les contrôleurs à cache, qui améliorent les temps de lecture;

- le fast write des 3990 qui améliore les temps d'écriture;

- la possibilité de connecter de nombreux canaux: jusqu'à 16 chpids peuvent servir un disque (mais un système 370 ne peut en gérer que 4);

- l'augmentation du nombre de chemins internes dans le string, avec le DLS (device level selection) ou le DLSE des 3990.

* capacité: plus de 600 millions d'octets pour une adresse de 3380, et il existe des disques de capacité double ou triple; on passe à 950 millions d'octets pour les 3390 (avec le double ou le triple pour les modèles 2 ou 3).

* fiabilité: moins de risques d'erreur de manipulation qu'avec les bandes. Par ailleurs, des pistes alternées sont prévues pour remplacer des pistes défaillantes (suite à un "data check");

* partage: un disque peut être partagé par plusieurs systèmes (par l'intermédiaire de plusieurs canaux ou plusieurs unités de contrôle). Nous verrons plus loin comment est réglé ce partage.

Les disques actuels (3380,3390), à cause de leur poids et de leur encombrement, sont inamovibles. Ils sont enfermés dans un HDA (head-disk assembly) qui les protège du milieu extérieur (poussières, fumées, etc.).

Physiquement, un disque est un empilement de surfaces magnétisées tournant à grande vitesse (environ 60 tours à la seconde) autour d'un axe central. Des têtes de lecture/écriture sont déplacées à leur surface par un mécanisme d'accès. Une piste du disque représente la quantité d'information lue pendant une rotation. Un cylindre est l'ensemble des pistes accessibles pour une position donnée du bras d'accès (toutes les têtes de lecture/écriture étant solidaires du bras d'accès occupent une même position dans un plan vertical).

Chaque enregistrement est stocké sous un format CKD (count-key-data). Il s'agit de 3 types de zones de stockage séparées par des "gaps". La zone count identifie l'emplacement de l'enregistrement par l'adresse physique CCHHR (CC=numéro de cylindre, HH=numéro de piste dans le cylindre, R= numéro de l'enregistrement dans la piste). La zone key (clé), optionnelle, est utilisée par certains types de fichiers (un PDS a des zones clés de 8 octets, pour repérer le dernier nom de membre stocké dans un bloc de répertoire). La zone data contient les données logiques du fichier, telles que les voit l'utilisateur; une zone data vide indique une fin de fichier (end of data). Le format FBA (fixed block architecture), utilisé pour les 3310 et 3370, consistant en pistes formatées en blocs de longueur fixe (512 octets), n'est pas employé en MVS.

b) La VTOC

La VTOC (volume table of contents) est une table maintenue par MVS, située sur le disque, qui dresse la liste des fichiers présents sur le disque, leurs caractéristiques physiques et leur emplacement (CCHHR). Elle est constituée de postes appelés DSCBs (dataset control blocks) décrivant les fichiers ou les portions d'espace libre. Ces DSCBs sont de 6 types (le type 0 correspond à un poste libre) :

* le type 1 décrit un fichier du disque. Un fichier peut être formé de 1 à 16 "morceaux" (extents) dispersés sur le disque. Le DSCB de type 1 décrit les 3 premiers extents;

* le type 2 complète la description d'un fichier ISAM (emplacement de l'index maître, des index de cylindre ou de piste);

* le type 3 peut compléter un DSCB de type 1, en décrivant jusqu'à 1 3 extents supplémentaires;

* le type 4, premier DSCB de la VTOC, unique, conserve des informations spécifiques du type de disque (capacité du disque, capacité d'une piste, pistes alternées), indique la taille de la VTOC, si elle est du type DOS ou MVS (non compatibles);

* le type 5 décrit les espaces libres disponibles sur le disque pour créer des fichiers ou des extents supplémentaires. Chaque DSCB de type 5 décrit jusqu'à 26 espaces utilisables;

* le type 6, marginal, décrit les extents partagés par plusieurs fichiers.

Toute création ou destruction de fichier disque se traduit sur la VTOC; un fichier est détruit simplement par effacement des DSCBs qui le décrivent et par mise à jour des DSCBs d'espace libre. Il n'y a pas effacement physique du contenu du fichier, sauf cas particuliers pour des raisons de sécurité (ERASE du DELETE VSAM ou "erase on scratch" de RACF).

Les postes décrivant les fichiers sont placés dans la VTOC par ordre d'arrivée; le système doit parcourir séquentiellement la VTOC pour accéder à l'entrée décrivant un fichier. Un volume disque pouvant contenir des milliers de fichiers, le temps d'accès à un fichier peut être impacté fortement par la recherche en VTOC. En MVS on peut créer une "VTOC indexée" pour accéder plus rapidement au DSCB voulu (les postes sont triés par nom de fichier). Cet index maintient aussi des informations sur l'espace libre dans le disque plus accessibles que les DSCBs de type 5 de la VTOC.

c) Le RESERVE/RELEASE; GRS

Quand un disque peut être partagé par plusieurs systèmes, l'intégrité des données ne peut plus être assurée: un fichier peut être modifié en même temps par plusieurs utilisateurs depuis des systèmes différents; on peut même imaginer le cas où l'un modifie le fichier tandis que l'autre le détruit ! Un dispositif de sérialisation est nécessaire, au moins pour les fichiers les plus importants du disque (comme la VTOC).

Le RESERVE (demandé en mettant le bit "reserve" du CCW) permet à la "tête de string" de limiter l'accès disque à un canal donné. Ainsi seul l'ordinateur émetteur du RESERVE et connecté au disque par ce canal privilégié pourra mettre à jour les données, tout accès par un autre système se heurtera à une condition "device busy". Avec XA la possibilité est étendue à tout le groupe de canaux de la machine qui sont connectés au disque (DPS, dynamic path selection). Quand les modifications sont terminées, un CCW de RELEASE est envoyé pour rendre à nouveau le disque disponible.

Le RESERVE est une fonction du matériel mise en oeuvre en standard par MVS et ses composants pour les fichiers les plus importants; elle n'est pas employée pour les fichiers de l'utilisateur (cela provoquerait trop de blocages de disques et conduirait à des étreintes fatales). Les autres systèmes 370 connaissent aussi le RESERVE: VSE avec son LOCKFILE pour ses propres ENQUEUEs, VM pour ses machines virtuelles (VM en fait cependant l'économie, en le simulant, dans le cas de minidisques partagés uniquement par des machines virtuelles, et non par d'autres systèmes natifs).

Le RESERVE est une fonction peu distinctive: elle bloque un disque entier là où il suffirait souvent de bloquer un seul fichier. De plus, rien n'est prévu pour éviter l'étreinte fatale entre deux systèmes cherchant chacun à accéder à un disque "réservé" par l'autre. Le logiciel GRS, apparu avec MVS-SP1.3, permet à plusieurs systèmes MVS, connectés par des liaisons canal à canal (CTC) dédiées, de sérialiser les accès à un niveau inter-systèmes. En particulier, les RESERVEs peuvent être transformés en simples ENQUEUEs (voir chapitre 2), reconnues à un niveau global (reserve conversion), ce qui réalise l'intégrité des données sans bloquer le disque. De même les ENQUEUEs de portée SYSTEM (connues d'un MVS) peuvent être transformées pour avoir une portée SYSTEMS (connues de tous les MVS du complexe GRS); l'ENQUEUE SYSDSN ainsi transformée augmente l'intégrité de tous les fichiers alloués par les travaux. Inversement les ENQUEUEs SYSTEMS peuvent devenir SYSTEM pour les ressources dont on est sûr de l'intégrité. Les systèmes MVS qui font partie

d'un complexe GRS sont liés par une espèce d'anneau à jeton (ring) qui informe tour à tour chaque MVS des nouveaux ENQUEUEs.

GRS est présent dès l'IPL de MVS, qu'il gère ou non un complexe de systèmes MVS (dans ce dernier cas, il se contente de gérer et de stocker les ENQUEUEs au niveau du système). GRS en tant que gestionnaire d'un complexe de systèmes MVS reliés par CTCs n'a pas la réputation d'être très performant; en effet toute demande de ressource doit faire le tour du ring avant d'être entérinée (sauf avec le "ring accelerator" en MVS-ESA). D'autres logiciels existent, qui autorisent la communication entre les différents MVS par disque partagé (sérialisé, lui, par le RESERVE) plutôt que par CTC. De tels logiciels (baptisés par certains "GRS du pauvre") peuvent suffire dès lors que le disque de communication ne subit pas trop d'accès de la part des autres composants du système. A noter que JES3 peut aussi sérialiser l'accès aux fichiers dans son propre complexe (voir chapitre 9).

d) La création de fichiers

Un fichier peut être créé sur disque par JCL, par allocation dynamique ou par la commande ALLOC de TSO (pour les fichiers VSAM, que nous laisserons ici de côté, l'emploi d'AMS est obligatoire). L'espace de "base" réservé au fichier est l'allocation primaire; on peut indiquer aussi un espace d'allocation secondaire, qui sera pris sur le disque (pourvu que la place existe) au cours de la vie du fichier si l'espace primaire ne suffit plus. L'espace secondaire pourra ensuite éventuellement être pris sur d'autres disques, pour le cas de fichiers multivolumes. La gestion de l'espace disque, réalisée par le composant DADSM (direct access device space management) de MVS, peut ainsi sembler très souple; du point de vue des performances, la dispersion d'un fichier en de multiples extents n'est pas optimale. L'espace primaire est alloué en un seul tenant sur le disque, autant que possible; sinon il pourra être alloué jusqu'en 5 extents maximum.

Le volume disque pour la création du fichier peut être indiqué explicitement (allocation spécifique), ou non. Dans ce dernier cas, le paramètre UNIT (nom ésotérique) indique toutes les unités sur lesquelles l'allocation peut être tentée. MVS établit cependant une distinction, selon qu'il s'agit de créer un fichier temporaire ou permanent. Les volumes ont un attribut d'utilisation, qui peut être:

* STORAGE (stockage): le volume est destiné aux allocations non spécifiques de fichiers permanents;

* PUBLIC: pour les allocations non spécifiques de fichiers temporaires (qui ne durent que le temps du travail);

* PRIVATE (privé): le volume ne peut être utilisé que pour des allocations spécifiques (le nom de volume doit être indiqué).

La distinction STORAGE/PUBLIC, qui peut sembler arbitraire, permet d'éviter une gruyérisation du disque qui proviendrait de la coexistence de fichiers permanents avec des fichiers temporaires (qui disparaissent en laissant un "trou" d'espace libre). L'attribut PRIVATE évite la pollution du disque par les fichiers temporaires. Les exemples suivants montrent comment fonctionnent les attributs d'utilisation (voir le chapitre 5 pour les paramètres du JCL):

```
//  DD  UNIT=3380,DISP=(NEW,CATLG),DSN=F1.PERM,VOL=SER=V1 (1)
//  DD  UNIT=3380,DISP=(NEW,CATLG),DSN=F2.PERM          (2)
//  DD  UNIT=3380,DISP=(NEW,KEEP),DSN=F3.PERM           (3)
//  DD  UNIT=(3380,2),DISP=(NEW,PASS),DSN=F4.PERM       (4)
//  DD  UNIT=3380,DISP=(NEW,PASS),DSN=&&TEMP            (5)
//  DD  UNIT=3380,DISP=(NEW,DELETE),DSN=F1.TEMP         (6)
//  DD  UNIT=3380,DISP=(NEW,CATLG),DSN=F5.PERM,
//  VOL=SER=(V1,V2,V3)                                  (7)
```

L'exemple (1) est un cas d'allocation spécifique: le fichier sera créé sur le volume V1 (quel que soit d'ailleurs son attribut). Dans les exemples (2), (3) et (4), le fichier sera créé sur un volume d'attribut STORAGE (pour l'exemple 4, on pourra avoir des allocations secondaires sur 2 volumes). Dans les exemples (5) et (6) les allocations se feront sur les volumes PUBLICs (ou sur les STORAGEs s'il n'y a pas de volumes PUBLICs déclarés). L'exemple (7) est le même que le (1), avec de plus les 3 volumes V1, V2 et V3 disponibles pour

l'allocation secondaire. Avec DFSMS, les attributs d'utilisation des disques perdent de leur importance (emploi de "pools" de disques).

Un avantage intéressant de l'allocation non spécifique est que sera choisi dans le pool de disques d'attribut STORAGE ou PUBLIC correspondant au nom ésotérique d'unité celui qui disposera d'assez d'espace pour l'allocation primaire, ce qui n'est pas possible pour l'allocation spécifique (exemple 7).

On peut chercher à contrôler l'allocation des fichiers: contrôle du nom de fichier, de l'espace demandé, et surtout du nom du disque demandé. L'exit IGGPRE00 associé à RACF peut être une première approche, en attendant DFSMS, qui est le dernier cri en la matière.

e) La gestion

Le support disque réclame une gestion moins triviale que celle du support bande. DFP est efficacement complété par plusieurs logiciels qui s'y consacrent.

DFDSS (data facility-dataset services) permet de copier des fichiers d'un disque à un autre (COPY), de prendre des sauvegardes (DUMP), de restaurer des fichiers (RESTORE), de récupérer les espaces libres répandus sur le disque (DEFRAG) ou présents dans les fichiers existants (RELEASE), et même de compresser des PDS (COMPRESS). DFDSS travaille au niveau fichier aussi bien qu'au niveau disque.

DSF (device support facilities) permet d'initialiser les disques (création du label, de la VTOC), de les examiner pour détecter des problèmes physiques à leur surface (et assigner éventuellement des pistes de remplacement), de créer ou supprimer la VTOC indexée.

DFDSS et DSF existent aussi en version autonome (stand-alone) pour fonctionner en l'absence de système d'exploitation.

ISMF (interactive storage management facility) consiste en un jeu de menus sous ISPF pour le suivi des fichiers et des volumes disques.

DFHSM (data facility hierarchical storage manager) fournit une gestion automatique des volumes disques qu'on lui déclare: sauvegardes périodiques complètes ou incrémentales (possibilité de ne sauvegarder que les fichiers modifiés depuis la dernière sauvegarde complète, grâce au bit "dataset changed" de la VTOC), migration sur support particulier des fichiers inactifs (une date "last referenced" est maintenue dans les DSCBs de la VTOC), maintien d'un espace libre suffisant sur le disque (avec la notion de seuil d'occupation). DFHSM s'appuie sur DFDSS ou AMS pour les opérations physiques. De fait, DFHSM crée bien une hiérarchie des supports magnétiques, avec les volumes primaires, mis à la disposition des utilisateurs et gérés par DFHSM, les volumes de premier niveau de migration qui contiennent les fichiers "migrés" ou sauvegardés par DFHSM (ces volumes peuvent être de performances moindres), les volumes de second niveau (bandes de préférence) pour des migrations supplémentaires.

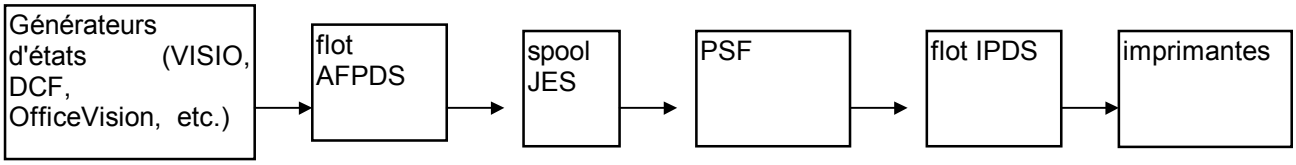
SMS (system managed storage) est un concept nouveau apparu avec DFP version 3 et MVS-ESA. Nous le décrivons plus en détail au chapitre 14.

H) Les fonctions avancées d'impression

Les fonctions avancées d'impression (AFP) constituent un nouveau mode d'impression basé sur les imprimantes en mode page (il s'agit chez IBM des familles d'imprimantes 3800, 3900, 38xx, 4028, 42xx, etc.) appelées aussi « adressables en tout point » (APA), l'élément de base de l'impression étant le pel (picture element) et non plus la ligne comme auparavant. Cette évolution est comparable au remplacement des écrans traditionnels de type 3270 par des écrans graphiques.

Les modèles du passé, dont l'exemple typique reste la 1403 d'IBM, étaient des imprimantes à impact, ligne à ligne, aptes à une production de masse avec une qualité d'impression passable. Le "flot 1403" mélangeait les données à imprimer avec des données de contrôle qui dirigeaient l'impression (codes saut ASA ou autres codes "machine"). Peu à peu, le mode ligne à ligne devenait impropre à une production de qualité, surtout quand une certaine structuration et des fonctions de mise en page étaient souhaitées (traitements de texte, PAO) avec la plus grande souplesse possible : différentes polices de caractères, du fenêtrage, la combinaison de texte et d'images, aussi bien sur le site central d'impression qu'en décentralisé. Avec AFP, une plus grande séparation s'impose entre l'applicatif et tout ce qui ressort de la mise en forme finale de l'état : l'aspect de l'état sur le papier peut être complètement modifié sans impact sur les applications (qui peuvent continuer à générer des états en mode ligne).

Le flot AFP (AFP data stream), issu des "documents composés", est envoyé sur le spool pour être traité par le gestionnaire d'impression PSF, qui communique avec JES par un interface FSS. L'impression génère un flot "IPDS" à destination des imprimantes. Le chemin suivi par les données est généralement le suivant :



PSF utilise les "ressources" AFP quand nécessaire pour mettre en forme les pages d'état. Il s'agit d'objets stockés dans des bibliothèques accessibles à PSF, créés par des logiciels adéquats (PMF, PPFA, OGL, FLSF, etc) et qui donnent toutes les caractéristiques physiques ou logiques de l'état :

- le FORMDEF décrit la page physique (bac à utiliser, recto/verso), le nombre de copies, le positionnement de la page "logique", le mode de présentation (portrait, paysage), les champs à supprimer, etc. ;

- le PAGEDEF décrit la page logique : les dimensions de la page, comment se placent les données sur la feuille, leur orientation, les polices de caractères à utiliser (fonts), la direction de l'impression, etc.

Il s'agit des deux principales ressources AFP, référencées dans le JCL par les mots clés FORMDEF et PAGEDEF. Au FORMDEF sont aussi liés les fonds de page (overlays), qui permettent d'ajouter diverses figures sur la page (boîtes, cercles, lignes, textes, etc.) et remplacent avantageusement les pré-imprimés. Les segments de page représentent des nuages de points issus le plus souvent d'images scannerisées (photographies, logos, signatures, etc.).

5. LE LANGAGE DE COMMANDE JCL

A) Généralités

Pour adresser ses requêtes au système d'exploitation, l'utilisateur a besoin d'un langage de commande pour décrire le travail demandé et les ressources nécessaires. MVS (comme tout système d'exploitation orienté vers le traitement par lots) reconnaît un langage de commande, le JCL (job control language), dans lequel sont codés tous les travaux. Grossièrement, il s'agit d'indiquer quels programmes doivent être exécutés et de quels fichiers ils ont besoin dans leurs traitements.

Le JCL de MVS est un langage nettement orienté "paramètres": moins "verbeux" que d'autres langages de commande (comme le JCL du DOS-VSE), il offre un petit nombre de commandes comportant de nombreux paramètres (qui peuvent eux-mêmes posséder de nombreux sous-paramètres). Les paramètres peuvent être de type positionnel: leur position relativement aux autres détermine leur signification; ou à mot-clé quand ils sont précédés d'un mot réservé qui indique leur nature et du signe "égal" (dans ce cas leur position n'a pas d'importance). Il ne sera pas question de détailler tous les paramètres, mais seulement de décrire le JCL dans les grandes lignes.

Un ordre de JCL est constitué de un ou plusieurs enregistrements de 80 octets (on parle encore de "cartes" en référence aux cartes perforées qui furent le premier support des ordres JCL). Chaque ordre a la structure suivante:

```
//nom commande paramètre1,paramètre2,... commentaires
```

La chaîne // permet de reconnaître qu'il s'agit d'un ordre JCL (//* pour une ligne commentaire) tandis que le nom sert à référencer la commande. Les paramètres sont séparés par des virgules.

Il y a trois commandes principales:

JOB: indique le début d'un travail (job). Un travail est une unité de traitement bien définie portant un nom (jobname) et gérée par le superviseur de travaux .

EXEC: indique le début d'une étape du travail. Un travail peut être divisé en plusieurs étapes (steps) pouvant dépendre les unes des autres et se déroulant les unes après les autres, selon l'ordre de succession des cartes EXEC qui suivent l'ordre JOB. Un travail peut comprendre jusqu'à 255 étapes. Le paramètre essentiel associé à l'ordre EXEC est le nom du programme à appeler pour démarrer l'étape.

DD (data definition): donne les caractéristiques d'un fichier utilisé dans l'étape. Il y a autant d'ordres DD suivant l'ordre EXEC que de fichiers à traiter dans cette étape. Chaque ordre DD porte obligatoirement un nom, le DDname, utilisé par le programme pour accéder aux fichiers associés. Dans les dernières versions de MVS on peut coder plus de 3000 cartes DD par étape.

Les fichiers SYSOUT (System Output) représentent un type de fichier particulier. Il s'agit de fichiers en écriture créés dans les différentes étapes et gérés par le superviseur de travaux qui les place sur disque (dans le SPOOL). Ces fichiers peuvent être ensuite consultés, imprimés ou détruits.

B) Les cartes JOB, EXEC, DD

Nous allons présenter seulement quelques paramètres importants pour chacun de ces ordres.

*** ordre JOB:**

Les deux premiers paramètres sont positionnels: une information comptable, le nom du programmeur. Ces paramètres ne sont pas obligatoires (sauf décision explicite du site).

CLASS= : indique la classe d'exécution pour le gestionnaire de travaux. Une installation peut associer à une classe de nombreux paramètres par défaut (temps d'exécution, etc.) qui s'appliquent à tout travail s'exécutant dans cette classe.

TIME= : limite en minutes et secondes du temps processeur consommé par le travail. Au-delà de cette limite, il y a fin anormale du travail. La valeur particulière TIME=1440 (qui correspondrait à 24 h) indique qu'il n'y a pas de limite.

REGION= : limite la taille de mémoire que le travail peut requérir dans la zone privée. Il ne s'agit pas d'un espace mémoire accordé systématiquement et d'emblée au travail (il n'y a pas de partition en MVS !). Ce paramètre est une simple limite pour l'acquisition de mémoire par les GETMAlNs, et particulièrement les GETMAlNs "variables" qui permettent de demander un espace mémoire de taille non précisée (comprise entre un minimum et un maximum). REGION=0K ne fixe aucune limite précise; de toute façon la LSQA restreint "par le haut" la zone privée, et REGION=0K peut entraîner le problème de "collision" évoqué au chapitre 3D dans le cas de GETMAINs trop "gourmands".

TYPRUN=: permet de vérifier la syntaxe du JCL sans exécution (TYPRUN=SCAN) ou de mettre le travail en attente (TYPRUN=HOLD).

*** ordre EXEC:**

Certains paramètres sont les mêmes que pour la carte JOB: REGION, TIME, etc. Ils imposent le même genre de limitation, mais uniquement au niveau de l'étape.

PGM= indique le nom du programme à exécuter. Par la combinaison PGM=*.stepname.ddname on peut appeler un programme créé de façon temporaire dans une étape précédente.

PARM= permet de passer au programme un paramètre (chaîne de 100 caractères maximum). Le système fait précéder en interne cette chaîne d'un demi-mot indiquant sa longueur et fait pointer le registre 1 vers l'adresse de la chaîne.

COND= permet de conditionner l'exécution de l'étape en fonction des codes retour des étapes précédentes (voir plus loin).

*** ordre DD:**

Cet ordre décrit un fichier de l'étape en vue de son allocation par l'initiateur. L'ordre DD est très riche en paramètres, aussi nous n'indiquerons que les principaux.

DISP= indique l'état du fichier et de quelle façon le système peut en disposer. Il y a trois sous-paramètres positionnels:

- l'état du fichier: indique s'il faut le créer (NEW), s'il existe déjà et s'il faut l'allouer de façon exclusive (OLD) ou en permettant le partage (SHR). Un sous-paramètre commode est MOD, qui permet de rajouter des enregistrements à la fin du fichier, s'il existe, ou qui équivaut à NEW dans le cas contraire. Les paramètres NEW, OLD et MOD génèrent une ENQUEUE exclusive sur le nom du fichier;

- la façon dont le système doit disposer du fichier en cas de fin normale de l'étape (pas d'ABEND): DELETE, KEEP, CATLG, UNCATLG et PASS (qui remet la décision à plus tard en conservant le fichier pour les étapes suivantes);

- de même la disposition à prendre en cas d'ABEND: DELETE, KEEP, CATLG, UNCATLG .

DCB= (data control block) comporte de nombreux sous-paramètres, qui permettent de préciser les caractéristiques physiques du fichier: longueur d'enregistrement LRECL, taille de bloc BLKSIZE, organisation

DSORG, format RECFM, nombre de tampons à prévoir BUFNO, etc. On peut se contenter de recopier les caractéristiques d'un autre fichier en se référant à lui dans le paramètre DCB. A noter que les caractéristiques du fichier déclarées par le paramètre DCB prévalent sur celles qui sont indiquées dans les étiquettes; cependant celles que comportent les macros DCB des programmes prévalent sur le DCB du JCL.

UNIT= indique l'unité périphérique à utiliser pour le fichier, par son adresse, ou un groupe d'unités, par le nom "ésotérique" qui le désigne.

SPACE= indique l'espace souhaité sur le disque, en pistes, cylindres ou blocs (n'a pas de sens pour une bande). Certains sous-paramètres précisent que l'allocation doit être contiguë (CONTIG), qu'il faut prendre le maximum d'espace sur le disque (MXIG), qu'il faut restituer l'espace inutilisé après écriture et fermeture du fichier (RLSE), que le fichier doit être alloué à une adresse cylindre/piste ab$olue sur le disque (ABSTR).

DUMMY indique qu'il s'agit d'un fichier fictif: il n'y a alors ni allocation ni entrée-sortie sur le fichier. On peut ainsi tester des programmes en indiquant DUMMY pour les fichiers qu'on ne veut pas créer physiquement.

DSNAME= indique le nom du fichier (dataset name). Si ce paramètre n'est pas précisé ou s'il commence par &&, le système génère un nom ésotérique (comportant la date et l'heure pour éviter des doublons). Le DSNAME ne peut dépasser 44 caractères de long. Pour les fichiers sur bande, seuls les 17 de droite sont écrits sur l'étiquette. DSNAME=NULLFILE équivaut à DUMMY.

VOLUME= indique notamment le volume sur lequel réside le fichier (SER=) et s'il faut le garder monté sur l'unité (RETAIN) en prévision des étapes suivantes (fichier bande).

* : ce paramètre indique que les cartes qui suivent, jusqu'à la première carte de JCL ou jusqu'à la carte /*, constituent un fichier "in-stream" (dans le flot du JCL) qui sera stocké dans le SPOOL. Ce fichier a ordinairement un LRECL de 80 caractères et ne peut être accédé par les programmes qu'en lecture. Par le DD * on évite de devoir créer, allouer et initialiser des fichiers qui ne contiendraient guère plus que quelques cartes. Les JCLs pour invoquer les utilitaires MVS comportent souvent une carte //SYSIN DD * pour entrer les paramètres d'appel de l'utilitaire. Si le fichier "in-stream" contient des cartes de JCL, il faut employer DATA au lieu de * et indiquer éventuellement un délimiteur de fin par le paramètre DLM=.

Une carte DD peut être constituée de plusieurs cartes regroupées sous le même DDNAME. On a alors une concaténation de fichiers: ces fichiers apparaîtront au programme comme formant un unique fichier logique. Par exemple:

```
//DD1  DD DISP=SHR,DSN=FICHIER1,DCB=BLKSIZE=23400
//          DD DISP=SHR,DSN=FICHIER2
//          DD DISP=SHR,DSN=FICHIER3
```

Le fichier logique de DDNAME DD1 pourra être lu comme s'il s'agissait d'un seul fichier; en fait les 3 fichiers seront lus physiquement les uns après les autres.

Dans les versions de MVS les moins récentes (avant DFP 2.3), il fallait que le fichier en tête de la concaténation ait le LRECL et le BLKSIZE le plus grand. On y remédiait artificiellement en rajoutant une carte DCB= pour imposer un LRECL et un BLKSIZE à la méthode d'accès (voir l'exemple). De même avant DFP version 3 les fichiers de la concaténation devaient résider sur des supports physiques de même genre (sauf si précisé autrement dans la macro DCB).

Quelques cartes particulières:

//STEPLIB DD: indique la ou les bibliothèques de modules à examiner pour charger le programme indiqué dans l'ordre EXEC de l'étape ainsi que tous les sous-programmes .

//JOBLIB DD: même chose au niveau du travail tout entier. Cependant, au niveau d'une étape, la STEPLIB, si elle est indiquée, remplace la JOBLIB pour la recherche des modules.

//STEPCAT DD: indique le ou les catalogues VSAM ou ICF à examiner pour localiser les fichiers catalogués de l'étape.

//JOBCAT DD: même chose au niveau du travail tout entier. Le STEPCAT, s'il est précisé, remplace le JOBCAT pour l'étape en question.

C) Les procédures

Pour éviter d'avoir à réécrire des séquences d'ordres JCL identiques ou très semblables, on peut créer des procédures. Il suffira de les appeler par un EXEC pour inclure automatiquement leur JCL dans les travaux. On peut s'épargner ainsi la peine de coder de nombreux steps remplissant une fonction identique. L'association aux procédures de paramètres symboliques d'appel permet de ne pas les figer et souvent de les rendre très générales. Une procédure peut être "in-stream" si elle figure dans le flot du JCL, dont elle est séparée par une carte // PEND (procedure end); elle n'a alors validité que pour le travail dans lequel elle figure. Elle peut être aussi "cataloguée" si elle figure dans les bibliothèques proclibs du système; c'est le cas très souvent pour les procédures de compilation, d'édition des liens, etc. L'appel d'une procédure s'effectue par une carte // EXEC PROC=
<nom de la procédure> ou plus simplement EXEC <nom de la procédure>.

Voici un exemple de procédure copiant un fichier séquentiel:

```
//COPIE PROC IN=,OUT=, paramètres sans valeur par défaut
//      PROG=IEBGENER     nom de l'utilitaire de copie
//STEP1          EXEC PGM=&PROG,REGION=900K
//SYSUT1         DD DISP=SHR,DSN=&IN    fichier en entrée à copier
//SYSUT2         DD DISP=SHR,DSN=&OUT   fichier en sortie
//SYSPRINT       DD SYSOUT=*            compte rendu d'exécution
//SYSIN          DD DUMMY               paramétrage de l'utilitaire
```

Les 3 paramètres sont &IN, &OUT et &PROG. L'appel de cette procédure COPIE s'effectuera par:

```
// EXEC COPIE,IN='FICHIER1',OUT='FICHIER2'
```

On peut modifier la valeur par défaut du paramètre &PROG; par exemple, avec des fichiers partitionnés, on aurait cet appel:

```
// EXEC COPIE,IN='PDS1',OUT='PDS2',PROG=IEBCOPY
```

(IEBCOPY étant l'utilitaire utilisé pour copier des PDS).

A noter que le pupitreur pourrait utiliser la procédure par une commande: START COPIE,IN='FICHIER1',OUT='FICHIER2'

En MVS les procédures connaissent quelques limitations: on ne peut avoir de carte DD * dans une procédure; une procédure ne peut en appeler une autre (cela compliquerait la conversion et il y aurait un risque de boucle avec des procédures s'appelant mutuellement).

On peut modifier une procédure lors de son appel autrement que par les paramètres symboliques. On peut effectuer un remplacement ("override") de nombreux paramètres du JCL ainsi que des cartes DD. Les cartes DD de remplacement doivent suivre la carte EXEC d'appel de la procédure et respecter l'ordre initial. Ainsi, dans notre exemple, si on veut modifier la REGION, et les cartes SYSPRINT et SYSIN:

```
//    EXEC COPIE,IN='PDS1',OUT='PDS2',PROG=IEBCOPY,
//     REGION.STEP1=2000K         override de la REGION
//STEP1.SYSPRINT DD DUMMY         override de la carte SYSPRINT
//STEP1.SYSIN    DD *             override de la carte SYSIN
```

```
 COPY I=SYSUT1,O=SYSUT2
/*
```

D) Le conditionnement des étapes

Les paramètres de conditionnement permettent de subordonner le démarrage de certaines étapes à la bonne (ou mauvaise) exécution d'autres. On peut ainsi avoir des travaux dont les différentes étapes s'exécutent de façon cohérente et pour lesquels les problèmes qui peuvent survenir à l'exécution seront pris en compte à l'avance.

Chaque programme invoqué à chaque étape dans l'EXEC PGM renvoie au système un code retour dans le registre 15 quand il rend le contrôle définitivement. Ordinairement (pour ce qui est des utilitaires IBM) les valeurs de 0 à 4 n'indiquent aucune erreur grave. Grâce au paramètre COND, on peut établir l'enchaînement des étapes selon les codes retour. Voici un exemple:

```
//STEP1 EXEC PGM=PGM1
//STEP2 EXEC PGM=PGM2,COND=(12,LT,STEP1)
//STEP3 EXEC PGM=PGM3,COND=((8,EQ,STEP1),(4,GE,STEP2))
//STEP4 EXEC PGM=PGM4,COND=(4,LT)
```

Le paramètre COND exprime en fait une condition de non-exécution de l'étape. Ainsi le STEP2 ne sera pas exécuté si le code retour du STEP1 est supérieur à 12. Le STEP3 ne sera pas exécuté si le code retour du STEP1 est 8 ou si celui du STEP2 est inférieur ou égal à 4. Enfin le STEP4 ne sera pas exécuté dès qu'une des étapes précédentes aura un code retour plus grand que 4.

Avec les paramètres ONLY et EVEN, on peut forcer une étape à être exécutée, respectivement, seulement ou même si une étape précédente s'est terminée en ABEND, alors que le traitement normal est de mettre fin à un travail dès qu'une étape fait ABEND. Ainsi, en poursuivant notre exemple:

```
//STEP5 EXEC PGM=PGM5,COND=((16,GE,STEP3),ONLY)
```

Le STEP5 ne sera pas effectué si une des conditions suivantes est vraie:

* aucune des étapes précédentes n'a fait ABEND;

* le STEP3 a renvoyé un code retour inférieur ou égal à 16. S'il advient que le STEP3 ne soit pas effectué (il est lui-même conditionné), la condition est simplement ignorée.

Le paramètre COND exprime la condition pour ne pas effectuer une étape; parfois on souhaiterait une condition pour effectuer une étape. On peut utiliser un artifice en rajoutant une étape factice. Par exemple pour exécuter un STEP6 uniquement si une des étapes précédentes a renvoyé un code supérieur à 4, on peut ajouter les étapes suivantes:

```
//FACTICE EXEC PGM=IEFBR14,COND=(4,LT)
//STEP6   EXEC PGM=PGM6,COND=(0,EQ,FACTICE)
```

Si l'une des étapes qui précèdent donne un code supérieur à 4, FACTICE n'est pas exécuté et STEP6 le sera, puisque son conditionnement sera ignoré. Dans le cas contraire, FACTICE sera exécuté, renverra un code 0 (IEFBR14 est un "utilitaire" qui fait uniquement retour à l'appelant en mettant le registre 15 à 0), et STEP6 ne sera pas exécuté.

E) Les apports d'ESA et de SMS

A partir de MVS/SP3, le JCL se trouve amélioré dans le sens d'une indépendance plus caractérisée par rapport à la géométrie physique des disques (SDB, "constructs" de SMS), de la simplification de certains paramètres, avec l'apparition de quelques autres liés notamment à SMS :

* l'espace d'allocation des fichiers peut être indiqué en nombre d'enregistrements, en kilo-octets ou en méga-octets plutôt qu'en pistes ou cylindres. Dans l'exemple qui suit, on alloue 500 000 enregistrements de 80 octets (nouveau paramètre AVGREC), le blocksize sera calculé par le système (SDB), les paramètres LRECL, RECFM et DSORG sont "promus" (pas de paramètre DCB) :

```
//ALLOC    EXEC  PGM=IEFBR14
//DDNAME   DD    DSN=FICHIER1,DISP=(NEW,CATLG),UNIT=SYSDA,
//         AVGREC=U,
//         SPACE=(80,(500000,1000)),
//         LRECL=80,RECFM=FB,DSORG=PS
```

Pour allouer un espace de 5 méga-octets, on indiquerait :

```
//         AVGREC=M,SPACE=(1,(5,5))
```

* les fichiers VSAM peuvent être alloués ou détruits par JCL (on peut aussi allouer des fichiers VSAM temporaires). Par exemple :

```
//ALLOC    EXEC  PGM=IEFBR14
//DDNAME   DD    DSN=FICHIER.VSAM,DISP=(NEW,CATLG),UNIT=SYSDA,
//         AVGREC=U,
//         SPACE=(80,(500000,1000)),
//         RECORG=KS,LRECL=80,KEYOFF=0,KEYLEN=5
//DDOLD    DD    DSN=FICHIER.VSAM.OLD,DISP=(OLD,DELETE)
```

* les ordres IF, THEN, ELSE, ENDIF, INCLUDE, JCLLIB, SET (à partir de MVS/SP4) amènent une souplesse attendue depuis longtemps, inspirée de ce qui existe dans les langages de programmation de haut niveau. IF, ELSE et ENDIF permettent de tester des valeurs de code retour ou code abend pour exécuter certaines portions de JCL. INCLUDE et JCLLIB permettent d'insérer des séquences de JCL dans un job. SET permet d'initialiser des paramètres symboliques qui, par substitution, vont donner un JCL adapté à des besoins particuliers. Par exemple :

```
//SETDSN   SET     DS1='PROD.CLIENTS'
//REFLIBS  JCLLIB  ORDER=(PROD.JCLPROC1,TEST.JCL)
//UPDATE   INCLUDE MEMBER=MAJ01020
//         IF   (ABENDCC=SE37 OR RC GE 8) THEN
//COMPRESS EXEC PGM=IEBCOPY
//SYSPRINT DD   SYSOUT=*
//SYSIN    DD   DUMMY
//SYSUT1   DD   DISP=SHR,DSN=&DS1
//SYSUT2   DD   DISP=SHR,DSN=&DS1
//REUPDATE INCLUDE MEMBER=MAJ01020
//         ELSE
//SAUVE    INCLUDE MEMBER=BACK1020
//         ENDIF
```

6. LA GESTION DES PROGRAMMES

A) La création et l'écriture des programmes

Le programme est une unité élémentaire de travail de l'informaticien (qui est peu visible sinon arbitraire pour l'utilisateur final). Il est écrit après détermination et analyse de la fonction exacte qu'on lui assigne dans le cadre d'un projet. En principe un programme accomplit un certain nombre de tâches prédéfinies; en fait il n'y a pas de règle délimitant dans un cadre fixe un programme par rapport aux objectifs visés. Pour une tâche donnée on peut écrire un seul programme ou scinder cette tâche en unités plus fines et écrire plusieurs programmes ou sous-programmes, selon l'ampleur du travail à réaliser et les moyens humains dont on dispose (la mort du programmeur, maintes fois annoncée, se fait encore attendre). Vis-à-vis d'un système d'exploitation en revanche un programme est une unité bien définie qu'il faut stocker, charger en mémoire, exécuter et gérer par des blocs de contrôle spéciaux.

Le programme est écrit dans un langage symbolique plus ou moins proche du langage humain (et de la logique humaine). Tel quel, qu'il soit écrit en assembleur, Cobol, PL/1, Fortran ou autre, il est sous la forme d'un "programme source" qui n'est pas utilisable par l'ordinateur, mais intéresse le programmeur seul. Une première étape consiste à le passer par un utilitaire spécifique au langage symbolique pour produire, après une compilation ou un assemblage, un module "objet" en langage machine. Ce module objet n'est pas exploitable par la machine, car il ne peut s'exécuter tel quel. Une dernière manipulation, la résolution des liens (link-edit), permet de regrouper différents modules objets (écrits peut-être par différents programmeurs dans différents langages) en "load-modules" (modules chargeables) stockés sur des fichiers sur disque, les "loadlibs" ; ces modules seuls peuvent être chargés en mémoire par le système pour leur passer le contrôle. La résolution des liens répertorie aussi toutes les "constantes d'adresse" (façon de référencer directement un endroit dans un module sans utiliser l'adressage indirect registre de base + déplacement) de manière à pouvoir les résoudre au chargement du module juste avant l'exécution. Sont aussi répertoriées des caractéristiques telles que le point d'entrée du module (première instruction à être exécutée), son mode d'adressage (AMODE) et de résidence (RMODE) et d'autres propriétés d'exécution. En MVS un module est désigné par un nom alphanumérique d'au plus 8 caractères; le même module peut recevoir plusieurs noms (alias).

L'écriture de programmes implique certaines conventions de base qu'on doit respecter si on veut un déroulement correct des opérations. En OS/MVS (et dans d'autres systèmes) l'appel de programmes et de sous-programmes obéit à des règles précises. On doit sauvegarder les registres du programme appelant pour qu'il continue son travail de manière transparente quand on lui rend le contrôle: on utilise pour cela une "save area" de 18 mots qu'il nous fournit, pointée par le registre 13; quand on rend le contrôle au programme appelant ou au système, on restaure le contenu des registres à partir de cette zone (mis à part le registre 15 qui est très souvent utilisé pour passer un code retour). Voici la description de la save area d'un programme (P):

* 1er mot: inutilisé (sauf par le langage PL/1);

* 2e mot: adresse de la save area du programme appelant (qui a appelé (P)). Cette adresse est stockée par le programme en cours (P): c'est le contenu du registre 13 à l'entrée dans (P);

* 3e mot: adresse de la save area du dernier programme appelé par (P) (ou zéro si aucun sous-programme n'a été appelé). Cette adresse est stockée par tout sous-programme appelé par (P);

* les autres mots servent à sauvegarder le contenu des registres généraux 14 (adresse de retour à (P)), 15 (point d'entrée du sous-programme appelé par (P)), 0 à 12 avant tout appel d'un sous-programme de (P). Cette sauvegarde est effectuée par le programme appelé par (P), mais dans une zone contrôlée par (P). On aboutit à un chaînage de save areas entre programmes qui permet à chacun de restaurer le contenu des registres avant de rendre le contrôle à son appelant.

A l'appel d'un programme le registre 15 donne l'adresse du point d'entrée de ce programme et le registre 14 l'adresse de retour au programme appelant (QU au système). Le registre 1 est utilisé pour pointer sur une liste d'adresses de paramètres. Un registre de base doit être initialisé en début de programme (soit en chargeant le

contenu du registre 15, soit par une instruction BALR); les adresses des zones de travail ou des instructions du programme seront exprimées en déplacements par rapport à l'adresse contenue par le registre de base (vu le jeu d'instructions 370, un registre de base "couvre" des déplacements de 0 à 4095, soit une page seulement; ceci oblige souvent à avoir plusieurs registres de base). On est ainsi indépendant de l'adresse de chargement du programme. Toutes ces conventions sont suivies autant dans les programmes assembleur (où cela est traduit dans le codage lui-même) qu'en langage symbolique où cela est transparent pour le programmeur (qui a, il faut bien l'avouer, d'autres préoccupations...).

Un programme peut être appelé de multiples façons. L'EXEC PGM du JCL invoque le programme principal de l'étape: il est chargé par l'initiateur du travail. Un programme peut appeler un sous-programme par simple branchement si le programme est déjà en mémoire ou s'il a été "link-edité" avec ce sous-programme (macro CALL, appel "statique"). L'appel peut aussi être dynamique (macro LINK, ou ATTACH si de plus on l'exécute en sous-tâche) et provoquer le chargement en mémoire du programme avant de lui passer le contrôle. L'appel dynamique par une autre macro, XCTL, a ceci de particulier que le programme appelé ne fait pas retour à l'appelant, mais à un programme de niveau supérieur (ce qui oblige à restaurer correctement les registres). L'appel par les macros LINK, ATTACH et XCTL provoque la création d'un bloc PRB (program request block) lié au TCB et qui sert à contrôler l'exécution du load- module.

L'appel dynamique est reconnu comme moins rapide: il nécessite la recherche dans les répertoires parfois très grands des loadlibs (puis éventuellement dans ceux des bibliothèques du système). En revanche il est plus souple et assure d'obtenir la version la plus récente du sous-programme (ou une version modifiée ad hoc). Pour cette raison un sous-programme très utilisé devrait toujours être appelé dynamiquement, sauf raisons de performance; un appel statique obligerait à refaire l'édition des liens avec tous les programmes appelants.

Sur la rédaction elle-même des programmes dans un environnement MVS, il y a peu de choses qu'on ne puisse dire dans tout autre système. Un programme devrait être lisible et suffisamment commenté pour que sa maintenance (assurée peut-être par une autre personne que l'auteur) puisse être commode. Les règles de programmation structurée proscrivent l'emploi du GO TO et autres branchements qui rendent parfois impossible à suivre le cheminement logique. Le programmeur doit prévoir tous les cas qui peuvent se présenter à l'exécution (et qui peuvent passer au travers des tests préliminaires à la mise définitive en exploitation): vérifier que telle donnée supposée reçue en format décimal packé l'est bien; que telle table interne qu'il gère dans son programme ne déborde pas (ce qui "écrase" d'autres données ou du code et provoque des résultats imprévisibles); qu'il reçoit bien du programme appelant le nombre prévu de paramètres (ce qui est difficile en COBOL mais simple en assembleur: la dernière adresse de paramètre est "signée" par un 1 dans le premier bit); qu'il ferme tous les fichiers qu'il a ouverts avant de rendre le contrôle; que sa programmation n'entraîne pas une "boucle" dont on ne puisse jamais sortir. Il s'agit là des erreurs les plus fréquentes. MVS et la mémoire virtuelle libèrent suffisamment le programmeur des anciennes contraintes liées à l'environnement matériel et à ses limitations (où va se charger mon programme ?; aura-t-il assez de place pour s'exécuter ?; de quels périphériques aura-t-il besoin ?) pour qu'il puisse se consacrer pleinement au développement applicatif (avec une programmation "propre") ainsi qu'aux tests.

B) Le chargement et l'appel

Les programmes que l'on exécute en mémoire sont les "load modules" issus du traitement des modules objets (ou même d'autres load modules) par l'édition des liens. Le load module se compose:

* du code destiné à être exécuté: le "texte". Ce texte exécutable est formé d'un ensemble de "sections de contrôle" (CSECTs); une section de contrôle est une unité de programmation, c'est l'entité de base de l'éditeur de liens;

* de données nécessaires à son chargement en mémoire: les dictionnaires de translation (RLD, relocation dictionary) qui répertorient les constantes d'adresse ("adcons"). Ces constantes, qui référencent de manière "absolue" un endroit du module, une autre section de contrôle (référence externe), un point d'entrée, doivent être "résolues" lors du chargement en mémoire du programme; elles sont initialement sous une forme relative, en sorte qu'il suffit de leur ajouter l'adresse de chargement du programme (inconnue avant l'appel) pour avoir l'adresse virtuelle vraie;

* d'informations utiles pour d'autres éditions de liens où il se trouverait impliqué: le dictionnaire des symboles externes (CESD, composite external symbol dictionary) qui comporte les noms des sections de contrôle et de leurs points d 'entrée .

Un load module est donc constitué, successivement, du CESD, du texte, du RLD et d'un indicateur de fin de load module. Après chargement, une fois mises à jour les constantes d'adresse, ne reste en mémoire que le texte à exécuter. Les modules objets dont est issu un load module ont une structure comparable: ESD, texte, RLD et indicateur de fin.

Le lieu de résidence habituel des programmes est la zone privée où ils sont chargés depuis la STEPLIB ou la JOBLIB; le programme résidera au-dessus ou en dessous de la "barre" des 16 Méga-octets selon son RMODE (residency mode). Une table de tous les programmes utilisés est maintenue pour chaque étape d'un travail, chaque poste ("CDE", contents directory entry) décrivant un module de la "job pack area queue"; de même, au niveau d'une tâche, pour les programmes chargés par la macro LOAD, est gérée la chaîne des "LLEs" (load list elements). Les programmes chargés par LOAD peuvent être détruits de la mémoire par la macro DELETE. Un programme appelé dynamiquement est détruit aussi quand il n'est plus utilisé; un compteur du nombre d'utilisateurs est maintenu pour cela, le "responsibility count", qui tombe à zéro dans ce cas.

Des programmes susceptibles d'être appelés par de nombreux espaces-adresses peuvent être mis en commun. Plusieurs possibilités sont offertes par MVS:

* le chargement en MLPA, FLPA ou PLPA. Les modules sont chargés une fois pour toutes au démarrage du système à partir de la bibliothèque SYS1.LPALIB et éventuellement de sa concaténation. Il s'agit du mode le plus performant: ils sont immédiatement disponibles puisque toujours présents en mémoire virtuelle (et même fixés, si on veut éviter les fautes de page) et peuvent être appelés par un simple branchement sans qu'on ait besoin de les charger en zone privée. On ne peut cependant les modifier ni prendre en compte une nouvelle version; de plus ils résident dans des zones protégées en écriture et doivent pouvoir être appelés par plusieurs tâches simultanément, ce qui leur impose un certain mode d'écriture très strict (voir plus loin la ré-entrance);

* la CSA est de plus en plus utilisée par divers logiciels pour charger des modules communs. L'emploi est le même que pour le cas précédent, avec un certain dynamisme en plus;

* la link-list. Il s'agit d'une liste de bibliothèques de modules déterminée au démarrage constituée de la SYS1 .LINKLIB et de sa concaténation (15 bibliothèques de plus en MVS-SP, plus d'une centaine en XA selon le nombre d'extents qu'elles possèdent). Ces bibliothèques sont systématiquement examinées quand on appelle un module qui n'est pas fourni en STEPLIB/JOBLIB. Quand le module est trouvé, il est chargé dans l'espace-adresse demandeur.

La recherche en link-list entraînant des lectures très longues de répertoires, ce qui posait des problèmes de performance en SP, on a eu l'idée en MVS-XA de réduire les délais de recherche en maintenant une table des entrées des répertoires classée par module et gérée par un nouvel espace-adresse, la LLA (link-list lookaside), capable de maintenir dans des tables à recherche rapide plus de 128000 modules. On obtient très rapidement l'adresse disque où réside le module à charger en appelant (par cross-memory) une routine de recherche en table. De plus on peut faire prendre en compte une nouvelle version d'un module (par une commande de refresh), ce qu'on ne peut faire avec le BLDL des versions SP. Le VLF de MVS-ESA étend encore ces possibilités en maintenant des copies (et non plus seulement des pointeurs disques) des objets les plus utilisés (programmes, tables, Clists de TSO);

* le "virtual fetch". Il s'agit d'une option un peu intermédiaire entre la link-list et la LPA. Un espace-adresse particulier pré-charge des modules déterminés sur les fichiers de pagination d'une manière semblable à des fichiers VIO et maintient un répertoire des modules. On accède à ce répertoire par cross-memory et on peut charger le module plus rapidement que par la méthode habituelle. Le virtual fetch nécessite cependant pour être employé une programmation supplémentaire par rapport aux appels "standard", ce qui fait qu'il n'est guère utilisé hormis par le SGBD IMS. Le VLF d'ESA devrait à terme le remplacer avantageusement.

L'ordre que suit le système pour rechercher un module appelé par une tâche essaie de favoriser quand il le peut la recherche en mémoire plutôt que sur disque, et dans les zones privées de l'espace-adresse plutôt que dans les zones communes. Voici sa séquence de recherche:

* il cherche d'abord, dans la Job Pack Area de l'espace-adresse, correspondant à tous les modules déjà invoqués dans cette étape du travail (jobstep), s'il y a une copie du module qui soit utilisable; en effet tout programme présent en mémoire déjà appelé ne peut être pris en compte dans tous les cas: s'il n'a pas été signalé réutilisable lors de l'édition des liens, ses données ou même ses instructions ont pu être modifiées lors d'un appel précédent (voir le paragraphe suivant) et le système considère que cette "corruption" possible ne le rend pas propre à être ré-exécuté;

* il cherche ensuite le module dans les bibliothèques déclarées en JOBLIB, STEPLIB ou autre TASKLIB, s'il y en a. Une TASKLIB est une bibliothèque de load-modules désignée lors de l'ATTACH d'une sous-tâche (paramètre DCB);

* s'il n'a pas trouvé le module dans les ressources propres au travail, il se tourne vers les ressources globales: I'"active LPA queue" constituée de certains modules de la PLPA mis dans cette file par divers composants; la FLPA puis la MLPA si elles existent; puis les modules de la PLPA représentés en mémoire par les blocs LPDE (link pack directory entry) issus du chargement de la SYS1.LPALIB au démarrage;

* enfin les bibliothèques de la link-list (appelée aussi SYS1.LINKLIB logique) en passant d'abord par les tables de la LLA si elles existent.

A ce point de la recherche, si le module n'est toujours pas trouvé, le travail est terminé anormalement (abend 806) ou, dans le meilleur des cas, le contrôle est passé à une routine d'erreur (paramètre ERRET des macros d'appel dynamique). On voit que le processus de recherche d'un programme appelé dynamiquement peut être relativement long; étant donné la performance des recherches en zones communes ou en link-list (avec la LLA), le goulet d'étranglement est le plus souvent la STEPLIB (ou JOBLIB). Quand c'est possible et pour des travaux utilisant presque uniquement des modules systèmes (programmes utilitaires, utilisateurs TSO), on a intérêt à éviter l'emploi des STEPLIBs/JOBLIBs.

La "terrible" barre des 16 Méga-octets

Voici une barre (virtuelle pourtant !) qui a plongé dans la perplexité nombre de programmeurs ! Avec MVS/XA et MVS/ESA, on dispose d'adresses sur 31 bits plutôt que 24, ce qui ouvre l'accès à une mémoire virtuelle beaucoup plus grande qu'auparavant. La gestion des programmes s'en trouve compliquée avec l'apparition d'attributs tels que le RMODE (le programme est-il situé en dessous ou au-dessus de la barre ?) et le AMODE (le programme est-il capable d'accéder aux données placées au-dessus de la barre ?). Si tous les programmes pouvaient profiter de l'adressage à 31 bits, ce serait tout bénéfice pour le programmeur, mais, comme toujours avec MVS, il faut tenir compte du passé et de la lenteur (parfois la réticence) d'IBM lui-même à faire évoluer le logiciel de concert avec des évolutions architecturales marquantes : on ne peut pas tout mettre au-dessus de la barre, ainsi parmi les méthodes d'accès seul VSAM bénéficie pleinement de l'adressage à 31 bits.

Les règles de communication entre programmes ne sont plus aussi simples qu'auparavant. Quand un programme appelle dynamiquement un autre programme, le mode d'adressage (indiqué dans le PSW) devient celui du programme appelé. Un appel statique ne fait pas changer le AMODE (le AMODE et le RMODE d'un load-module découlent des AMODE et RMODE des CSECTs réunies lors de l'édition des liens). Le AMODE d'un load-module détermine le mode d'adressage qui sera celui du programme une fois chargé ; cependant le programme peut par la suite modifier de lui-même le mode d'adressage grâce à l'instruction BSM :

```
* Passage à un adressage à 31 bits
        L    R1,LABEL1     charger adresse avec bit x'80'
        BSM  0,R1          passage en amode 31 bits
 LABEL1 DC   A(LABEL2+X'80000000') adresse de branchement plus flag
```

```
LABEL2   DS   0H
```

* Passage à un adressage à 24 bits

```
         LA   R1,LABEL1     charger adresse sans bit x'80'
         BSM  0,R1          passage en amode 24 bits
LABEL1   DS   0H
```

Les figures suivantes représentent :

- les possibilités d'appel dynamique entre programmes chargés en mémoire virtuelle sous ou sur la barre ;

- les combinaisons d'attributs possibles pour un load-module.

C) Propriétés des programmes

La possibilité d'être utilisé plusieurs fois (de suite ou simultanément) est une propriété importante des programmes. Elle permet de minimiser les entrées-sorties pour leur chargement et de les rendre disponibles en même temps à de nombreux utilisateurs.

Une première propriété importante est celle de réutilisation en série: un programme qui vérifie cette propriété peut être appelé plusieurs fois de suite dans un espace-adresse. Cela suppose deux choses:

* le programme a été déclaré "reusable" lors de l'édition des liens;

* le programme est effectivement réutilisable, c'est-à-dire que l'appel précédent n'a rien modifié dans son code ou ses données qui puisse altérer le déroulement de l'appel suivant; à cette fin on peut tolérer que lors de l'appel précédent certaines zones modifiées soient réinitialisées.

Un programme déclaré réutilisable au système est gardé en mémoire virtuelle, après chargement, en un seul exemplaire maximum. Il ne peut être appelé que par une seule tâche en même temps; toute tâche qui appelle un programme réutilisable déjà en cours d'utilisation est suspendue jusqu'à ce que le programme soit rendu disponible.

Un programme ré-entrant peut être appelé simultanément par plusieurs tâches s'exécutant sur différents processeurs. Cela implique qu'il ait été déclaré tel à l'édition des liens (option RENT) et que son code et ses données ne soient pas changés en pratique par l'exécution. Les données de travail de chaque appelant (si elles existent) sont stockées dans des zones différentes obtenues par GETMAIN. Une seule copie du programme en mémoire suffit à tous les utilisateurs et cette copie, non modifiée par les différents appels, peut résider dans une zone accessible uniquement en lecture (LPA). Un programme ré-entrant peut être normalement rafraîchi par le rechargement d'une copie neuve (option REFR de l'édition des liens): on n'envisage pas le cas d'école où la ré-entrance, simulée par une sérialisation des appelants (par ENQ/DEQ) de façon à n'avoir qu'un appelant à la fois exécutant le programme, ne serait qu'une "réutilisabilité" déguisée. A strictement parler, la réentrance est la faculté qu'a un programme de pouvoir être appelé plusieurs fois en même temps, tandis que l'attribut "refreshable" implique en plus que le programme ne se modifie pas lui-même, code et données figées (comme les modules de la PLPA). Dans la suite nous confondrons ces deux propriétés.

Tous les programmes du système qui sont susceptibles d'être invoqués par plusieurs tâches sur plusieurs processeurs sont ré-entrants (noyau, LPA). L'écriture des programmes ré-entrants est particulière: les macro-instructions à paramètres variables doivent être employées sous une forme "execute" par laquelle on positionne ces paramètres variables, associée à une forme "list" qui donne les paramètres fixes. La forme "list", elle-même parfois modifiable par une forme "modify", sert de squelette à la forme "execute"; le but est d'obtenir un appel avec des paramètres passés en dehors du code ré-entrant. Un programme ré-entrant est d'office réutilisable en série.

Les SGBD IMS et CICS ont une conception semblable à celle de MVS (souvent moins restrictive) pour la réutilisabilité et la ré-entrance. Il est important dans des systèmes transactionnels qu'on puisse utiliser plusieurs fois la même copie d'un programme sans devoir dégrader le temps de réponse en rechargeant une copie fraîche depuis un disque. En CICS la ré-entrance est de rigueur (une transaction peut être suspendue après un EXEC CICS et une autre transaction vouloir utiliser le même programme); en IMS la réutilisabilité suffit (les transactions s'exécutent les unes après les autres dans chaque région de message).

Voyons maintenant les propriétés réservées aux programmes orientés système. Certaines bibliothèques de load-modules peuvent être "autorisées" (APF: authorized program facility). Tout programme installé dans une telle bibliothèque aura accès à toutes les instructions: il pourra se mettre en mode superviseur, changer de clé ou de PSW, accéder ou modifier des zones mémoire protégées, etc. Pour qu'une étape d'un travail soit APF-autorisée, il faut cependant qu'elle remplisse deux conditions supplémentaires: que le programme appelé ait été "link-édité" avec l'option AC=1 (authorization code=1), et qu'aucune recherche de programme en bibliothèque non autorisée n'ait été faite dans l'étape (autrement dit, pour avoir l'autorisation, toutes les bibliothèques concaténées en STEPLIB devraient être autorisées), sans quoi l'autorisation est perdue. Cette dernière condition empêche qu'on puisse introduire par fraude une version modifiée d'un programme autorisé. D'autres logiciels permettent de contrôler et de limiter l'appel des programmes; seul l'APF est livré en standard en MVS et fournit le minimum de protection exigible. Les bibliothèques autorisées sont signalées dans l'lEAAPFxx de la PARMLIB (nom des fichiers et volumes de résidence) et une table APF est créée en mémoire au démarrage. Cette protection empêche tout module non autorisé de "casser" le système (tout au plus l'espace-adresse qui l'appelle).

Certaines propriétés particulières peuvent être assignées à des programmes autorisés; elles sont rassemblées dans une table particulière, la PPT (program properties table) qui est sous forme d'un module IEFSDPPT en LPA (créé en SYS1.LPALIB) ou d'un membre SCHEDxx de la PARMLIB (à partir de MVS-XA 2.2). Un programme peut être déclaré non-cancellable (emploi obligatoire de la commande MVS "FORCE" plutôt que "CANCEL"); protégé par une clé système (de 0 à 7); non-swappable; privilégié pour SRM (pas de swap sauf en cas de "long wait"); tâche système (pas de limitation d'exécution dans le temps, comme si on avait le paramètre TIME=1440); outrepassant tout contrôle d'accès fichier ("bypass protection password"); sans intégrité pour ses propres fichiers ("no dataset integrity") ce qui rend accessible aux autres utilisateurs même un fichier qu'il détient en DISP=OLD (aucune ENQUEUE n'est générée) - cela peut être commode par exemple pour une PROCLIB de JES; enfin on peut désigner les processeurs sur lesquels le programme peut s'exécuter (CPU affinity). Ces propriétés s'appliquent à l'étape qui invoque le programme autorisé dans l'ordre EXEC du JCL.

D) Les programmes "système" (SVC, exits)

Nous entendons par "programmes système" les programmes qu'une installation peut rajouter à un système MVS pour répondre à certains besoins spécifiques. Les moyens standards qu'offre MVS sont relativement limités et bien contrôlés: les routines de SVC, les exits.

Les routines SVC sont un moyen commode de développer un service facile à appeler (instruction SVC) et pour lequel une autorisation est nécessaire. Une SVC reçoit le contrôle en mode superviseur et clé zéro, ce qui lui permet de tout faire. IBM emploie pour son propre usage un peu plus des 100 premiers numéros, ce qui laisse à peu près autant de "points d'entrée" disponibles pour des SVCs propres au site. Officiellement, seuls les SVCs de numéro 200 à 255 sont utilisables, mais certains logiciels implantent des SVCs de numéro inférieur.

Les SVCs se présentent comme des modules présents en SYS1.LPALIB ou incorporés au noyau IEANUCxx. Leurs propriétés sont indiquées dans le membre IEASVCxx de la SYS1.PARMLIB (avant MVS-XA 220, une génération de MVS était nécessaire).

On distingue 6 types de SVC (le type 5 correspondant à un point d'entrée vide). Les SVCs de types 1, 2 et 6 résident dans le noyau (IEANUCxx) ; on les réserve pour des routines devant fonctionner ininterruptibles (seules celles de type 6 le sont constamment). Les SVCs de type 3 et 4 sont en LPA: elles peuvent supporter des fautes de page (quand elles sont interruptibles). Les types 3 et 4 sont quasiment identiques (une SVC de type 4 peut comprendre plusieurs load modules, une de type 3 est restreinte à un seul). Les SVCs peuvent elles-mêmes appeler d'autres SVCs (sauf celles de type 6, ininterruptibles, et de type 1, qui détiennent le verrou LOCAL).

L'écriture des routines SVCs obéit évidemment à des règles contraignantes: ré-entrance, nom du load-module imposé, mise en bibliothèque système. MVS construit une table (SVCtable) pour maintenir les caractéristiques des 256 routines possibles: y figurent le type de la routine, les verrous que MVS (le FLIH) doit acquérir avant l'appel, le mode d'adressage et un code indiquant si l'autorisation APF est requise, ce qui permet de restreindre l'utilisation de la routine; de même et de façon plus souple une tâche peut interdire l'emploi d'une SVC à ses sous-tâches (SVC screening).

En principe une routine SVC est installée en mémoire virtuelle une fois pour toutes au démarrage. IBM propose depuis MVS-XA une macro-instruction SVCUPDTE qui permet (sans devoir arrêter MVS) de changer les caractéristiques d'une SVC, d'en charger une nouvelle ou de détruire une ancienne (options EXTRACT, REPLACE, DELETE). Les modifications ainsi apportées à la SVCtable sont notées pour mémoire dans une table spéciale (SVC recording table). Certains logiciels modifient directement la SVCtable pour installer "en frontal" de la SVC une routine d'interception qui apporte des fonctionnalités supplémentaires (on trouvera même parfois des "frontaux de frontaux").

IBM a trouvé un moyen d'augmenter le nombre de SVCs possibles en faisant bénéficier certaines routines des caractéristiques des SVCs : les "extended SVC entries", dont le nom doit être de la forme IGX00nnn (nnn=200 à 255, les valeurs de 0 à 199 sont réservées pour IBM) qui doivent être appelées par l'intermédiaire d'une SVC de routage (ESR, extended SVC router) : SVC 109 (types 3 et 4), SVC 116 (type 1), SVC 122 (type 2), SVC 137 (type 6). Avant l'appel de la SVC de routage, le numéro de routine nnn doit être renseigné dans le registre 15.

Les exits sont des points d'entrée "standards" dans différents composants du système; ils sont destinés aux besoins particuliers du site. Le constructeur a préféré laisser cette possibilité plutôt que de voir son code modifié inconsidérément par le client avec des "zaps" (ou "patches") dangereux et difficiles à maintenir par leurs auteurs mêmes. Au contraire l'exit est un programme dont l'emploi est soigneusement contrôlé: s'il est présent il est invoqué en lui passant les paramètres qui lui sont nécessaires (en se protégeant au besoin contre toute erreur par une routine de reprise); absent il est ignoré ou remplacé par une routine par défaut qui n'accomplit rien. Le constructeur fournit de nombreux points d'entrée pour les exits. Cette politique obéit à la tendance qu'il a de ne plus fournir les "sources" de ses logiciels ("object code only"), ce qui est somme toute une protection élémentaire de sa propriété (mais qui peut gêner les "bricoleurs" et les curieux autant que la concurrence...).

Voici quelques exemples d'exits très courants en MVS:

* IGGPRE00 : exit de pré-allocation de nouveaux fichiers ;
* IKJEFF10 : exit de soumission de travaux sous TSO ;
* IKJEFLD : exit de connexion à TSO ;
* IEFDB401 : exit de l'allocation dynamique des fichiers ;
* IEFUJI : exit SMF (job initiation) ;
* IEFUJV : exit SMF de validation des travaux ;
* IEFUTL : exit SMF (expiration du temps machine accordé à un travail) ;
* IEFACTRT : exit SMF de fin de step ou de fin de job ;
* IEALIMIT : limite la région privée sous la barre des 16 Méga-octets ;
* IEAVMXIT, IEECVXIT: exits de gestion des messages console ;
* IFG0EX0A : exit d'ouverture de fichier (DSCB non trouvé) ;
* IFG0EX0B : exit d'ouverture de fichier ;
* ICHRCX01 : exit "RACHECK preprocessing" ;
* ICHRTX01 : exit SAF ;
* IATUXnn : exits JES3.

Pour plus de clarté, les programmes systèmes SVC ou exits devraient être sur des librairies bien identifiées, en link-list ou Ipa-list, plutôt que dans les librairies du système MVS (LINKLIB, LPALIB). Une autre possibilité est de les installer par le logiciel SMP comme "usermod" (user modification).

L'évolution normale d'un système d'exploitation doit conduire à éliminer les exits au profit d'un paramétrage plus riche ou de nouvelles fonctions ; il ne faudrait employer les exits qu'en tout dernier ressort (dans certaines recommandations d'lBM on peut lire: règle numéro 1: ne pas envoyer les exits; règle numéro 2 : en cas de

problème se reporter à la règle numéro 1...). Il faut au moins identifier les exits qui sont implantés sur le système, savoir les ré-installer, et aussi savoir les désactiver si nécessaire. Un exit n'est pas un moyen correct de "customiser" un logiciel. Il s'agit d'un pis-aller, d'une "verrue légale" qui complique le travail d'installation et de maintenance : l'exit doit être réimplanté et adapté à chaque changement de version (avec tous les risques d'erreur concomitants). Cependant ce peut être parfois un moyen très puissant d'imposer certaines règles ou certains traitements qui n'ont pu être intégrés autrement dans le produit.

On peut distinguer 3 sortes d'exits:

* les exits propres à un logiciel, qui n'impactent qu'un nombre réduit d'utilisateurs;

* certains exits en LINKLIB ou link-list (exit de SUBMIT de TSO, exits MPF, etc.), dont la modification pose peu de problèmes (attention à la prise de nouveaux extents pour ces bibliothèques, et au fait qu'on ne peut compresser sans risque la SYS1.LINKLIB);

* les exits en LPA, qui doivent être soigneusement testés, car leur modification ne peut être en principe prise en compte que par un IPL.

E) La gestion des modifications (SMP)

Les modules du système MVS livrés par IBM sont des programmes complexes et très nombreux, et il est inévitable qu'ils comportent certaines imperfections ou certaines erreurs. Le constructeur y remédie en livrant des APARs (authorized program analysis report), corrections ponctuelles et en diffusant des PTFs ("program temporary fix" souvent issues d'APARs) qui sont des corrections préventives, des améliorations ou des fonctionnalités supplémentaires livrées la plupart du temps sous forme de bande PUT (program update tape) envoyées à intervalles réguliers (une dizaine par an). Ces "PTFs" sont temporaires car elles sont destinées à être intégrées dans une version ou une release future du logiciel concerné. Notons que la correction d'une erreur peut en entraîner une autre et que certaines PTFs ("PE": program error) sont elles-mêmes corrigées par d'autres. L'application de toutes ces corrections est effectuée par un logiciel particulier, SMP (system modification program), qui connaît l'imbrication des unités logicielles de base (sources, macros, modules, load-modules). Ainsi une modification (SYSMOD) sur un élément entraîne une modification sur tous les éléments impactés: un changement dans un module est répercuté sur tous les load-modules qui contiennent le module, SMP se chargeant de faire exécuter toutes les éditions de liens; un changement dans une macro peut entraîner le ré-assemblage automatique de tous les sources qui l'utilisent, etc. Le travail du client est simplifié (il n'a pas besoin de connaître les relations entre les éléments) et standardisé par l'emploi d'une procédure unique commune à tous. SMP note le niveau de maintenance appliquée sur chaque élément, ce qui est utile pour le personnel du support logiciel IBM. SMP est enfin l'outil privilégié pour l'installation d'un nouveau logiciel ou d'un système (voir le chapitre 13).

Il y a deux sortes de bibliothèques système maintenues par SMP: les bibliothèques de distribution (DLlBs, distribution libraries) qui contiennent la version d'origine des logiciels livrés par le fournisseur, et qui servent à générer le produit ou un élément du produit; les bibliothèques cibles (target libraries) issues des précédentes, qui représentent le logiciel effectivement utilisé par le client; les DLlBs, elles, ne sont utilisées que pour une génération de produit ou un retour arrière sur une PTF (restore).

SMP stocke ses informations sur un ou plusieurs fichiers VSAM KSDS: les CSIs (consolidated software inventory). Les données de base de SMP peuvent appartenir à trois zones différentes dans les fichiers CSls:

* une zone cible (target), qui reflète le niveau des bibliothèques cibles, donc le niveau en cours des logiciels du site;

* une zone de distribution, qui reflète l'état des bibliothèques de distribution;

* une zone globale, qui référence les zones cibles et distribution et contient des paramètres généraux ainsi que les noms des produits installés ou installables (FMlDs).

La marche à suivre pour appliquer une modification ou installer un nouveau produit est la suivante:

* réception (receive) en zone globale à partir du SMPPTFIN livré au client contenant les PTFs ou le produit, et du SMPHOLD (signalant les PTFs en erreur); le fichier disque SMPPTS est alimenté, éventuellement des fichiers SMPTLlBs s'il s'agit d'une distribution de produit par la technique relative (qui évite l'engorgement du SMPPTS, plutôt adapté aux PTFs). La zone globale du CSI reflète les éléments reçus. Le retour arrière peut être effectué par un "reject". A cette étape, le système cible n'est pas encore modifié;

* application (apply) des PTFs ou du produit sur les bibliothèques du système (target libraries) et mémorisation dans la zone cible; auparavant SMP aura vérifié le niveau des éléments et la présence de toutes les SYSMODs requises. En effet les SYSMODs qui s'appliquent sur une unité donnée (source, macro, module) ne sont pas des modifications indépendantes (qui pourraient se révéler contradictoires entre elles) mais sont obligatoirement liées les unes aux autres par des relations d'environnement: les unes précèdent les autres (prerequisits); ou doivent être installées ensemble (requisits); certaines en remplacent d'autres (supercede) en les corrigeant s'il s'agit de PTFs en erreur. L'application peut être annulée par un "restore" (qui rétablit les éléments modifiés à partir des bibliothèques de distribution, toujours intactes);

* enfin, après une période de tests, on peut décider de l'installation définitive du produit par une acceptation (accept), qui met à niveau les bibliothèques de distribution et la zone correspondante. On ne peut alors plus revenir en arrière sur le niveau du produit ou sur les modifications.

L'application de modifications par SMP est assez lourde et, il faut le reconnaître, aboutit souvent à des travaux très longs. C'est cependant la seule façon de conserver un environnement cohérent et de connaître le niveau de maintenance du système.

On peut réunir sur un seul CSI les 3 types de zones. On peut aussi mettre chaque zone sur un CSI particulier. Ce dernier type de structure est recommandé pour une maintenance plus cohérente et plus sûre d'un système MVS. On dispose d'un CSI pour la distribution (placé sur le volume DLIB), un pour la zone globale, et deux CSls pour deux zones cibles, placés sur chaque disque système dont ils reflètent l'état (un disque pour le système en cours, un autre servant soit de disque de secours, soit de disque de test pour un système de niveau plus avancé). On est ainsi assuré en cas de problème sur un disque système de connaître le niveau résultant d'une restauration complète du disque (le CSI cible étant en phase avec les fichiers restaurés). Il est toujours préférable, dans le cas d'une maintenance "sensible" (c'est-à-dire qui impacte notamment les fichiers LINKLIB, LPALIB, NUCLEUS) et qui nécessite un arrêt du système pour être prise en compte, d'utiliser un autre disque que le disque
système actuel, qu'on sait être "sûr", et de dupliquer les fichiers cibles et le CSI cible. L'application des modifications sur un système cible différent du système présent apporte le maximum de sécurité. La même recommandation élémentaire vaut pour l'installation d'une nouvelle version d'un produit, l'opération d'apply ayant pour effet de détruire la version précédente.

Certaines fonctions de SMP (nous parlons ici de la version courante SMP/E et non de l'ancien SMP/4) sont particulièrement intéressantes:

JCLIN: renseigne le CSI à partir d'un JCL d'installation comprenant des étapes d'assemblage, édition des liens ou copie. Le JCL est analysé pour recueillir les modes d'installation de tous les éléments. Ainsi est déterminée l'imbrication des divers éléments entre eux.

GENERATE: inversement un JCL est créé à partir des informations du CSI pour construire un jeu de bibliothèques cibles à partir de celles de distribution. Cela peut être utile si le système cible (ou une partie du système) est endommagé .

Les SYSMODs installés sur un système peuvent être:

* des APARs ou des PTFS;
* des modifications apportées par l'utilisateur lui-même: USERMODs;
* des "fonctions", c'est-à-dire des logiciels ou composants du système.

Le schéma décrit les étapes d'installation d'une SYSMOD, avec les nombreux fichiers de travail dont SMP a besoin au cours de l'installation des modifications. Voici ces fichiers

* SMPPTFIN contient les modifications ou produits livrés par le fournisseur, sous forme de fichiers sur bande ou cassette.

* SMPHOLD, sur le même support, contient des informations sur les PTFs en erreur ou sur d'autres "états d'exception" (hold data): il s'agit de contraintes lors de l'application de certaines PTFs: une action à entreprendre (ACTION), la nécessité d'appliquer une PTF sur un produit non forcément connu de ce CSI (DEP), une génération de MVS à faire (IOGEN, FULLGEN), ou simplement la mise à jour de la documentation (DOC).

* SMPPTS (SMP PTF temporary storage) et les fichiers SMPTLlBs contiennent les sysmods déchargées par le RECEIVE.

* SMPMTS et SMPSTS contiennent respectivement des macros et des sources modifiés par l'APPLY; ce sont en fait des extensions du système cible qui permettent de prendre en compte dès l'APPLY des modifications portant sur des macros et des sources n'existant que dans les bibliothèques de distribution .

* SMPSCDS sauvegarde certaines informations sur les éléments du CSI cible qui ont pu être modifiés à l'APPLY par un JCLIN "in line"; ce genre de JCLIN (qui fournit un modèle de JCL pour effectuer un APPLY) a pu établir des relations différentes entre les éléments. Il faut pouvoir retrouver, lors d'un éventuel RESTORE, les anciennes relations de manière à rétablir correctement les éiéments modifiés.

F) Le débogage

Tout programme est susceptible de comporter des erreurs, de traiter des données ou des paramètres incorrects qui l'empêchent de s'exécuter normalement. Le programme est alors stoppé et subit une fin anormale: un ABEND (abnormal end). L'ABEND peut être déclenché par la macro ABEND par le programmeur (user abend) ou par le système MVS (system abend). Un code sur 3 digits est censé indiquer la cause de l'arrêt anormal : il est affiché en hexadécimal (Sxxx) ou en décimal (Udddd).

Pour aider à la résolution du problème, MVS émet parfois des messages d'erreur, et le plus souvent il fournit un "dump": il s'agit d'une copie de la mémoire et de certains blocs de contrôle au moment de l'ABEND. Le but est d'aider à la correction de l'erreur (débogage) en fournissant une image de l'environnement d'exécution du programme au moment de l'ABEND. Le dump est écrit sur les fichiers associés aux DDNAMEs SYSUDUMP, SYSABEND ou SYSMDUMP. En SYSUDUMP est fourni l'environnement de la tâche en ABEND, programmes, blocs de contrôle, registres. En SYSABEND on rajoute la LSQA et certains blocs du noyau. En SYSMDUMP on fournit un DUMP beaucoup plus complet sous une forme non formattée: il faut traiter ce fichier DUMP par l'utilitaire AMDPRDMP ou le logiciel IPCS (interactive problem control system) pour obtenir les zones ou blocs de contrôle qui intéressent le programmeur. IPCS est un outil privilégié (obligatoire en ESA) pour traiter les dumps non formattés: il permet d'accéder de façon interactive à toutes les zones ou blocs intéressants sans avoir à parcourir une grande masse d'informations comme c'est le cas avec les dumps formattés. On peut de plus, par la commande VERBEXIT, appeler des routines spéciales qui formattent les blocs de contrôle de composants ou logiciels particuliers (IMS, TSO, CICS, VTAM, etc.).

Les ABENDs dans les routines de MVS provoquent des "SVC dumps", eux aussi non formattés, qui sont dirigés vers des fichiers particuliers, les fichiers SYS1.DUMPxx (sur disque ou plus rarement sur bande). Le DAE (dump analysis and elimination) introduit en MVS-XA permet d'éviter de prendre des dumps plusieurs fois pour le même problème. Le SVC dump est provoqué par la macro SDUMP ou par l'opérateur avec la commande MVS "DUMP", qui permet d'ailleurs à tout moment de dumper un espace-adresse sans provoquer sa fin. Cependant toute prise de SVC dump met toutes les tâches du système dans un état non dispatchable le temps de la prise du dump, pour éviter toute modification en mémoire qui ne rendrait pas le dump conforme à l'état du système au moment du problème. Cette mise en attente paraît "bloquer" le système quelques secondes (elle a été diminuée en ESA).

Enfin, dans les cas désespérés où MVS semble ne plus répondre, on peut prendre un dump "stand-alone" (dump autonome), formatté ou non, de tous les espaces-adresses et des zones communes. Ce programme n'est pas sous le contrôle de MVS, il est copié sur disque et chargé en mémoire par un IPL.

Le débogage est un art difficile et incertain. Les informations les plus importantes et qu'on doit examiner en premier dans un dump sont le PSW, les registres et le chaînage des save areas. Le PSW indique l'instruction qui allait être exécutée, et dans la plupart des cas cela suffit pour déterminer le programme et l'endroit dans ce programme où est survenu le problème, et par là la cause du problème. Les problèmes d'écrasement de données en mémoire ou de "branchement sauvage" sont les plus ardus. Dans certains cas le dump ne suffit pas à résoudre le problème (on aurait préféré avoir un dump de la mémoire quelques instructions avant...). Il faut alors recourir à d'autres méthodes, et MVS fournit de nombreux instruments pour aider au débogage. Voici les principaux:

* le snap dump: le programmeur, par la macro SNAP, provoque un dump formatté de certaines zones à l'endroit du programme où il le souhaite;

* le SLIP (serviceability level indication processing), orienté système, permet de "piéger" l'exécution d'une instruction, un branchement, la modification d'une zone en mémoire, un ABEND ou une autre erreur;

* les traces dont notamment:

-- la trace système: garde en mémoire (zone commune ou espace-adresse TRACE) les événements les plus récents: demandes d'entrée-sortie, interruptions, dispatchings, etc. On peut ainsi étudier dans le dump les événements qui ont précédé l'erreur. Toutes proportions gardées, la trace équivaut à la "boîte noire" des avions;

-- la master trace: permet d'avoir les messages console qui ont précédé l'erreur;

-- la trace GTF: il s'agit d'un programme s'exécutant à la demande dans un espace-adresse et qui peut pister toutes sortes d'événements à un niveau très fin. Son emploi doit être limité et précis, car les "hooks" (crochets dans le système) qu'il utilise entraînent une interruption programme traitée par GTF; tout abus provoque une déperdition de ressources importantes. GTF stocke ses traces soit en mémoire, soit sur un fichier trace, à traiter par AMDPRDMP, IPCS, ou par programme utilisateur;

* le fichier SYS1.LOGREC enregistre les erreurs logicielles et matérielles; il peut être consulté par le programme EREP (environmental recording editing and printinq) et on peut mettre ses données en historique.

Ces moyens de débogage peuvent être combinés entre eux: le SLIP par exemple peut renseigner la LOGREC ou la trace GTF.

La plupart des langages de programmation évolués comportent aussi de nombreuses aides au débogage initialisées par des paramètres de compilation particuliers. Pour l'assembleur, citons, sous TSO, la commande TEST qui permet d'appeler un programme avec des points d'arrêt pour contrôler sa bonne exécution .

Dans la plupart des programmes systèmes, pour lesquels une erreur peut être très dangereuse, on se protège de l'abend par des routines de reprise. La macro ESTAE notamment permet de reprendre le contrôle pour analyser et éviter l'abend (voir le chapitre 11).

7. LA GESTION DES CATALOGUES

A) Généralités

Un catalogue répertorie chaque fichier qu'on lui fait connaître et retient le minimum d'information requis pour que tous les travaux puissent accéder à ce fichier en ne connaissant de lui que son nom: ces informations sont au moins le nom du volume où se trouve le fichier et le type de l'unité sur laquelle doit être monté le volume. Pour certains types de fichiers sur disque (les VSAMs), le catalogue enregistre des caractéristiques supplémentaires: étendue du fichier, longueur de l'enregistrement, type de fichier, etc. Pour les autres le système trouvera ces informations supplémentaires dans les VTOCs des disques, les labels de bande, ou simplement le JCL; le catalogue évite principalement à l'utilisateur d'avoir à mémoriser le nom du volume sur lequel est chacun de ses fichiers et de devoir l'indiquer dans son JCL.

Une installation informatique courante gère souvent des milliers de fichiers différents sur de nombreux volumes. L'utilisation de catalogues (évidemment placés sur disque) est alors indispensable. Un seul catalogue pourrait suffire à répertorier tous les fichiers du site. En pratique, pour éviter de trop nombreux accès à un catalogue unique et éviter de centraliser l'information, on préfère disposer de deux niveaux de catalogues: des catalogues "utilisateurs" (USERCATs), chacun gérant un certain nombre de fichiers; un catalogue maître (MASTERCAT) dans lequel sont catalogués les USERCATs. La recherche d'un fichier en catalogue passe d'abord par le catalogue maître (qui est le seul catalogue obligatoire en MVS), qui indique le catalogue utilisateur où est référencé le fichier, ceci le plus souvent en fonction du premier qualifiant du nom de fichier (alias). Les fichiers systèmes nécessaires au démarrage de MVS sont les seuls à devoir être catalogués directement dans le MASTERCAT, qui est un catalogue relativement invariant.

Il existe actuellement en MVS trois types de catalogues:

* le CVOL (catalog volume): une ancienne structure héritée de l'OS. Le CVOL enregistre les fichiers que contient un volume. Son nom est imposé: SYSCTLG.Vxxxxxx (où xxxxxx est le volser du volume disque contenant le CVOL). Ce type de catalogue devient de plus en plus désuet à cause de ses limitations: manque de souplesse quant aux niveaux de qualifiants (si un fichier X.Y y est catalogué, alors on ne peut cataloguer un fichier X.Y.Z); mauvaises performances (une entrée-sortie est nécessaire par niveau de qualifiant); support insuffisant par les utilitaires MVS; impossibilité d'employer un CVOL en JOBCAT ou STEPCAT: impossibilité d'avoir plus d'un CVOL par volume: impossibilité enfin de cataloguer un fichier VSAM dans un CVOL.

* le catalogue VSAM: il tire parti de la méthode d'accès VSAM (c'est un KSDS), qui le rend bien plus souple que le CVOL. Une contrainte importante avec ce type de catalogue est qu'il "s'approprie" un volume, dans le sens où tous les fichiers d'un volume ne peuvent appartenir qu'à un seul catalogue VSAM (catalog ownership). Tous les objets VSAM doivent être contenus dans un "space"; ainsi est établie la relation avec le catalogue propriétaire.

En cas d'erreur grave, on peut restaurer les informations du catalogue (s'il a été déclaré "recoverable") à partir d'une zone de sauvegarde, la CRA (catalog recovery area). Une CRA est un ESDS placé sur chaque volume dont un catalogue VSAM recouvrable est propriétaire; la sécurité est établie par duplication des informations de catalogage. Une commande IMPORTRA permet de restaurer le catalogue.

* le catalogue ICF (integrated catalog facility), type de catalogue spécifique à MVS depuis MVS/SP (alors que VSE emploie aussi les catalogues VSAM). L'ICF lève les contraintes des précédents types de catalogue; notamment les fichiers d'un volume peuvent appartenir jusqu'à 36 catalogues ICF différents. L'indépendance avec le support physique est accrue: réorganisation et déplacement aisés, suppression du contrôle de synchronisation avec le volume, gestion plus souple, presque banalisée (commandes REPRO, EXPORT, IMPORT).

Un catalogue ICF est composé de deux parties différentes:

- le BCS (basic catalog structure), qui est un KSDS contenant des informations strictement "de catalogage", assez statiques: le nom des fichiers catalogués, leur volume de résidence, leur détenteur, les mots de passe pour y accéder. Pour les fichiers VSAM, l'information est organisée en "sphères" qui réunissent tous les composants VSAM associés à un cluster (par exemple ses index alternés). La sauvegarde d'un BCS s'effectue par un export.

- les VVDS (VSAM volume data set); un VVDS est un ESDS qui contient des informations sur les fichiers VSAM nettement orientées "volume" (extents, plus grande RBA, etc.) et qui sont mises à jour pour toute ouverture ou fermeture de VSAM. Chaque volume sur lequel se trouve un fichier VSAM contient un unique VVDS (créé automatiquement sur le volume en même temps que le premier VSAM) qui référence tous les VSAMs du disque catalogués dans différents catalogues ICF: c'est en quelque sorte une VTOC des fichiers VSAM, en plus de la VTOC disque habituelle. Le VVDS ne contient pas d'information sur les fichiers non VSAM, le BCS et la VTOC étant suffisants (sauf avec DFSMS).

Le VVDS comporte un VVCR (VSAM volume control record) qui décrit tous les catalogues ICF qui possèdent des fichiers VSAM sur le volume, puis des VVRs (VSAM volume record) pour décrire les fichiers VSAM eux-mêmes. Avec SMS on trouvera des NVRs (NONVSAM volume record) pour les fichiers SMS non VSAM.

Cet éclatement des informations de catalogage (BCS orienté catalogue et VVDS orienté volume) permet d'avoir un catalogue cohérent même après restauration complète d'un volume. La restauration d'un ICF peut s'effectuer sans avoir à stopper tous les travaux qui utilisent des fichiers catalogués; on peut de plus empêcher tout accès tant que l'ICF n'est pas restauré (ALTER LOCK). Cela demeure tout de même une opération délicate, surtout pour la partie VVDS qui ne peut être restaurée facilement, sauf restauration du disque entier ou restauration par IMPORT de tous les VSAMs du disque... La commande AMS DIAGNOSE permet de contrôler les différences de niveau entre VVDS et BCS ou entre VVDS et VTOC. L'outil ICFRU permet de reconstruire un catalogue ICF à partir d'une version de sauvegarde et des fichiers SMF.

Le catalogue ICF est le type de catalogue vers lequel IBM pousse ses clients: il est destiné à remplacer les autres types, qui peuvent être aisément convertis en ICF (commande CNVTCAT). Le MASTERCAT peut être un catalogue VSAM ou ICF. Le MASTERCAT a la même structure que tout autre catalogue; la différence réside dans le fait qu'il a été désigné comme "maître" lors du démarrage de MVS.

Tout fichier catalogué devrait l'être dans un seul catalogue. Rien n'empêche cependant (avec les ICFs) qu'il le soit dans plusieurs. Ceci permet en jouant sur les JOBCATs/STEPCATS d'accéder à différents fichiers portant le même nom. On peut aussi se créer des catalogues de secours par copie de catalogues existants (et une copie de sauvegarde du MASTERCAT, catalogue le plus important, est toujours utile). Un même fichier physique peut ainsi être catalogué dans plusieurs ICFs (c'est déconseillé). En modifiant les informations alias du MASTERCAT (qui font la correspondance entre les premiers qualifiants des noms de fichiers et les USERCATs), on peut diriger la recherche vers un USERCAT plutôt que vers un autre. Un fichier VSAM aussi peut être catalogué dans plusieurs ICFs; cependant le VVDS ne mentionne pour un VSAM qu'un seul BCS de catalogage, le plus récent. La destruction physique d'un VSAM ne peut se faire qu'en passant par ce catalogue.

Pour les fichiers VSAM, le catalogue contient toutes les informations indispensables pour accéder aux fichiers; ces informations sont remises à jour à la fermeture du fichier. Dans certains cas, un fichier VSAM accédé en mise à jour peut ne pas être fermé correctement (interruption machine, ABEND), et le catalogue reflète alors des informations incorrectes. Ces informations doivent être rectifiées pour éviter au moins de perdre des enregistrements situés après la fin de fichier (supposée). Cet état incohérent est détecté par un indicateur du catalogue, resté positionné et signalant que le fichier est ouvert en écriture (ce qui peut aussi provenir d'un accès concurrent depuis un autre système MVS). La commande AMS VERIFY peut être utilisée pour corriger les informations fin de fichier (EOD, end of data), dernière clé (EOKR, end of key range), plus haute RBA utilisée. Pour ce faire, le fichier VSAM est examiné depuis la plus haute RBA connue au catalogue jusqu'à la fin du fichier.
Actuellement le VERIFY est fait implicitement à l'ouverture du fichier.

En MVS-XA un espace-adresse CATALOG a été introduit pour soulager les zones CSA et PLPA: on peut par commande MVS lui demander de fermer un catalogue ICF ou un VVDS. Ce CAS (CATALOG address space) mémorise les entrées utilisées pour éviter autant que possible la relecture sur les catalogues. Avec des

catalogues partagés par plusieurs MVS, la resynchronisation se fait en invalidant les entrées mémorisées (seulement les plus récentes avec DFP V3).

B) Utilisation des catalogues

Un catalogue VSAM ou ICF est créé par la commande AMS "DEFINE USERCATALOG" (avec le paramètre ICFCAT pour les ICFs). Un CVOL est catalogué au MASTERCAT comme un fichier non VSAM. Tout catalogue créé par un autre système MVS peut être reconnu par la commande "IMPORT CONNECT". La commande "DEFINE ALIAS" permet de préciser les qualifiants rattachés aux USERCATs. Une fois les alias définis, tous les fichiers dont le premier qualifiant correspond à l'alias seront automatiquement catalogués dans le USERCAT associé à l'alias: sans cela le catalogage s'effectue au MASTERCAT. Les alias doivent être choisis soigneusement et correspondre à une stricte normalisation. En MVS, le premier qualifiant d'un fichier catalogué n'est pas une information indifférente. S'il n'est pas reconnu comme alias, le catalogage s'effectue dans le MASTERCAT (c'est le cas pour les fichiers systèmes commençant par SYS1). Avec DFP V3 un alias peut être constitué de 1 à 4 qualifiants (multiple level alias).

La mise à jour des catalogues s'effectue de plusieurs façons:

* par le paramètre DISP du JCL quand il prend les valeurs CATLG, UNCATLG ou DELETE.

* par la macro CATALOG (qui génère une SVC) employée dans un programme (pour les fichiers non VSAM).

* par les commandes AMS "DEFINE" qui permettent de créer des fichiers VSAM ou non VSAM. Ces derniers sont enregistrés au catalogue mais ne sont pas créés physiquement sur le volume. La commande DELETE, elle, détruit physiquement les fichiers (avec le paramètre NOSCRATCH elle se contente de les supprimer du catalogue).

Signalons un catalogage particulier (par DEFINE NONVSAM) qui est très utile pour les fichiers systèmes résidant sur un même "disque système": l'identification indirecte (indirect volser identification). Ces fichiers sont enregistrés au catalogue avec un nom de volume fictif ****** et un type d'unité 0000. Le système "reconnaîtra" alors que ces fichiers sont placés sur le disque système (SYSRES), quel que soit le nom que celui-ci puisse porter et quel que soit le type de disque. On rend ainsi le MASTERCAT indépendant du disque système utilisé au démarrage de MVS.

Les différents objets qui peuvent être catalogués sont:

- les catalogues utilisateurs, les clusters, les index alternés, les chemins (un "path" est une combinaison entre un index alterné et un cluster de base), les fichiers page ("pagespace");
- les fichiers non VSAM, les alias, les GDGs (voir plus loin).

La méthode de recherche d'un fichier catalogué dans les catalogues du système est la suivante:

- recherche dans le catalogue explicitement spécifié par la macro ou la commande AMS, le cas échéant;
- puis dans le ou les catalogues spécifiés en //STEPCAT;
- si aucun stepcat n'a été indiqué, recherche en //JOBCAT;
- si le premier qualifiant est un nom de USERCAT ou au ALIAS lié à un USERCAT ou un CVOL (cas le plus fréquent et le plus normal en MVS), recherche dans le catalogue utilisateur correspondant;
- enfin recherche au MASTERCAT.

L'emploi des STEPCATs/JOBCATs (survivance de l'OS/VS1 où il n'y avait pas d'autre moyen de cataloguer les fichiers courants) devrait être évité en pratique, sauf pour des cas très particuliers, par exemple l'installation d'un nouveau système MVS, pour avoir un second jeu de bibliothèques facilement accessibles. Il fausse en le contournant le jeu naturel des alias et, en créant une dépendance dans le JCL, empêche de transférer le usercat sur un autre disque et sous un autre nom (ce qui est très commodément effectué en recopiant le catalogue par un REPRO) .

Le passage par le catalogue introduit certainement une déperdition (overhead) par rapport à un accès direct au volume (par l'emploi dans le JCL des paramètres UNIT et VOL=SER). Le catalogue rend cependant les JCLs indépendants des volumes où résident les fichiers: on peut ainsi déplacer les fichiers de façon transparente pour les JCLs qui les invoquent. De plus le catalogue (VSAM ou ICF) est absolument indispensable pour accéder aux fichiers VSAM. Enfin, dans l'optique d'une gestion automatique des espaces magnétiques (avec DFHSM ou DFSMS), le catalogage des fichiers devient une nécessité.

C) Les fichiers à génération (GDG)

Le fichier à génération est un type de fichier catalogué non VSAM utilisé depuis longtemps en MVS. Les caractéristiques des fichiers GDG (generation dataset group) sont les suivantes:

* ils portent tous le même nom (GDG base), et se distinguent entre eux par le numéro de génération, maintenu par le système, qui l'incrémente à mesure que de nouveaux fichiers sont créés.

* le système peut détruire automatiquement les générations les plus anciennes, et maintenir un nombre constant de fichiers vivants (option SCRATCH).

Cette gestion automatique d'une chronologie de fichiers est le trait le plus intéressant, et qui explique le succès que les fichiers à génération connaissent dans tous les sites MVS.

Le nom d'un fichier GDG est le suivant: X.GnnnnVpp où X est le nom de base de GDG, nnnn le numéro de génération (de 0001 à 9999) et pp le numéro de version (de 00 à 99). Ainsi X.G0001V00 est la plus ancienne génération du GDG X qui puisse exister.

En pratique on emploie plus souvent des numéros relatifs: X(0) représentera la génération la plus récente, X(-1) la précédente, etc. X(+1) sera employé pour créer une nouvelle génération.

Le numéro de version permet d'avoir plusieurs versions de fichiers par génération: la création d'un fichier X.G0001V01 permettra le remplacement au catalogue du fichier X.G0001 V00 (qui n'en sera pas détruit physiquement pour autant).

Les différentes générations peuvent être des fichiers d'organisations et de types différents. Toutefois, avec des organisations, des attributs de fichiers et des supports physiques identiques, on peut accéder à toutes les générations existantes (255 au maximum) comme s'il s'agissait d'un seul fichier, en omettant dans le DSNAME le numéro relatif (ou absolu) de génération.

Un GDG base est créé par l'ordre BLDG de l'utilitaire IEHPROGM ou plus simplement par la commande DEFINE GDG d'AMS. Une conception un peu ancienne oblige encore, pour créer une nouvelle génération relative, à posséder un modèle de label pour établir à la création tous les attributs du fichier (taille d'enregistrement, de bloc, etc.). On peut créer sur le volume où réside le catalogue qui référence le GDG (à l'exclusion de tout autre volume) un fichier de même nom que la base du GDG: par la suite toute génération du GDG prendra à sa création les attributs de ce "DSCB model". La contrainte est de devoir déplacer le modèle en même temps que le catalogue. Aussi préfère-t-on souvent créer un modèle unique (qui est un fichier catalogué créé uniquement à cette fin) et préciser systématiquement dans le JCL ce modèle avec éventuellement des paramètres de remplacement. Par exemple:

```
//NOUVEAU DD DISP=(NEW,CATLG),DSN=GDG(+1),
// UNIT=DISK,SPACE=(TRK,1),
// DCB=(MODELE.UNIQUE,LRECL=80,BLKSIZE=6160,RECFM=FB)
```

Avec DFP v3 et SMS cette contrainte tombe; de plus le nombre de générations à conserver (LIMIT du GDG) peut être modifié dynamiquement (commande ALTER d'AMS). Les data classes de SMS permettent de se dispenser d'un DSCB model. Le fait qu'avec SMS les nouveaux fichiers soient catalogués dès leur allocation (en début de step) plutôt qu'en fin de step amène trois états possibles pour un fichier à génération :

- rolled-in (ou active) : le fichier est catalogué et accessible par la base (numéro relatif) ;
- rolled-off : le fichier étant "trop vieux" sort de l'active set du GDG (la limite du GDG a été atteinte et l'option NOSCRATCH est en vigueur) ;
- deferred roll-in : le fichier est catalogué mais n'est pas associé à sa base (il ne peut être retrouvé que par son numéro absolu GnnnnVpp). Cet état intermédiaire a dû être créé pour tenir compte des allocations avec DISP=(NEW,KEEP) ou DISP=(NEW,PASS). Le re-run des jobs est facilité par le fait que le fichier en deferred roll-in sera réutilisé s'il est désigné sous le forme X(+1). Il pourra prendre l'état définitif rolled-in avec une allocation en DISP=(OLD,CATLG) ou par la commande AMS : ALTER ... ROLLIN.

8. LA GESTION GLOBALE DES RESSOURCES

A) Description de SRM

SRM (system resources manager) est le composant de MVS chargé de gérer les 3 ressources processeur, mémoire et périphériques. Ses objectifs sont les suivants:

* optimiser l'utilisation des ressources. En fonction de son paramétrage, SRM peut considérer comme sous-utilisé un processeur occupé à moins de 100 % de son temps, ou un canal occupé à moins de 30 %; il s'employera à leur fournir du travail et à augmenter ainsi le "débit" du système (point de vue global) .

* affecter les ressources aux tâches en accord avec les objectifs qui lui ont été donnés par l'installation. Par défaut, aucun utilisateur n'est plus favorisé qu'un autre: il revient au site de définir ses préférences et de les signaler à SRM. On influe ainsi directement sur le temps de réponse des utilisateurs du système (point de vue individuel).

A partir de son paramétrage et grâce à de nombreux algorithmes, SRM opère en faisant connaître ses décisions aux autres composants: décisions de swap, de vol de page, d'ajustement de priorité de distribution, choix d'unité à allouer pour les allocations non spécifiques, décision d'empêcher la création d'espace adresse en cas de pénurie de mémoire, etc.

SRM maintient un certain nombre d'indicateurs à l'échelle du système sur lesquels il se fonde pour prendre ses décisions: ces indicateurs sont notamment le taux de pagination, I'UIC maximum, le nombre de cadres de page disponibles (AFQ), le taux d'utilisation du processeur. Les décisions prises ne sont ni infaillibles ni absolues: il s'agit plutôt d'un simple mécanisme d'autorégulation .

Voici un exemple concret de ce mécanisme: à un moment donné, SRM détecte un taux global de fautes de pages trop important: la décision prise est simplement de diminuer de un le nombre d'utilisateurs actifs, et cela jusqu'à ce que le taux de fautes de page soit redevenu supportable. Au contraire, si ce taux redevient faible (toutes choses égales par ailleurs), SRM tentera d'augmenter le nombre d'utilisateurs actifs (swap-in); de la sorte on obtiendra pour les indicateurs globaux un taux acceptable situé le plus souvent dans une même fourchette (appelée le "happy range").

Autre exemple: si l'UIC (qui mesure le vieillissement des cadres de page er mémoire) tombe en deçà d'une certaine valeur (par exemple 20), SRM restreindra les possibilités de vidage logique en jouant sur l'intervalle de temps d'inactivité après lequel le vidage devient physique (think time): il y aura alors plus de chances de voir se produire des vidages physiques et la mémoire s'en trouvera soulagée; I'UIC augmentera alors, et SRM, au-dessus d'une certaine valeur (30), favorisera à nouveau le vidage logique (le "happy range" est ici de 20-30 pour l'UIC dans l'optique du vidage logique).

SRM peut être présenté comme le "chef d'orchestre" ou le conducteur du système, qui, selon l'état des ressources et les options du site prend automatiquement les décisions qui s'imposent et les fait exécuter par les autres composants. Aussi son importance est cruciale pour le bon fonctionnement de MVS. S'il ne peut par lui-même dispenser plus de ressources qu'il n'en existe dans le site, un SRM mal paramétré peut en revanche avoir une influence catastrophique sur les performances.

SRM comporte trois fonctions de base:

* le gestionnaire de charge (workload manager) qui suit les consommations des utilisateurs et les compare entre elles en fonction des objectifs prévus par le site pour chacun d'entre eux. Il reflète le point de vue "individuel" (qui peut être en contradiction avec le point de vue du système);

* le gestionnaire de ressources (resource manager) et ses "load balancers" (équilibreurs de charge) surveillent la consommation des ressources CPU, mémoire et E/S à un niveau global;

* une partie "contrôle", qui fait exécuter les routines de SRM et décide des swaps en fonction des recommandations émises par les deux composants précédents.

Le système communique avec SRM par la macro SYSEVENT pour lui notifier certains événements ou pour lui demander ses décisions. Dans les premières versions, SRM se contentait ainsi de "réagir" aux événements et ne disposait pas d'algorithmes généraux exécutés périodiquement.

B) Notions fondamentales

Nous allons examiner quelques notions indispensables pour comprendre SRM:

* la seconde SRM: il s'agit de la période de base qui règle la fréquence d'intervention des algorithmes de SRM et détermine donc la fréquence des changements de priorité, d'examen de la charge du système, du vol de page, etc. Cette "seconde" est d'autant plus petite que le processeur est rapide: elle correspond par exemple un dixième de seconde pour un 3083JX contre à peu près une seconde pour un 4341;

* l'unité de service: il s'agit d'une unité fictive qui sert à mesurer la consommation des 3 ressources par un utilisateur. La consommation en SU (service unit) est la somme des consommations en temps CPU (TCB et SRB), en entrées-sorties et en mémoire, affectées de coefficients de pondération. La consommation CPU en SU dépend de la puissance du processeur: par exemple 1 seconde CPU sur un 3083JX correspond à 420 SU (en fait 4200 à cause du coefficient qui est habituellement de 10 en mode TCB). Un travail qui a effectué 100 EXCPs (pour transférer 100 blocs) aura consommé 100 SU (500 avec le coefficient de 5). La consommation mémoire, basée sur le nombre de cadres de pages de mémoire réelle utilisés par le consommateur, est très variable d'un moment à un autre: on la multiplie par la consommation CPU pour tenir compte de l'importance de ses variations au cours de l'activité du travail;

* les domaines: il s'agit de groupes réunissant des utilisateurs ou des travaux de profil identique: utilisateurs TSO, travaux batch, régions IMS, espaces-adresses CICS, etc. On peut ainsi avantager certains groupes par rapport à d'autres: on peut surtout imposer pour chaque domaine un nombre minimum d'utilisateurs en mémoire, le minimum MPL (multiprogramming level) et un nombre maximum, le maximum MPL. SRM ne pourra vider les utilisateurs du domaine si leur nombre tombe en deçà du minimum MPL; inversement il ne cherchera pas à activer plus d'utilisateurs que le nombre maximum. Le nombre d'utilisateurs du domaine actifs en mémoire est fixé dynamiquement par SRM à un MPL cible (target MPL) variant entre le minimum et le maximum;

* le taux de service: il s'agit de la consommation en unités de service mesurée sur un intervalle de temps donné. Il permet de comparer, en vue d'un swap, les domaines entre eux, ainsi que les utilisateurs dans un domaine. Pour garder l'équilibre dans un domaine, SRM aura tendance à demander le vidage des gros consommateurs et le retour en mémoire des utilisateurs défavorisés. Cela est pris en compte dans le calcul en tenant compte du temps de résidence récent et du dernier intervalle de temps passé hors mémoire (pour cause de vidage);

* une transaction: SRM ne s'intéresse pas forcément au taux de service d'un espace-adresse mesuré du début jusqu'à la fin, mais uniquement sur une période de temps correspondant à l'unité de travail qu'est une transaction. Une transaction est ainsi définie:

-- le temps de traitement d'une commande entrée depuis un terminal sous TSO. Les commandes groupées en Clist peuvent compter pour autant de transactions, ou au contraire la Clist peut être considérée comme une seule transaction (option CNTCLIST de l'lEAOPTxx);

-- un job batch compte pour une seule transaction de la première étape jusqu'à la dernière (sauf si différents niveaux de performances sont souhaités d'une étape à l'autre).

Le taux de service est évalué sur la durée d'une transaction (les transactions passées n'ont pas d'influence sur ce taux ni sur les décisions de SRM).

* un "groupe" de performance: définit le profil de performance d'une transaction depuis son début jusqu'à sa fin. On indique ainsi des paramètres tels que le domaine de rattachement, la priorité de dispatching, les limites d'isolation mémoire, etc. On peut aussi découper une transaction en plusieurs périodes consécutives associées à des profils différents, les premières périodes étant les plus favorisées;

* les objectifs de performance: il s'agit de déterminer l'importance des utilisateurs dans un domaine et des domaines entre eux en leur accordant différents taux de service. Un objectif est défini par une fonction qui fait correspondre un taux de service à une recommandation de swap (ou "contention index" pour les domaines), analogue à une priorité, qui est le véritable critère de comparaison des consommateurs: plus la recommandation de swap est petite, plus le consommateur devrait être vidé. Les objectifs de performance permettent de comparer entre eux les domaines et, à l'intérieur d'un domaine, chaque consommateur en vue d'un swap; ils ne valent évidemment pas pour les espaces-adresses non swappables.

C) Le paramétrage de SRM

1) GENERALITES

SRM est un système de gestion immédiate des performances de MVS qui est d'une rare complexité: de nombreux paramètres sont disponibles et on court le risque de se perdre dans le détail sans même être capable de mesurer les conséquences sur les performances du changement de tel ou tel paramètre. On risque aussi de mal paramétrer SRM et de dégrader artificiellement les performances; ou d'abuser du paramétrage et de provoquer une déperdition de ressources (overhead) non négligeable. Nous allons examiner les principaux champs d'actions de SRM et voir de quelle façon on peut intervenir et lui faire connaître nos choix.

L'installation peut paramétrer SRM à partir de 3 membres de la PARMLIB:

* IEAIPSxx (IPS, installation performance specification): définit les profils de performance souhaités pour les utilisateurs du système: objectifs de performance, domaines, groupes de performances, etc., en un mot, la "politique" du site;

* IEAICSxx (ICS, installation control specification): établit une relation entre les paramètres de l'IPS et les travaux ou utilisateurs du système. On associe un profil à un travail ou à une classe de travaux;

* IEAOPTxx: définit des paramètres généraux (et fixes) de SRM: il s'agit souvent de valeurs de seuil, que l'on peut neutraliser ou renforcer.

Les paramètres sont pris en compte à l'IPL mais peuvent être modifiés dynamiquement par les commandes MVS: SET IPS, SET ICS ou SET OPT (de même la commande SETDMN permet de modifier les caractéristiques d'un domaine).

2) LA MEMOIRE

Les principaux mécanismes de gestion de la mémoire virtuelle ont été étudiés dans le chapitre 3. Le paramètre capital du point de vue de SRM est le "system high UIC", la plus grande valeur des UIC des pages non récemment référencées, qui mesure la contrainte sur la mémoire. Un UIC très faible indique que les pages "vivent" peu de temps en mémoire: elles sont constamment volées pour fournir des cadres de page libres à d'autres travaux demandeurs. Un très grand UIC (255) indique que la mémoire réelle est abondante et le vol de page très faible. Le high UIC est mesuré par SRM et sert en même temps de critère pour la gestion du MPL (pour SRM beaucoup d'utilisateurs actifs signifie beaucoup de mémoire utilisée), du swap logique, et du vol de page; le mécanisme d'autorégulation apparaît avec le fait que les décisions de SRM, fondées sur le high UIC, font elles-mêmes évoluer ce high UIC.

Le vidage logique est représenté par un paramètre, le "think time" (temps de réflexion) qui est la période de grâce accordée à un espace-adresse pour rester en mémoire avant de subir un vidage physique. Pour les

utilisateurs du logiciel de temps partagé TSO (les plus intéressés par le vidage logique), cette période peut correspondre effectivement au temps de réflexion pris par l'utilisateur avant d'entrer la prochaine commande. Le think time est fonction de la contrainte sur la mémoire, représentée notamment par le high UIC: quand ce dernier tombe en dessous d'un seuil bas (souvent 20), le think time est diminué de 1 seconde; quand l'UIC dépasse un seuil haut (souvent 30), le think time croît d'une demi-seconde. Il reste cependant dans une fourchette donnée (de 0 à 30 secondes par défaut).

L'isolation mémoire permet de diminuer l'impact du vol de page sur un espace-adresse en lui accordant un nombre de pages TWSS (target working set size) compris dans une fourchette minimum/maximum (paramètre PWSS). On peut affiner cette taille mémoire en la rendant dépendante du taux de fautes de pages que subit l'espace-adresse quand il s'exécute (PPGRT) ou quand il est en mémoire (PPGRTR, en XA): si ce taux est supérieur à un seuil, on augmente le TWSS de 7 % (espérant réduire les fautes de pages); s'il est inférieur à un seuil bas, on réduit le TWSS de 3 % (trop de mémoire a été isolée). Les zones communes (CSA, LPA) peuvent aussi être protégées par isolation mémoire (paramètre CWSS); à notre avis il vaut mieux dans ce cas précis laisser fonctionner le jeu naturel de l'algorithme de remplacement de page.

3) LE PROCESSEUR

L'accès à la ressource processeur est déterminé par la priorité de distribution (DPRTY) du demandeur. Cette priorité peut être spécifiée par l'IPS ou par l'utilisateur lui-même (ce qui n'est pas recommandé). SRM contrôle un certain ensemble de priorités, I'APG range (automatic priority group). Les priorités, s'échelonnant de 0 à 255, peuvent être divisées en 16 groupes de 16 (en hexadécimal groupe 0 jusqu'à groupe F). On indique à SRM quels groupes il contrôle (paramètre APGRNG). Dans chaque groupe x (x=0 à F) on peut avoir des priorités:

- fixes: la priorité prend une des valeurs les plus hautes de groupe (entre xB et xF), et ne sera pas modifiée par SRM.

- tournantes (rotate): la priorité vaut xA. Les espaces-adresses auxquels cette priorité aura été affectée seront "déplacés" périodiquement par SRM dans la file du distributeur (le plus haut dans la file passant derrière ses pairs). Il s'agit de donner des chances égales à des utilisateurs d'importance comparable (I'exemple classique étant les régions de messages d'lMS). On peut imposer la fréquence de rotation en fractions de seconde SRM (time units). Depuis MVS-XA 217, la priorité rotate a disparu pour devenir une priorité fixe (voir chapitre 2.C.3)

- MTTW (mean-time-to-wait), tient compte, comme son nom l'indique, du temps moyen d'exécution entre attentes. La priorité varie dans la fourchette x0 - x9. SRM tient compte de l'utilisation du processeur pour affecter la priorité: il donnera une priorité faible (près de x0) aux gros consommateurs et une forte (plus près de x9) aux petits consommateurs (plutôt occupés à faire des E/S). On peut ainsi mélanger des travaux d'importance comparable mais de comportements différents par rapport à la ressource processeur, sans que les uns n'aient un impact défavorable sur les autres.

En pratique, la priorité n'est pas spécifiée dans l'IPS par sa valeur absolue, mais par une notation relative: DP = Fab, DP = Ra, DP = Ma pour les priorités fixes, de rotation ou MTTW respectivement, avec a = 0 à 9 (maximum 10 groupes en APG) et b = 0 à 4 (correspondant aux valeurs xB - xF). Par exemple:

APGRNG = (0-15): contrôle par SRM des priorités 00-9F
PGN = 3, ..., DP = F23: la priorité sera 2E
PGN = 4, ..., DP = R2: la priorité sera 2A
PGN = 5, ..., DP = M1: la priorité variera entre 10 et 19

Après XA 217, DP = R2 devra être remplacé par DP = F2 (priorité fixe 2A).

La gestion des priorités peut encore être affinée (ou compliquée ?) par le "time-slicing" (partage de temps): la priorité d'un espace-adresse peut varier alternativement entre deux valeurs: I'une de base, I'autre de time-slice, et les périodes d'alternance peuvent avoir différentes longueurs. Par exemple, avec le paramétrage suivant dans l'IPS:

PGN=2, ..., DP=M8 : pour travaux batch
PGN=4, ..., DP=F71,TSDP=F90,TSGRP=1 : pour DB2
TSPTRN = (1,1,1,1,*) : time-slice pattern
TUNIT=1 : time-slice de 1s SRM (fréquence de changement des DP)

les travaux batch ont une priorité de base M8 supérieure à celle de DB2 (F71). Cependant, pendant 4 intervalles sur 5 (TSPTRN), soit 80 % du temps, DB2 prendra une priorité F90 supérieure à celle du batch. C'est une politique du: "à chacun sa chance", qui empêche l'un d'étouffer l'autre, mais qui permet tout de même d'indiquer sa préférence (ici DB2 est préféré à 80 %).

4) LES ENTRÉES-SORTIES

Au regard de ce qui a été vu pour les ressources processeur et mémoire, I'intervention de SRM dans les opérations d'E/S sera jugée peu importante, avec raison. Effectivement SRM a moins de latitude pour jouer sur le déroulement d'une E/S, surtout en XA avec le sous-système canal qui choisit les chemins d 'accès.

SRM prend des relevés périodiques de l'activité des canaux (il y a en fait, par regroupement des canaux qui se "partagent" des unités, une notion de "canal logique" en 370 ou de "chemin logique" en XA). Sur cette base il détermine les unités à utiliser à un instant donné pour une allocation non spécifique (sans volser): le canal ou chemin le moins utilisé est préféré, et sur cette voie l'unité qui a le moins d'utilisateurs en allocation (en 370) ou le temps de délai le plus faible (en XA). Ce temps de délai correspond au temps passé dans les files de l'IOS (IOS queue time) et dans le sous-système canal (pending time). Le choix de SRM est à courte vue, il découle d'une situation donnée au moment de l'allocation. Pour les unités à bandes (qui ne peuvent être allouées qu'à un seul utilisateur), on choisit (par défaut) sur la voie la moins chargée l'unité dont l'adresse suit celle de la dernière unité allouée (SELTAPE NEXT).

On peut affecter aux E/S dans les files d'attente de l'IOS des priorités qui épousent celles des espaces-adresses émetteurs (IOQ = PRTY), ainsi les demandeurs les plus prioritaires sont retardés au minimum. L'alternative est de laisser se constituer les files dans l'ordre d'arrivée (IOQ=FIFO).

En XA et pour une configuration multi-processeur, SRM a toute latitude pour "démasquer" vis-à-vis des E/S un processeur supplémentaire dès qu'il estime importante la charge d'E/S (en XA tout processeur peut traiter toute interrup- tion d'E/S). En principe le processeur de base est apte à traiter les E/S, les autres sont "disabled"; SRM estime la charge à partir du nombre d'E/S traitées par l'instruction TPI.

Enfin MVS comptabilise le service d'E/S rendu aux consommateurs à partir du nombre de blocs transférés (1 SU = 1 bloc) appelé parfois "EXCP count". MVS-XA peut utiliser la notion plus précise de "temps de connexion", qui mesure le temps d'utilisation du canal (1 SU = 8, 32 millisecondes de temps de connexion).

5) LE MPL, LE SWAP, LES OBJECTIFS

SRM modifie le nombre d'utilisateurs en mémoire en fonction de la sous-utilisation ou de la sur-utilisation des ressources processeur ou mémoire. L'adaptation est graduelle: le "target MPL" est augmenté ou diminué d'une unité seulement à chaque fois (ou reste stable). Ceci entraîne un swap-in ou un swap-out à la fois, ce qui permet d'ajuster progressivement tant la charge mémoire que la charge processeur.

Les espaces-adresses d'un domaine entrent en compétition entre eux. Le critère de compétition n'est pas le taux de service, un taux de service pouvant correspondre à un "bon service" pour un job batch peut être un très mauvais service pour un utilisateur TSO ou un CICS. Le critère est la recommandation de swap (RS) issue de l'objectif de performance du consommateur.

Un exemple très simple d'objectif de performance pour les espaces-adresses est donné en figure 2. Les utilisateurs associés à l'objectif 1 sont plus favorisés que ceux qui ont pour objectif le 2, de même ces derniers à l'égard de ceux qui sont associés à l'objectif 3. Un utilisateur débute sa transaction avec un taux de service nul,

donc une RS maximale (point A pour l'objectif 1, avec une RS de 150). A mesure qu'il consomme des ressources, son taux de service augmente et sa RS diminue (point B par exemple), et tombe à 100 dès qu'il atteint un taux de 5 000 (point C). Si SRM décide de diminuer le target MPL, ce consommateur peut subir un swap d'échange avec un utilisateur du même domaine qui a été vidé et présente un taux de service plus bas (point B). De même un travail associé à l'objectif 2 (point D) pourra être vidé pour favoriser un travail swappé-out du même domaine (point E). Dans l'exemple de la figure, l'objectif 1 permet d'assigner des RS valant de 100 à 150 (150 est le "cut-off level", obtenu par extrapolation), ce qui permet de ne jamais être comparé (défavorablement) avec les utilisateurs des objectifs 2 et 3.

Pour éviter la trop grande fréquence des swaps d'échange, on impose une consommation minimale avant que le swap puisse intervenir. Il s'agit de l'ISV (interval service value). Ainsi, tant qu'il n'aura pas consommé 100 000 unités de service (ISV par défaut), l'utilisateur avec l'objectif 1 restera à une RS de 150 (point A).

Les courbes (qui sont en fait des droites) d'objectifs de performances sont définies par quelques coordonnées; contrairement à l'usage l'axe vertical est celui des abscisses, le taux de service étant la variable, mesurée par SRM, dont découle la RS. Présenter la RS comme un niveau de charge (workload level) fictif dont on tirerait le taux de service à accorder aux utilisateurs nous paraît ne pas refléter la réalité (SRM n'assure explicitement aucun taux de service, il réprime ou encourage les utilisateurs en fonction de leur RS).

Au niveau du système tout entier, la compétition ne s'exerce pas entre les différents utilisateurs, mais entre les domaines. Ceux-ci possèdent aussi une espèce de RS, le "contention index" (CI) qui peut être établi par une fonction analogue à l'objectif de performance d'un utilisateur individuel (voir figure 3). Le CI d'un domaine peut résulter de la moyenne des taux de service des utilisateurs du domaine (AOBJ, average objective), de la somme (DOBJ) ou être fixé (FWKL). L'AOBJ est assez favorable, puisque le nombre d'utilisateurs du domaine joue peu, contrairement au DOBJ. Le FWKL peut être réservé à un domaine jugé très important ou très peu important.

Dans des conditions favorables, le MPL cible du domaine de plus haut CI est augmenté de 1 s'il comporte des utilisateurs prêts à être dispatchés (ready users); dans des conditions défavorables le MPL du domaine de plus petit CI est diminué de 1. Dans tous les cas le MPL reste dans les bornes assignées au domaine: les "constraints" (paramètre CNSTR), qui permettent de favoriser artificiellement un domaine (DMN 1 de l'exemple) ou de le verrouiller (DMN 9). Dans l'exemple le domaine 1, assez "large", conviendrait à des utilisateurs TSO (sur une centaine d'utilisateurs connectés, au moins 10 seraient actifs ensemble, 30 au maximum); le domaine 9 est celui des utilisateurs "punis", toujours vidés (MPL à zéro).

6) LES "LOAD BALANCERS"

La recommandation de swap évoquée jusqu'à présent est une vue individuelle de l'importance du consommateur. Les load balancers ont une vue plus globale de l'emploi des 3 ressources processeur, mémoire et E/S, avec des notions de sous-utilisation, de sur-utilisation et de consommateur "significatif" de la ressource ; ils fournissent leurs propres recommandations de swap, optionnelles (paramètre RTB). Ils auront tendance à défavoriser les gros consommateurs en période de charge ou au contraire à les favoriser en période creuse.

7) LES CHANGEMENTS SURVENUS AVEC MVS/SP4.2

L'expansion de la mémoire virtuelle amenée par MVS/ESA imposait de remanier un SRM qui n'avait guère changé depuis 15 ans. Avec ESA, la mémoire centrale et la MAP devenant de plus en plus sollicitées, des contrôles plus fins et plus efficaces s'imposaient pour ne pas dégrader la qualité de service. IBM en profite également pour simplifier un paramétrage de SRM un peu complexe. En conséquence les sites qui passent à cette version de MVS doivent modifier leurs paramètres IEAICS, IEAIPS (principalement) et IEAOPT.

Les "load balancers", qui étaient plus ou moins efficaces (seul le "balancer" dédié à la mémoire avait quelque importance), sont éliminés au profit d'un gestionnaire d'espace vital (working-set manager), qui prévient l'écroulement en surveillant les gros consommateurs de mémoire, ceux qui paginent beaucoup et consomment

peu de CPU (ou trop de CPU dite "improductive"). Dans ce but, le gestionnaire a le pouvoir de modifier les recommandations de swap et de déclencher un nouveau mode de pagination, la pagination par bloc (implicit block paging) : les pages de même UIC contiguës en mémoire virtuelle (donc vraisemblablement référencées séquentiellement à un moment donné) seront volées en une seule entrée-sortie. Une application peut aussi déclencher une pagination par bloc de manière explicite, par appel à des routines de service. Le gestionnaire introduit de nouvelles raisons de swap: swap pour améliorer l'utilisation de la mémoire centrale ou le taux de pagination global, et, à l'inverse, swap pour réintroduire en mémoire par échange un espace-adresse pénalisé depuis trop longtemps (30 s pour un TSO, 10 mn pour un batch).

La gestion de la charge (workload management) se trouve simplifiée par suppression des paramètres AOBJ, DOBJ, FWKL, ISV, OBJ, RTB, SRV et WKL. Grâce à de nouveaux paramètres ASRV et DSRV on indique directement la fourchette de taux de service souhaitée pour le domaine, par exemple :

```
DMN=1,CNSTR=(10,30),ASRV=(100,5000)
```

Le CI d'un domaine dépend du taux de service réel : quand ce dernier est dans le "happy range" indiqué par ASRV ou DSRV, le CI prend une valeur entre 1 et 100 ; il est inférieur à 1 quand le taux dépasse la valeur haute, et supérieur à 100 quand le taux est plus petit que la borne inférieure. De façon similaire, les recommandations de swap varient entre 0 et 100 (en croissant si l'utilisateur est "out", en décroissant s'il est actif en mémoire) ; en deçà et au-delà de ces bornes, l'utilisateur est candidat au swap-out et au swap-in respectivement. Le target MPL d'un domaine est remplacé par un couple de 2 valeurs "target" tel que :

minimum MPL < IMPLT < MPL courant < OMPLT < maximum MPL

la valeur in-target (IMPLT) correspond au nombre d'espaces-adresses "in" quand la pagination ou la consommation CPU sont trop fortes, la valeur out (OMPLT) correspondant au contraire à une absence de contention. SRM ordonne des swaps unilatéraux pour garder le MPL courant dans la fourchette (IMPLT, OMPLT). Le site reste maître des valeurs minimum MPL et maximum MPL.

La marge de contrôle de l'emploi de la mémoire centrale et de la MAP est augmentée ; en particulier on peut, avec les paramètres ESCTxxx(n) de l'IEAOPT et ESCRTABX de l'IEAIPS, doser au niveau d'un domaine l'emploi de la MAP pour la pagination ou le swap.

D) La gestion des performances

1) GÉNÉRALITÉS

Dans un environnement grand système MVS il est nécessaire de suivre les performances du système, de détecter les éventuels goulets d'étranglement, d'améliorer le temps de réponse et le débit global (tuning) quand le besoin s'en fait sentir; on peut aussi être amené à extrapoler à partir de l'état actuel pour évaluer les besoins informatiques à moyen ou long terme (capacity planning). Nous nous en tiendrons à quelques généralités sur le "tuning" de MVS, non parce que la gestion des performances serait un domaine secret réservé aux spécialistes, mais plutôt parce que le champ d'action est vaste et les soucis, les outils et les manières de procéder assez nombreux.

La démarche de tuning (réglage du système) consiste à mesurer divers paramètres-clés, à les analyser et à modifier dans le système ce qui doit l'être pour améliorer les performances. L'évaluation de ces paramètres peut découler de certaines règles empiriques ("ROT", rules of thumb) en vigueur dans tous les sites MVS, et qui fournissent des points de repère intéressants pour une première approche.

La mesure d'un grand nombre de données nécessite un outil d'enregistrement; en MVS, il s'agit de SMF (system management facilities), le grand "collecteur" de toutes les données en provenance du système ou des utilisateurs. En XA, SMF est un espace-adresse autonome. Les fichiers SMF (SYS1.MANx) recueillent notamment toutes les statistiques de consommation des ressources, des éditions sur imprimantes, des changements d'état des unités d'E/S ou des fichiers, et tous les événements notables; ils constituent la base de

référence pour la facturation, le suivi des performances et de toute l'activité du site. Les enregistrements SMF sont organisés en "types" bien spécialisés, dont un sous-ensemble est réservé au système; le paramétrage de la collecte est défini dans le membre SMFPRMxx de la PARMLIB. Les fichiers SMF (VSAM) peuvent être déchargés sur des fichiers séquentiels (programme IFASMFDP ou procédure SMFDUMP) pour être traités par les programmes de statistiques adéquats (SLR d'lBM par
exemple).

RMF (resource measurement facility) est un outil de mesure très utile pour l'analyse des performances. Il relève l'utilisation des ressources, les contentions, l'évolution des divers indicateurs de SRM, etc., en procédant par échantillonnage sur des intervalles de temps donnés (cycles). Les résultats sont presentés dans des sessions interactives (monitors II ou III), dans des comptes-rendus, ou enregistrés dans SMF pour être traités ensuite par un RMF "post-processor".

Nous allons passer en revue les différentes ressources du système, par ordre d'importance, et indiquer dans les grandes lignes la façon de les gérer dans le sens d'une plus grande performance.

2) LE PROCESSEUR

Le taux d'utilisation du processeur mesure sa charge: il est évalué en mesurant le temps d'activité sur une période donnée. Au-delà de 80 à 90 % d'occupation, il faut sérieusement envisager une évolution vers une machine plus puissante. SRM utilise le taux d'utilisation des processeurs pour la régulation du MPL; du point de vue de SRM ce taux varie entre 0 % et 101 %, ou depuis MVS/XA 220 entre 0 % et 128 % (128 % d'occupation correspondant à une occupation à 100 % avec au moins 28 utilisateurs qui n'ont pu être dispatchés sur la période de mesure).

On doit étudier aussi la manière dont la consommation se répartit entre les différents protagonistes, batch, TSO, télétraitement, etc. (grâce par exemple aux enregistrements SMF de type 30 ou SMF/RMF de type 72). Ce qui n'a pu être attribué constitue la déperdition (overhead) de MVS, imputable à l'activité du processeur qui n'a pu être mesurée. Cette déperdition peut varier de quelques % jusqu'à 25 % de la consommation globale. La notion de "capture ratio" permet en capacity planning de comparer la consommation mesurée avec la consommation réelle; les ROTs donnent 80 % pour un utilisateur TSO, 90% pour un job batch.

La puissance d'un ordinateur est évaluée couramment en "Mips" (millions d'instructions par seconde). Il ne faut pas trop prendre au sérieux cette mesure qui n'a souvent qu'une valeur commerciale et ne fait pas référence chez les constructeurs; si elle ne signifie rien en valeur absolue, en revanche elle permet aisément de comparer entre elles les puissances de plusieurs ordinateurs.

Les multiprocesseurs améliorent le débit du système plus qu'ils n'apportent de la puissance supplémentaire. En effet, la grande majorité des espaces-adresses s'exécutent en monotâche, donc sur un seul processeur à la fois: dans le meilleur des cas, n processus peuvent occuper simultanément n processeurs et utiliser (au maximum) un nième de la puissance totale (puissance qui est d'ailleurs loin d'être équivalente à n fois la puissance du monoprocesseur de référence, à cause de la déperdition propre aux multiprocesseurs).

Les priorités vis-à-vis du distributeur devraient respecter soigneusement une hiérarchie fondée sur le principe: les espaces-adresses serveurs d'abord. Les priorités n'ont pas d'importance en valeur absolue mais en valeur relative, et l'ordre de classement n'est jamais indifférent. Par exemple, pour un site doté de CICS, DB2 et TSO, on pourrait adopter l'ordre suivant, par priorités décroissantes:

- espaces-adresses MVS (PCAUTH, TRACE, ALLOCAS, DUMPSRV)
- tâches "privilégiées" (inits)
- JES
- VTAM
- DB2 IRLM
- DB2 MASTER
- DB2 database manager
- CICS (production)

- TSO première période
- toutes les autres "started tasks"
- TSO deuxième période
- CICS de test
- TSO troisième période
- batch première période
- batch seconde période

En "queue de peloton", TSO 3e période se rapproche du batch. Les travaux batch doivent être dans un domaine peu performant et leur MPL rester dans une fourchette assez restreinte pour permettre à SRM de jouer le gendarme dans ses décisions de vidage sans défavoriser TSO ni les serveurs. Le MTTW est tout à fait préconisé pour le batch et TSO 3e période, alors que les espaces-adresses importants doivent avoir une priorité fixe assez élevée. TSO "première période" correspond à une consommation faible (quelques centaines d'unités de service) et peut être favorisé autant que l'on veut.

3) LA MÉMOIRE RÉELLE

Dans un système à pagination tel que MVS, les paramètres qui témoignent de la faible ou forte utilisation de la mémoire sont le "high UIC", l'AFQ, le taux de pagination.

L'UIC mesure le vieillissement des cadres de page en mémoire: une valeur de 255 indique qu'il n'y a aucune contrainte, tandis qu'une valeur entre 0 et 2 indique une très forte utilisation (en-dessous de 2 SRM commence à diminuer le MPL).

L'AFQ indique la taille de la réserve de cadres disponibles dans laquelle puise MVS pour satisfaire les utilisateurs, et qui est alimentée par les swaps ou vols de pages dès qu'elle tombe en-dessous d'un seuil bas (14).

Le taux de pagination correspond selon les cas à un taux de fautes de page (page-in ou reclaim), un taux de demande de page (page-in et out) ou à une pagination globale si l'on tient compte en plus du swap et du VIO. SRM suit particulièrement le taux de demande de page et veille à ce qu'il reste dans une fourchette fixe dépendant du type d'ordinateur (plusieurs centaines de pages par seconde). On peut obliger SRM à contrôler aussi le taux de fautes de pages (paramètres RCCPTRT de l'IEAOPT), qui est un important facteur de dégradation des performances s'il est trop élevé.

Le vidage logique et l'isolation mémoire permettent d'exploiter plus rationnellement la mémoire si on n'en abuse pas. La fourchette de "think time" pour le vidage logique est par défaut de 0 à 30 secondes (paramètre LSCTMTE); une fourchette plus "resserrée" de 3 à 20 secondes peut se révéler plus adéquate. L'isolation mémoire doit pouvoir varier dynamiquement en fonction du taux de fautes de pages que subit l'utilisateur à "isoler" (paramètre PPGRTR de l'IPS); la fourchette pour CICS ou IMS tourne autour de 5 fautes de pages maximum.

L'évaluation des besoins en mémoire centrale doit tenir compte des espaces-adresses non swappables (JES, VTAM, CICS, IMS, DB2, les espaces de service MVS), des swappables (TSO, batch), des mémoires non paginables (SQA, noyau, HSA) et paginables (LPA, CSA). On peut aussi tenir compte de l'AFQ et de la mémoire occupée par le vidage logique. MVS, souvent réputé pour ses besoins en mémoire, sait aussi s'adapter aux situations de pénurie; dans de tels cas, se pose avec plus d'acuité le choix de l'utilisateur à privilégier.

4) LA MÉMOIRE VIRTUELLE

Le suivi de la mémoire virtuelle est une affaire d'organisation plutôt que de recherche de performance. Les frontières des zones LPA, CSA, private, sont fixes et ne peuvent être modifiées que par un IPL, et le problème est crucial pour les zones en dessous de la barre des 16M: toutes choses égales par ailleurs, l'augmentation de la taille de la LPA entraîne une diminution de la taille de la zone privée, aggravée en XA par la nécessité

d'avoir une frontière private/CSA arrondie au Méga-octet. Il faut donc n'installer qu'avec discernement des programmes en LPA en-dessous de la "barre", suivre l'occupation de la CSA et sa fragmentation possible, veiller à ce qu'il y ait une taille suffisante de private "below" (la private au-dessus de la barre atteint en XA 2000 Méga-octets, et par défaut la REGION du JCL est de 32M).

Pour la zone privée elle-même, se pose le problème de la "collision" entre les données utilisateurs allouées depuis le bas de la private et les zones systèmes (LSQA, SWA, AUK) allouées depuis le haut. Les jobs batch y sont plus exposés que les "started tasks", car, tant qu'un initiateur batch est actif, sa LSQA n'est pas rafraîchie et subit une fragmentation qui peut gêner les travaux qui s'exécutent avec cet INIT. Une autre possibilité intéressante est de déplacer au-dessus de la barre certains blocs de contrôle de la SWA (possible à partir de MVS/XA 220).

5) LA MÉMOIRE AUXILIAIRE

Les processus physiques de pagination et de swap en MVS sont performants, grâce notamment au suspend/resume (voir chapitre 4.E.2) et à la priorité donnée aux interruptions d'E/S pour ASM sur toute autre interruption. On évitera certains ennuis en suivant ces règles simples:

* donner une taille suffisante aux fichiers de pagination: quelques cylindres de 3380 pour les fichiers PLPA et COMMON, beaucoup plus pour les fichiers locaux (entre 100 et 300 cylindres en pratique);

* isoler ces fichiers (en les créant évidemment sur des disques dédiés différents, connectés si possible à des canaux différents);

* éviter l'emploi des fichiers swap, qui n'ont d'autre intérêt que de séparer les opérations de swap (TSO, batch) de celles de pagination, ce qui ne profite qu'aux espaces-adresses non swappables.

Le suivi des performances de pagination et de swap s'effectue en mesurant le temps moyen pour transférer une page ou un swap-set de 12 slots. Le MSPP (millisecondes par page) ou le "page delay time", temps de mise à disposition d'une page par ASM, est utilisé par SRM comme paramètre de tuning en même temps que la longueur moyenne de la file d'attente d'ASM (ASM queue length).

La présence de la mémoire d'arrière-plan, qui opère comme un cache de la mémoire auxiliaire géré par RSM, améliore considérablement les performances de pagination et de swap. On dispose cependant de peu de moyens pour contrôler son emploi avant SP4.2. Quand la MAP est pleine, le processus de migration ramène toutes les pages en mémoire centrale pour que le sous-système de canaux puisse les envoyer en mémoire auxiliaire; la mémoire réelle peut se trouver momentanément chargée. Le "migration age", analogue au "system think time", détermine le délai après lequel l'espace vital d'un espace-adresse TSO quitte la MAP pour aller en mémoire auxiliaire. De grandes valeurs indiquent que la MAP n'est pas chargée. En dessous de 100, SRM tend à éviter l'emploi de la MAP pour de nouvelles pages.

6) LES E/S

Il s'agira pour nous essentiellement des E/S disques qui sont les plus complexes et les plus importantes du point de vue des performances en MVS. Beaucoup de recommandations ont déjà été données dans les chapitres précédents: emploi du VIO pour les fichiers temporaires, suppressions des STEPLlBs quand c'est possible, etc. MVS/ESA et DFSMS apportent aussi des nouveautés considérables dans ce domaine.

La taille du bloc physique des fichiers (BLKSIZE) est un facteur trop négligé et qui dans l'idéal devrait résulter d'un compromis: trop petite, elle donne lieu à de trop nombreuses E/S et à un mauvais emploi de l'espace disque; trop grande, l'occupation canal (pour le transfert) augmente ainsi que la taille des tampons de données (qui sont fixés en mémoire centrale lors des E/S). Si l'on s'en tient à l'optimisation de l'espace disque sans souci

de performance, une grande taille de bloc permet de réduire le nombre de "gaps" inter-blocs: 23476 serait la taille idéale pour les disques 3380 (98 % de remplissage) et 27998 pour les 3390. Pour les fichiers de LRECL 80, très communs, on peut s'en tenir à des BLKSlZEs plus modestes: par exemple 9040 (95 % de remplissage). IBM utilise souvent des tailles de 3120 (85 % de remplissage) ou 6160 (90 %) pour l'installation de ses propres logiciels. Le SDB est la meilleure solution au problème du choix du blocksize (voir chapitre 5.E).

Le temps de réponse disque peut être grossièrement décomposé ainsi:

TEMPS	IOSQ time : attente dans l'IOS (MVS) --> SSCH	■ file des UCBs ■ priorités
D'ATTENTE	PENDING time : attente dans le sous-système de canaux--> lancement E/S vers le device	■ file des LCUs ■ canal, contrôleur ou disque occupés ■ disque réservé
TEMPS DE	DISCONNECT time (pas de transfert) secteur)	■ SEEK (mouvement du bras entre cylindres) ■ SET SECTOR (positionnement ■ latency (positionnement du record sous la tête de lecture)
SERVICE	CONNECT time	■ SEARCH (recherche du record) ■ transfert des données

Pour un disque 3380, le temps de réponse moyen d'une E/S pourrait être celui-ci :

IOS queue time : 5 ms
pending time : 1 ms
disconnect time : 19 ms
connect time : 5 ms
TOTAL : 30 ms

Il ne s'agit que d'une exemple donné à titre indicatif (avec les caches on peut ramener le temps de réponse à quelques millisecondes, en diminuant fortement le temps de service). Les contrôleurs de disques permettent des transferts sur les canaux à une vitesse de 3 millions d'octets par seconde (4,5 pour les derniers modèles, et on peut dépasser ce chiffre avec ESCON). Quand aux canaux auxquels des contrôleurs de disques 3380 sont connectés, on estime qu'ils peuvent être occupés jusqu'à 50 ou 60% de leur temps sans dégradation.

7) EN CONCLUSION

Il nous faut redire en conclusion que la séparation en ressources processeur, mémoire et E/S que nous avons faite jusqu'ici a un intérêt didactique certain (elle est aussi adoptée par SRM), mais elle peut se révéler insuffisante dans le domaine du suivi des performances, où une vision plus globale est nécessaire. Résoudre un problème de performance bien précis peut reporter ailleurs le goulet d'étranglement et provoquer l'apparition d'un autre problème. Ainsi, dans certains cas, installer un ordinateur plus puissant, sans se préoccuper du reste de la configuration, pourra faire ressentir plus cruellement l'inadéquation des unités d'E/S, dont les temps de réponse, devenus disproportionnés par rapport à la vitesse des processeurs, freineront toute l'activité du système. Dans une telle situation, augmentera aussi le besoin en mémoire: en effet, quand les E/S sont peu performantes, les travaux occupent plus longtemps la mémoire (un travail en attente du résultat d'une E/S ou subissant une faute de page n'est pas swappé pour autant, il perd seulement l'usage du processeur). De même, un site pauvre en mémoire centrale et en MAP subira une augmentation de la consommation CPU à cause de la pagination supplémentaire et de la migration de la MAP vers les fichiers page ou swap. Dans un site équilibré, la configuration (processeur/taille de mémoire réelle / types d'unités disque) a une certaine cohérence, qui correspond aux besoins de l'entreprise (et à ses moyens financiers).

Autres exemples de l'imbrication des ressources informatiques, cette fois dans l'appréhension de l'importance des processus, et qui montrent combien la tâche d'optimisation d'un système MVS est délicate: rendre non-

swappable un CICS (ou un IMS) et lui donner une forte priorité de distribution ne suffit pas à lui éviter un fort taux de pagination, car il ne pourra référencer assez souvent les pages de son espace vital (énorme) pour leur éviter d'être volées (hormis avec l'isolation mémoire). Inversement, donner une faible priorité à un travail lui fera aussi subir plus fortement le vol de page: dispatché moins souvent, il aura moins l'occasion de référencer ses zones mémoires. Enfin, donner une forte priorité de distribution à un travail très demandeur en E/S est souvent considéré comme une bonne politique..., sauf lorsque les E/S sont ciblées sur un seul disque: dans ce cas, les travaux moins prioritaires auront beaucoup de mal à accéder au même disque (IOQ=PRTY dans l'IPS transmet la priorité de distribution aux E/S).

Notons finalement que l'optimisation des performances d'un CICS, d'un IMS ou d'un DB2, sort très vite du cadre de MVS et de SRM. Ces logiciels ont leur propre distributeur, leur propre façon de gérer les E/S, et l'étude des files d'attente internes (internal queuing, software queuing) réclame d'autres compétences.

9. LE SOUS SYSTEME DE GESTION DES TRAVAUX

A) Notion de sous-système

Le terme de sous-système s'emploie en informatique dans le sens le plus vague pour désigner un composant offrant un certain nombre de fonctions vis-à-vis de certains utilisateurs. Ainsi, pour certains, TSO ou VTAM sont des sous-systèmes au même titre que DB2, IMS ou JES (ce qui n'est pas le cas dans un sens strict). Le terme s'utilise dans un sens plus précis en MVS. Un sous-système est une extension du système; il peut être représenté par un espace-adresse (mais ce n'est obligatoire que s'il doit pouvoir agir de sa propre initiative); il est désigné par un nom d'au plus 4 caractères reconnu par MVS; on peut communiquer avec lui grâce à un interface sous-système, le SSI (subsystem interface).

Le nom du sous-système peut être déclaré dans le membre IEFSSNxx de la PARMLIB, dans le module IEFJSSNT ou lors de la génération de MVS. Un sous-système est mis en place soit par une commande START soit par l'appel d'une routine particulière au démarrage de MVS. Les sous-systèmes sont représentés par les blocs de contrôle SSCVT (subsystem communication vector table) qui décrit le sous-système et SSVT (subsystem vector tab!e), construite par le sous-système pour décrire les fonctions qu'il offre.

Les routines du SSI sont en zone commune; elles peuvent être invoquées directement par une macro IEFSSREQ. Les blocs suivants sont impliqués dans l'appel:

- un SSOB (subsystem options block): c'est une liste de paramètres pour le SSI (avec notamment un code précisant la fonction demandée au SSI);
- un SSIB (subsystem identification block), facultatif: il indique le sous-système auquel on s'adresse (par défaut le sous-système "primaire", JES).

Tout sous-système établit au cours de son démarrage la liste des services qu'il peut rendre. Ces services de SSI sont plus aisément accessibles que ceux qu'offre un espace-adresse par le cross-memory, plus complexe et d'invention plus récente. De nombreux logiciels fonctionnent en sous-systèmes (DB2, IMS, NETVIEW, etc.). Consulter l'annexe pour un exemple d'utilisation du SSI.

A noter aussi la notion de fichier sous-système. On peut par le paramètre SUBSYS de la carte DD associer ainsi à un fichier une méthode d'accès non standard offerte par un sous-système. On peut aussi accéder par allocation dynamique à un groupe de données qui n'est pas un fichier au sens habituel mais qui est considéré comme un fichier par le sous-système (il s'agit par exemple pour JES des SYSOUTs ou SYSlNs des travaux).

Au démarrage de MVS est installé un sous-système MSTR (master subsystem) qui est une espèce de JES rudimentaire disponible en permanence ; il sert à initialiser le master scheduler, premier espace-adresse de MVS (voir chapitre 10); il informe les autres sous-systèmes d'événements pouvant les concerner (apparition d'un message, envoi d'une commande MVS, fin d'une tâche, etc.); il peut enfin être utilisé pour lancer des tâches indépendamment de JES (et notamment pour initialiser JES lui-même). Les espaces-adresses régis par le MSTR se voient cependant imposer les contraintes suivantes: nom à 4 caractères reconnu comme sous-système, lancement d'un JCL mono-étape par commande START d'une procédure en SYS1.PROCLIB, recherche des fichiers catalogués à allouer uniquement dans le MASTERCAT, paramètre TIME obligatoire (TIME=1440 permet de se débarrasser du problème), pas d'accès au SPOOL (donc pas de fichiers SYSOUT). L'intérêt peut être de disposer de procédures de reprise quand JES est défaillant. Cependant un tel sous-système est insuffisant dans la marche normale de l'exploitation informatique.

Un sous-système, appelé sous-système primaire, est nécessaire pour traiter les travaux, c'est-à-dire les faire admettre dans le système et les recueillir, ainsi que les impressions qu'ils ont créées, après leur achèvement. Il s'agit d'un superviseur ou planificateur (scheduler) de travaux. En MVS, un tel sous-système est JES (job entry subsystem) qui existe sous deux variantes: JES2 et JES3. Le sous-système primaire est déclaré à MVS à la génération ou, depuis MVS SP220, dans l'lEFSSNxx. Au cours du traitement de chaque travail, MVS

communique constamment avec ce sous-système par le SSI. En MVS, JES est le planificateur de travaux de plus haut niveau (SRM et le distributeur pouvant être considérés comme des planificateurs de niveau inférieur).

B) Le traitement des travaux par JES

JES traite les travaux à leur entrée dans le système et à leur sortie (fin d'exécution, impressions). Le SPOOL est constitué d'un ou plusieurs fichiers sur différents volumes; il est utilisé pour stocker les travaux, leurs paramètres (SYSlNs) et leurs fichiers d'impressions (SYSOUTs). On peut distinguer plusieurs étapes de traitement:

* la réception (job entry): un travail est soumis à JES depuis un périphérique (disque le plus souvent) ou d'une station de travail éloignée (remote) reliée par des lignes de télécommunication (remote job entry). Le travail et ses données sont copiés sur le SPOOL; le travail reçoit un numéro (job id). Un travail est le plus souvent soumis par la commande SUBMIT de TSO ou par copie dans un lecteur interne (internal reader), qui est un fichier SYSOUT créé sur le SPOOL et repris par l'"input service" de JES pour traitement;

* la conversion: le JCL du travail est analysé et les procédures appelées sont éventuellement incorporées, puis il est converti par le "converteur" de MVS en texte interne (s'il ne comporte pas d'erreur). Ce texte interne, seul compris de MVS, est stocké sur le SPOOL;

* l'interprétation: le texte interne est utilisé pour construire en mémoire divers blocs de contrôle. JES2 n'effectue cette opération que lorsque le travail sort de la file d'attente pour être exécuté par MVS (job select); JES3 effectue l'interprétation à l'avance et stocke les blocs sur le SPOOL;

* l'allocation des fichiers: avant de démarrer une étape, l'initiateur alloue tous les fichiers de l'étape et crée en SWA les blocs correspondants: les fichiers nouveaux sont créés, les volumes requis sont montés. En fin d'étape, les fichiers sont désalloués. JES3 a une fonction d'allocation qui lui est propre, MDS, par laquelle il peut contrôler tout ou partie des volumes et allouer à l'avance tous les fichiers pour toutes les étapes du travail; les blocs de SWA sont créés à l'avance, stockés sur le SPOOL, passés à l'initiateur qui n'a plus qu'à allouer les volumes non contrôlés par JES3;

* la planification (scheduling): un initiateur est un espace-adresse destiné à l'exécution d'un travail batch. L'installation (ou le pupitreur) peut contrôler le nombre d'initiateurs, en démarrer ou en arrêter plusieurs de manière à décider du nombre maximum de travaux batch et des classes de travaux à traiter. Les initiateurs sont rattachés à un certain nombre de classes de travaux qu'ils traitent exclusivement. JES passe à un initiateur le travail de plus haute priorité dans une classe donnée (priorité au sens JES, et non MVS); en fait JES répond à l'initiateur qui lui a requis un travail par l'intermédiaire du SSI. L'initiateur crée en SWA les blocs d'allocation et lance le programme de la carte EXEC;

* l'exécution est à la charge de MVS. Les accès au SPOOL sont faits via le SSI. En fin d'étape, la fonction MVS de terminaison provoque la désallocation des unités allouées et informe JES2 par le SSI;

* la sortie (output): les messages et impressions créés par le travail sont recueillis une fois celui-ci terminé pour être éventuellement dirigés sur les imprimantes du site en fonction de leurs classes et de leurs priorités de sortie. Il est possible d'éditer des SYSOUTs avant la fin du travail par désallocation dynamique (éditions "spin-off");

* la purge: le travail et toutes ses données sont purgées du SPOOL, soit que le pupitreur ou le propriétaire du travail en ait décidé ainsi (par exemple par la commande CANCEL PURGE de TSO ou par une commande JES qui nettoie le SPOOL en fonction de l'âge des SYSOUTs), soit que ce travail soit purgé automatiquement (exit JES, purge "conditionnelle" pour les travaux terminés sans erreur).

C) JES2 et JES3

IBM propose deux sous-systèmes de gestion des travaux: JES2 et JES3, qui proviennent respectivement de deux systèmes de spool HASP (Houston automatic spooling priority system) et ASP (asymetric multiprocessing system) qui étaient utilisés sur les systèmes MVT et SVS prédécesseurs de MVS. JES3, moins employé, est plus difficile à mettre en oeuvre que JES2, est plus consommateur de ressources, mais il offre un plus grand contrôle des opérations. Nous allons voir en quoi diffèrent exactement les conceptions de JES2 et de JES3.

* JES3 exerce une gestion centralisée des travaux: un processeur global gère jusqu'à 8 systèmes MVS pour former un "complexe" JES3. Le "JES3 global", connecté aux autres "JES3 locaux" en couplage lâche (LCMP) par liaison canal-à-canal (CTC adapter), reçoit les travaux, gère le SPOOL et les initiateurs et finalement distribue les travaux aux systèmes MVS. Au contraire, avec JES2, chaque système (7 au maximum) accédant à un SPOOL partagé fonctionne de manière indépendante, autant pour la sélection des travaux que pour les impressions. Les files d'attente sont communes, et un travail peut être reçu par un JES2 et être exécuté sur un autre;

* JES3 offre une vision centralisée du complexe: une seule console dédiée à JES3 suffit pour suivre les opérations sur tous les processeurs. JES2 communique avec le pupitreur par la console du système MVS sur lequel il s'exécute;

* de même JES3 permet, de façon optionnelle, une centralisation des allocations de volume et de fichier, grâce à son composant MDS (main device scheduling). Avant de passer un travail à un initiateur, JES3 peut tenter d'allouer fichiers et volumes en fonction des tables dont il dispose: table des volumes montés, des volumes résidents, qui permet de diriger le travail sur le processeur du complexe qui les détient, table des volumes indisponibles, table de tous les fichiers alloués par les travaux (par JCL), qui fournit une intégrité des fichiers au niveau du complexe qui n'existe pas avec JES2: un travail ne démarre pas s'il demande un volume indisponible ou un fichier pris par ailleurs. Par JES3 on peut allouer tous les fichiers du travail pour toutes les étapes avant que le travail ne démarre. Avec JES2, c'est l'initiateur qui se charge des allocations avant chaque étape;

* la planification des travaux est fondée en MVS sur la notion de classe d'exécution. Avec JES3, à chaque classe peuvent être associés des travaux s'exécutant sur certains processeurs, et éventuellement un profil d'exécution: travail consommateur en entrées-sorties, ou plutôt en processeur, ou profil "médian" ; JES3 peut ainsi équilibrer la charge de travail. Les classes sont réunies en groupes d'exécution, disposant d'initiateurs et d'unités pour l'allocation desquelles ils sont prioritaires. Les travaux d'un même groupe entrent en compétition pour l'usage des initiateurs. La souplesse de JES3 dans la définition des classes va jusqu'à la possibilité d'établir des relations entre elles et de pouvoir empêcher l'exécution dans une certaine classe si trop de travaux s'exécutent dans une autre. A chaque classe correspond un nombre maximum de travaux sur chaque processeur et au niveau du complexe JES3.

Avec JES2 la notion de classe est bien plus simple. Chaque processeur dispose d'un nombre fixe d'initiateurs traitant une ou plusieurs classes, avec un ordre de préférence. JES2 essaie de fournir à chaque initiateur le travail de plus haute priorité dans la classe préférée par l'initiateur; chaque processeur sur lequel s'exécute JES2 prend ainsi, dans les files d'attente communes, des travaux dans les classes que ses initiateurs traitent, et ceci indépendamment des autres processeurs du complexe JES2. En jouant sur les relations initiateurs/classes, on peut favoriser certaines classes de travaux en les mettant en tête dans les préférences d'un grand nombre d'initiateurs.

* JES2 et JES3 offrent différentes fonctionnalités:

-- le DJC (dependent job control) est une intéressante fonction de JES3 qui permet de soumettre un groupe de travaux (réseau) liés par des relations de dépendance, qui imposent un ordre de passage. Il n'y a pas d'équivalent en JES2 (un produit spécifique existe, le CJS);

-- la fonction "execution batch scheduling" de JES2 permet de traiter en un seul flot d'entrée des travaux d'une classe donnée et de mêmes caractéristiques d'allocation et d'exécution. Un même programme d'exécution batch (XBATCH) traite les travaux comme autant de fichiers paramètres différents. On évite ainsi d'initialiser n fois n travaux de même profil (cette fonction est performante mais assez limitée, aussi elle est peu utilisée);

-- la fonction "deadline scheduling" de JES3 permet d'augmenter la priorité d'un travail s'il n'est pas sélectionné pour exécution à une certaine heure (ce qui n'assure pas forcément son exécution, mais en augmente la probabilité...). JES2 et JES3 disposent aussi d'un "priority aging" qui permet d'augmenter automatiquement la priorité d'un travail dans une classe donnée s'il ne démarre pas, cela en fonction du temps d'attente en JES2 et du nombre de tentatives de scheduling en JES3;

-- JES3 dispose de nombreux DSPs (dynamic support processes), qui sont autant d'utilitaires très commodes: ils permettent de faire des copies carte à carte (CC), bande à bande (TT), carte ou bande vers imprimante (CP, TP), carte à bande (CT), de soumettre des travaux lus par des "readers" carte, bande ou disque (CR, TR, DR), d'écrire des labels sur des bandes (TL), de sauver des travaux sur bandes (DJ, dump job), etc.;

-- JES2 permet de mettre en place des commandes MVS ou JES qui seront exécutées à une heure fixe de la journée (voir paragraphe E).

JES2 et JES3 disposent tous deux d'une fonction NJE (network job entry) qui leur permet de participer à un réseau d'ordinateurs connectés par lignes de communication ou CTCs; ils peuvent ainsi transférer des données ou des travaux avec d'autres systèmes JES2, JES3 ou même VM (avec RSCS) ou DOS/VSE (avec VSE/POWER). Il n'y a pas dans ces réseaux de "maître" ni d'"esclave" ; chaque système fonctionne indépendamment des autres mais assure une fonction de routage d'un point (node) à un autre: un travail peut ainsi être soumis depuis un site, exécuté sur un autre, et produire des états sur un troisième. JES2 supporte les protocoles de communication BSC et SNA, tandis que JES3 ne supporte que le BSC.

D) Le SPOOL

Le SPOOL est le fichier ou le groupe de fichiers sur support en accès direct (disque) où sont stockés les travaux, leurs données ou paramètres (SYSIN), leurs sorties ou impressions (SYSOUT). La destination principale du SPOOL est, comme toujours dans les systèmes modernes, d'augmenter le parallélisme des opérations en désynchronisant l'exécution d'un travail avec l'impression des états qu'il crée: on stocke de facon temporaire ces états pour les imprimer à la demande, quand les imprimantes seront disponibles. Le SPOOL doit être suffisamment grand (souvent plusieurs milliards d'octets) pour contenir toutes les SYSOUTs que peut créer le site; un SPOOL qui devient plein à 100 % conduit à un état critique: plus aucun travail (ou utilisateur TSO) ne peut démarrer, puisqu'il n'y a même plus assez de place pour copier son JCL...

Le SPOOL sert aussi à stocker la SYSLOG (system log) qui contient tous les messages émis par les travaux ou par MVS, qu'ils apparaissent ou non sur les consoles.

Les états en sortie peuvent aussi être récupérés par un programme spécifique pour être retraités, copiés sur un autre support, imprimés en décentralisé, etc. MVS fournit un utilitaire standard, l'external writer (XWTR), qui permet de copier sur disque les SYSOUTs d'une classe donnée.

Le moyen de communication privilégié avec le SPOOL est le SSI, utilisé tant par l'external writer que par TSO. En JES2 le produit SDSF rend de nombreux services, et est bien plus commode que les commandes CANCEL, OUTPUT de TSO.

La gestion du SPOOL est facilitée par divers services:

* une fonction de déchargement permet de le soulager des grosses SYSOUTs: il s'agit de l'offload de JES2 ou du DJ (dump job) de JES3;

* le SPOOL peut être purgé des SYSOUTs les plus anciennes par des commandes de purge (en JES2) ou le DSP JSM (JES3 spool maintenance) de JES3. La classe des SYSOUTs et leur âge sont les critères du nettoiement;

* avec JES2 des fichiers SPOOL peuvent être ajoutés, formatés ou retirés dynamiquement; JES3, plus rigide, n'offre qu'une fonction de démarrage en WARM REPLACE pour modifier la configuration SPOOL et quelques

commandes pour empêcher ou permettre l'emploi d'un fichier SPOOL. En revanche, il permet aisément de créer des partitions de SPOOL pour réduire les délais d'accès disque; on peut affecter un ensemble de fichiers SPOOL à un processeur ou à des classes de travaux.

Le SPOOL peut être partagé par plusieurs systèmes. En JES3 ce partage ne pose pas de problème puisque la gestion est centralisée par le processeur JES3 global qui communique par CTC avec les JES3 locaux. En JES2 on utilise un petit fichier, le CHECKPOINT, accédé régulièrement par chaque membre du complexe MAS (multi-access spool) pour communiquer avec les autres et connaître la file d'attente des travaux; le RESERVE fournit la sérialisation nécessaire.

Le fichier CHECKPOINT employé tant en JES2 qu'en JES3 est destiné à l'origine à accélérer le redémarrage en conservant certaines informations importantes. En JES3 il contient les blocs de contrôle créés à l'initialisation, ce qui permet de redémarrer JES3 sans qu'il relise son fichier paramètre (l'init deck); il contient aussi l'état des processeurs dans le complexe et l'identification des fichiers du SPOOL. En JES2 le CHECKPOINT contient les files d'attente de travaux et de SYSOUTs, qui sont revalidées au démarrage.

JES2 et JES3 connaissent plusieurs types de démarrage après IPL selon que le SPOOL est réinitialisé (cold start) ou réutilisé (warm, quick ou hot start). Le démarrage à froid est nécessaire la première fois, ou lors de certains changements de version ou de paramétrage de JES. Le contenu du SPOOL est perdu, son reformattage peut prendre un certain temps. Les démarrages à chaud conservent le contenu du SPOOL et peuvent prendre en compte de nouveaux paramètres. Le démarrage en "hot" permet de relancer JES sans faire IPL ni arrêter les travaux en cours d'exécution. En JES3, ces types de démarrage ne concernent que le global; les JES3 locaux démarrent tous en "local start" après que le global est actif.

E) L'automatisation

MVS est un système qui procure une automatisation poussée, et, à part les opérations manuelles (montage de bande, changement de papier sur une imprimante), il y a peu de choses qu'on ne puisse automatiser sur un site MVS. Ainsi, les composants de MVS envoient peu de messages avec réponse obligatoire du pupitreur (WTOR); encore ces messages peuvent-ils avoir un traitement automatique avec les fonctions MPF (message processing facility) de MVS ou le gestionnaire de réseau NETVIEW. MPF est une fonction intéressante qui permet, par des exits, d'entreprendre certaines actions lors de l'apparition de certains messages. On peut ainsi, par exemple, déclencher les utilitaires de vidage adéquats dès que certains fichiers système sont pleins (LOGREC, fichiers SMF, fichiers DUMPxx). On peut aussi prévoir certaines actions à l'apparition de certains messages d'erreur, etc.

JES2 offre la possibilité de passer des commandes MVS ou JES2 de façon automatique à certaines heures de la journée; ces commandes doivent cependant être réinstallées chaque jour une fois qu'elles ont été exécutées. Un exemple d'application est un traitement de la SYSLOG chaque jour à minuit: commande WRITELOG pour la récupérer dans une classe de SYSOUT, puis envoi d'un external writer pour l'archiver sur disque.

Nous avons vu les mécanismes de planification des travaux (scheduling) avec JES, fondés sur l'emploi des initiateurs, des classes d'exécution et des priorités JES. Le CJS avec JES2 ou le DJC standard en JES3 permettent d'établir des relations entre les travaux et un ordre d'exécution. Un logiciel spécifique (par exemple OPC, operations planning and control) est nécessaire pour obtenir un véritable ordonnancement des travaux, avec prise en compte des fréquences et périodes de passage, mise en place des relations de conditionnement entre travaux, de procédures de reprise quand un travail se termine en ABEND, etc.

Les tendances actuelles sont les suivantes :

- le support bande est remplacé par le support cassette (chargeurs, robots) ;
- les opérations disques sont automatisées par HSM et SMS ;
- les éditions sont consultables à l'écran pour réduire le volume d'impression ;
- des automates prennent en charge les performances, la gestion du réseau et des consoles.

La finalité est la limitation des interventions (et donc des erreurs) humaines et l'augmentation de la productivité.

10. INITIALISATION D'UN SYSTEME MVS

A) Les étapes de l'initialisation

Le démarrage d'un système 370 est une opération qui se déroule en plusieurs étapes, chaque étape préparant la suivante et établissant à chaque fois un environnement plus complexe pour obtenir finalement un système complètement opérant.

La première étape, qui est commune à tous les systèmes 370 est celle d'lML (initial microcode load). Le contrôleur de processeur (3082 sur les 308X ou 3092 sur les 3090) teste la mémoire disponible pour repérer d'éventuelles erreurs et charge le microcode et l'image de la configuration matérielle qui est à sa disposition dans un IOCDS. Les IOCDS, créés par le processus d'lOCP (voir chapitre 4), sont des tables qui sont stockées en permanence dans un disque intégré au contrôleur de processeur. L'IML provoque le chargement du microcode et de ces tables dans la HSA, zone de 300 à 500 K qui n'est pas adressable. Le microcode pourra ensuite être chargé depuis la HSA vers les processeurs et le composant d'interface avec les canaux (external data controller des 308X ou channel control element des 3090). L'IML n'est obligatoire qu'en cas de mise hors tension de l'ordinateur, lors de la première installation, lors d'une maintenance ou pour prendre en compte une nouvelle configuration.

L'étape suivante est l'IPL (initial program load) de MVS, obligatoire après un arrêt (prévu ou non) du système MVS. L'opérateur déclenche l'IPL depuis la console système (directement rattachée au contrôleur de processeur) par la fonction LOAD en précisant l'adresse du disque système (SYSRES) et éventuellement certains paramètres (numéro du noyau). Ce disque a ceci de particulier qu'il supporte les fichiers nécessaires au démarrage de MVS et qu'il contient au cylindre O piste O un texte d'lPL (placé par l'utilitaire DSF) comportant 3 enregistrements. Un processus de "bootstrap" amène en mémoire le texte d'lPL: le matériel charge le premier enregistrement (IPL1); celui-ci comporte un CCW qui provoque la lecture du second enregistrement (IPL2). Cet IPL2 reçoit le contrôle; il comporte une plus longue chaîne de CCWs dont l'exécution amène en mémoire, à l'adresse zéro, le texte d'lPL proprement dit, IEAIPLOO. Le programme d'lPL nettoie le reste de la mémoire en la remettant à zéro. Il recherche sur le SYSRES le fichier SYS1.NUCLEUS pour charger des modules d'initialisation, les IRlMs (IPL resource initialization modules). IEAIPLOO fournit à ces modules quelques services qui forment déjà un embryon de système d'exploitation: il résout les fautes de page des IRlMs et supporte une dizaine de SVCs (LOAD de module, mise en WAIT ininterruptible si problème, allocation de cadres de page, opérations d'entrée-sortie). Les IRlMs IEAlPLxx chargent le noyau (IEANUC01 par défaut), DAT-off et DAT-on, depuis le SYS1 .NUCLEUS, initialisent VSM et RSM, créent la table des cadres (PFT), la SQA, l'UCB du SYSRES, et une carte du noyau (nucmap).

L'initialisation se poursuit avec le NIP (nucleus initialization program) qui met en place tous les composants de MVS. Le NIP utilise lui aussi un ensemble de RlMs (resource initialization modules) IEAVNlPx ou IEAVNPxx. L'opérateur recoit à la console MVS le rnessage suivant:

IEA101A SPECIFY SYSTEM PARAMETERS

On peut préciser ainsi le membre IEASYSxx de la bibliothèque SYS1.PARMLIB à prendre en compte (IEASYSOO par défaut) ainsi que d'autres paramètres. Les paramètres sont traités par les RlMs qui initialisent complètement les composants suivants: l'IOS, VSAM, SRM, ASM, VSM, RSM, GRS, etc. Toutes les unités périphériques sont testées, les blocs de contrôle UCBs sont initialisés et complétés par les informations de la Vatlist (pour les disques). La table des espaces-adresses (ASVT) est créée. Les fichiers principaux du système sont reconnus et traités: link-list, lpalib, LOGREC, fichiers de pagination. Les SVCs dont a besoin le NIP étaient anciennement chargées à partir de SYS1.SVCLIB. La LPA est chargée. Le catalogue maître est utilisé pour repérer les fichiers systèmes (sauf ceux qui doivent obligatoirement être sur le SYSRES) ainsi que les fichiers de pagination. Le catalogue maître est désigné selon la réponse au second message de MVS:

IEA437A SPECIFY MASTER CATALOG PARAMETER

Si la réponse est 'yy', alors le membre SYSCATyy de la SYS1.NUCLEUS indique le nom du mastercat à ouvrir et son type (VSAM ou ICF). Par défaut le membre SYSCATLG sera considéré. Finalement le module IEAVNIPX passe le contrôle à l'étape suivante.

La dernière étape est l'initialisation du "master scheduler", qui est le premier espace-adresse de MVS (ASID 1, son nom est *MASTER*). Ce composant remplit plusieurs rôles en MVS: il traite les commandes opérateur, il régit la création des espaces-adresses. L'IPL et le NIP ont construit ses tables de page, autant pour sa zone privée que pour les zones communes (qui vaudront ainsi pour tous les espaces-adresses). Certaines routines et blocs de contrôle sont mis en place de façon à pouvoir démarrer une tâche et interpréter un JCL: le SWA manager, l'initiateur, l'interface sous-système, etc. Le JCL du master scheduler est pris dans un module MSTJCLxx de la SYS1.LINKLIB. Le logiciel de sécurité (RACF par exemple) est initialisé.

Les espaces-adresses "composants système" sont créés: en MVS-XA il s'agit de PC/AUTH (qui contrôle le cross-memory), TRACE, GRS, DUMPSRV (services de dump), CONSOLE (communication avec les pupitreurs par les consoles MVS), ALLOCAS (contrôle des allocations de fichiers), SMF (comptabilité système). Ces espaces-adresses sont construits indépendamment du master scheduler par des routines particulières, et pour la plupart ne nécessitent ni commande START ni procédure en proclib.

Les sous-systèmes déclarés dans le module IEFJSSNT ou mieux dans le membre IEFSSNxx de la PARMLIB sont initialisés et l'interface sous-système mis en place (blocs SSVT, SSCVT).

Enfin le sous-système primaire (JES) est démarré, soit par une commande START présente dans le MSTJCLxx, soit par le membre COMMNDxx de la SYS1.PARMLIB en même temps que les autres commandes "automatiques".

Toute erreur incorrigible survenant dans l'IPL ou le NIP provoque une mise en attente ininterruptible avec un code explicatif. Cela survient le plus souvent dans les cas suivants: disque système incorrect (pas de texte d'lPL, pas de fichier NUCLEUS), pas de console MVS, fichiers de pagination incorrects, NUCLEUS en plusieurs extents, absence de certains fichiers système (LOGREC, SVCLIB, PARMLIB, LINKLIB), mémoire réelle insuffisante.

B) Les types de démarrage

Tout système d'exploitation peut être démarré complètement à partir de zéro (démarrage à froid) ou depuis un état intermédiaire résultant du fonctionnement précédent; dans ce dernier cas, pourvu que cet état soit cohérent, le redémarrage s'effectue plus vite.

En MVS il y a ainsi trois sortes de démarrages:

- le cold start (démarrage à froid), obtenu par le paramètre CLPA (create LPA): la LPA est complètement bâtie à partir de la SYS1.LPALIB et de sa concaténation. On effectue un cold start pour prendre en compte des modifications de la LPA (SVCs par exemple), pour tester une nouvelle version de système ou pour utiliser un nouveau jeu de fichiers page;
- le quick start (démarrage rapide), obtenu par le paramètre CVIO (clear VlO): la LPA est rechargée à partir du fichier page PLPA qui résulte du précédent fonctionnement de MVS. On ne peut prendre en compte une modification de la LPALIB, sauf par la MLPA (en fonction des paramètres de la PARMLIB). Ce type de démarrage est celui qu'on emploie après un arrêt normal de MVS;
- le warm start (démarrage à chaud): traite la LPA comme le quick start, de plus il préserve les fichiers temporaires VlO. Le démarrage en warm peut servir en cas d'arrêt brutal de MVS ("plantage") pour retraiter certains travaux (voir chapitre 11) .

C) Les fichiers du système

Certains fichiers sur disque sont indispensables au démarrage:

* SYS1.NUCLEUS: contient le noyau, les routines nécessaires à l'IPL et au NIP. Il doit résider sur le disque système.

* SYS1.SVCLIB: bibliothèque APF-autorisée qui contenait les SVCs du NIP. On peut aussi l'utiliser pour charger des programmes en MLPA. Elle doit résider sur le disque système.

* SYS1.LOGREC: enregistre les erreurs matérielles et logicielles.

* SYS1.LINKLIB: bibliothèque autorisée, contient les programmes et utilitaires système non résidents en mémoire et qu'on appelle dynamiquement

* SYS1.LPALIB: contient les modules à charger en LPA (elle est indispensable pour un cold start).

* SYS1.PARMLIB: contient tous les paramètres d'initialisation.

* SYS1.DUMPxx: optionnels, utilisés pour contenir les dumps système.

* SYS1.DAE: utilisé à partir de XA pour enregistrer et éliminer les dumps.

* SYS1.STGINDEX: contient des tables de correspondance page/case de page pour les fichiers VIO (en vue d'un warm start).

* SYS1.MANx: pour les enregistrements SMF.

* SYS1.DCMLIB : utilisé avant MVS SP220 pour sauvegarder les définitions de touches de fonction des consoles MVS.

* le MASTERCAT, les fichiers de pagination, éventuellement de swap.

Sont aussi indispensables, le cas échéant, les fichiers constituant la LINKLIB concaténée et la LPALIB concaténée (à partir de XA).

Certains fichiers, alloués par le master scheduler, sont nécessaires pour la suite des opérations:

* SYS1.PROCLIB: contient les procédures JCL supportant les tâches lancées par la commande START (notamment JES);

* SYS1.UADS décrit les utilisateurs du temps partagé TSO, SYS1.BRODCAST enregistre les messages véhiculés entre ces mêmes utilisateurs.

Le fichier SYS1.CMDLIB contient les commandes TSO, il est normalement dans la concaténation de la LINKLIB. Il fait partie des fichiers qui, considérés comme fichiers du système, ne sont cependant pas nécessaires pour l'IPL, ainsi SYS1.HELP (aide aux commandes TSO), SYS1.IMAGELIB (FCBs et UCS pour les impressions JES), SYS1.MACLIB (macro-instructions), SYS1.PAGEDUMP (utilisé pour le dump autonome), et plus généralement toutes les bibliothèques "cibles" au sens de SMP obtenues par génération de MVS: SYS1.SAMPLIB, SYS1.VTAMLIB, etc.

Doivent évidemment être disponibles au moment de l'IPL le disque système, une console MVS (depuis MVS SP220 on peut utiliser une console différente pour l'IPL et le NIP), et un minimum de mémoire réelle (au moins 4 Méga-octets, mais il est douteux que cela suffise par la suite...).

Depuis MVS-XA, le même disque système peut être utilisé par plusieurs MVS. Pour ce faire il faut placer sur différents disques les fichiers qui ne peuvent être partagés et ne garder sur le SYSRES que les fichiers accédés en lecture. Ne devraient pas être partagés le MASTERCAT, les fichiers de pagination, la LOGREC, les fichiers SMF, STGINDEX et DUMPxx. L'identification indirecte des fichiers système (voir chapitre 7) doit être utilisée pour éviter d'avoir un MASTERCAT lié exclusivement à un disque système.

D) La PARMLIB

Ce fichier de paramètres est destiné à modifier ou compléter certaines options prises lors de la génération de MVS. Indispensable au démarrage, il est utilisé par le NIP et après l'initialisation de MVS par divers composants pour accéder à leurs paramètres: IPCS, GTF, GRS, SMF, RMF, TSO, etc.

Nous allons indiquer les principaux membres de la SYS1.PARMLIB:

* IEASYSxx: ce membre indique d'une part les paramètres principaux de l'initialisation (taille de la CSA, de la SQA, nom des fichiers page, nombre maximum d'utilisateurs concurrents, etc.) et d'autre part il indique les autres membres de la PARMLIB à prendre en compte: par exemple SSN=xx,VAL=vv indiquera les membres IEFSSNxx et VATLSTvv. Le ou les membres IEASYSxx à considérer sont indiqués par le pupitreur en réponse au message SPECIFY SYSTEM PARAMETERS. Une réponse SYSP= (aa,bb) indiquera les membres IEASYSaa et IEASYSbb (les paramètres indiqués par ce dernier membre prévaudront sur ceux de IEASYSaa). Les paramètres des IEASYSxx sont réunis entre eux et avec ceux de IEASYSOO (membre par défaut, qui existe obligatoirement) pour obtenir les paramètres définitifs. Le pupitreur peut aussi spécifier des valeurs de paramètres, qui prévaudront sur celles de la PARMLIB.

* COMMNDxx: indique les commandes automatiques à passer lors de l'initialisation du master scheduler. On peut ainsi démarrer divers espaces-adresses, mettre ON ou OFF certaines unités, rajouter des fichiers page, etc.

* IEAAPFxx: table des bibliothèques de modules autorisées (sous la forme nom du fichier et volume où il réside).

* IEABLDxx: utilisé en MVS-SP pour construire les tables BLDL (voir chapitre 3D).

* IEAFlXxx: indique les modules à mettre en FLPA (fixed LPA).

* IEALODxx: indique des modules de la PLPA fréquemment appelés. Le NIP les placera dans la LPA active (avec ceux de la MLPA et de la FLPA), où ils seront localisés plus vite que par la consultation habituelle du répertoire de la LPA.

* IEAPAKxx: indique les modules de la LPALIB à regrouper en mémoire lors du chargement de la PLPA. L'idée est de mettre à proximité les uns des autres des modules exécutés ensemble, pour éviter des fautes de page. La routine RIM de création de la PLPA essaie d'utiliser au mieux la mémoire virtuelle en chargeant les modules de plus de 4K sur des frontières de page et complétant les "trous" avec des modules plus petits (on charge les modules par ordre de taille décroissante). Les modules de chaque groupe de la "pack list" sont chargés les premiers, si possible dans une seule page.

* LNKLSTxx: liste des bibliothèques de loads de la link-list, logiquement concaténée à la SYS1.LINKLIB. Elles doivent être cataloguées au MASTERCAT. En MVS-SP elles sont automatiquement APF-autorisées. A partir de XA on peut n'autoriser qu'une partie de la link-list.

* LPALSTxx: en XA, liste des bibliothèques de loads de la Ipa-list, concaténées à la SYS1.LPALIB, à prendre en compte lors d'un démarrage à froid. Elles doivent être autorisées pour pouvoir procéder au chargement.

* MPFLSTxx: liste des messages MVS à supprimer de l'affichage des consoles ou, uniquement à partir de XA, à traiter par un exit particulier: on peut ainsi répondre automatiquement aux REPLYs de MVS, supprimer ou réaffecter les messages à d'autres consoles, déclencher certaines actions par commande MVS, etc.

* VATLSTxx: liste des disques à monter ("volume attribute list"), avec leurs attributs de montage (résident de façon permanente, réservé ou démontable) et d'utilisation (privé, public, stockage).

* IKJTSOxx: paramétrage de TSO (commandes et programmes autorisés).

Avec ESA on trouvera des membres COFDLFxx (hiperbatch), COFVLFxx (VLF), CSVLLAxx (bibliothèques de la LLA), EXSPATxx (boucle spin), IGDSMSxx (SMS), DEVSUPxx (IDRC), etc. A partir de MVS/SP4 on aura LOADxx (paramètres d'IPL), COUPLExx (Sysplex), ALLOCxx, NUCLSTxx, etc.

E) La création des espaces-adresses

Il y a en MVS trois sortes d'espaces-adresses:

- les "started tasks", tâches démarrées par la commande START. Le JCL est obtenu à partir d'une PROCLIB. La commande MOUNT, utilisée pour monter logiquement (car elle ne s'accompagne pas forcément d'un montage physique) un support magnétique sur son unité, génère aussi une "started task";
- les "jobs batch". Leur structure comporte un TCB qui n'existe pas pour les "started tasks", l'initiateur, contrôlé par le sous-système primaire JES. Un job est soumis par la commande SUBMIT de TSO ou par écriture dans un "internal reader" de JES;
- les utilisateurs TSO, créés par une commande LOGON depuis un terminal. Ces espaces-adresses comportent un TCB "TMP" (terminal monitor program) qui répond aux commandes de l'utilisateur.

La création d'un espace-adresse s'effectue ainsi:

* le "master scheduler" traite la demande et affecte un numéro ASID (address-space identifier) à l'espace-adresse et crée l'ASCB (memory request). Le numéro ASID n'est pas le numéro affecté par JES: il est réutilisable quand l'espace-adresse se termine. Il est limité par le MAXUSER de la PARMLIB (nombre maximum d'espaces-adresses);

* SRM est informé de la création; il peut la refuser en cas de manque de certaines ressources (cadres de page, mémoire auxiliaire notamment);

* VSM fournit la mémoire virtuelle nécessaire et crée la LSQA. RSM crée les tables de page et la table de segments (memory create);

* le contrôle est passé à la RCT (region control task) de l'espace-adresse, qui poursuit la création et initialise par ATTACH la tâche de DUMP et la tâche STC (started task control);

* la tâche STC crée s'il le faut le JCL en mémoire (cas des commandes START, MOUNT) et le passe à JES. Les routines de LOGON font de même pour un utilisateur TSO;

* JES copie le JCL sur le SPOOL, le traduit en texte interne (conversion) et lui assigne un numéro (jobid). A partir de ce texte interne les blocs de contrôle nécessaires sont mis en SWA (interprétation). Enfin les routines d'initiation allouent les fichiers nécessaires et par ATTACH créent la tâche de dernier niveau correspondant à l'EXEC PGM du JCL.

Les tâches liées à un espace-adresse sont donc, en suivant l'ordre de chaînage des TCBs:

* la RCT, tâche de plus haut niveau et de plus haute priorité;

* la tâche de DUMP, qui permet de prendre un dump des tâches de plus bas niveau sans affecter leur environnement propre;

* la STC, présente dans tous les cas, utile surtout pour les "started tasks";

* l'INIT, initiateur des jobs batch;

* le TMP (terminal monitor program) des utilisateurs TSO. Il répond aux commandes de l'utilisateur. Le TMP est indiqué dans l'EXEC PGM de la procédure de LOGON;

* enfin la tâche ou les tâches de plus bas niveau, liées pour la première à l'EXEC PGM du JCL de la "started task" ou du job, et pour celles de niveau inférieur aux commandes ATTACH émises ensuite.

La figure donne le détail de la hiérarchie des tâches pour un job batch, un utilisateur TSO et une started task (il s'agit de JES2). On remarquera la structure verticale pour TSO, correspondant à plusieurs couches pour différents environnements, en contraste avec une structure horizontale "en râteau" pour JES2, provenant de la coexistence de tâches spécialisées relativement indépendantes. Chaque programme correspond à une tâche unique :

- IEAVAR00 : region control task
- IEFSD060 : initialisation de started task
- IEAVTSDT : tâche de SVC dump
- IEFIIC : initiateur de job batch

- IKJEFT01 : initiateur TMP
- IKJEFT02 : tâche principale du TMP
- IKJEFT09 : TMP de second niveau
- ISPMAIN : tâche principale ISPF
- ISPTASK : tâche ISPF correspondant à un écran logique

- HASJES20 : tâche principale JES2
- HOSCNVT : tâche de conversion
- HOSPOOL : tâche d'allocation de spool
- HASPVTAM : tâche SNA
- HASPIMAG : tâche de chargement d'image
- HASPACCT : tâche SMF
- $HASPWTO : tâche de communication
- HOSALLOC : tâche d'allocation et désallocation dynamique
- HA$PSUBS : services généraux de sous-tâches

F) Evolution

MVS évolue constamment vers une automatisation plus marquée et plus de commodité d'emploi (le RAS). Depuis longtemps, certains sites recherchent un fonctionnement en permanence sans nécessité de "power-off" ou arrêt de MVS ("non-disruptive change"), que ce soit pour installer de nouveaux matériels ou de nouvelles fonctions logicielles. MVS/ESA SP4 apporte beaucoup de nouveaux moyens pour rendre plus faciles certaines opérations liées à l'exploitation ou au système. Il devient possible de définir la configuration de périphériques sans devoir arrêter le système. HCD (hardware configuration definition) et DRM (dynamic reconfiguration management) permettent de construire l'IOCP dynamiquement à partir d'un interface graphique sur micro-ordinateur : la prise en compte d'un nouvel IODF (I/O definition file) entraîne la mise à jour des informations sous-système canal en HSA et la mise à niveau des blocs de contrôle MVS. Les voies ESCON peuvent aussi être adaptées dynamiquement. Du côté du logiciel, l'effort est le même, que ce soit pour les SVCs, la table des bibliothèques APF et l'IPL lui-même, qui peut être conduit sans opérateur.

11. DISPONIBILITE DU SYSTEME MVS

Un système d'exploitation doit être "disponible" à tout moment. Cela ne signifie pas qu'il n'ait aucun droit à l'erreur, la perfection n'existant en informatique pas plus qu'ailleurs, mais il doit surmonter sans impact notable pour les utilisateurs tous les problèmes qui peuvent survenir. Le concepteur d'un système doit adopter un parti pris de modestie: tout programme étant sujet à l'erreur (qu'il y ait faute de l'auteur ou non), il faut établir, au moins pour les programmes les plus critiques, un environnement de reprise (recovery) qui puisse traiter l'erreur et prendre les décisions qui s'imposent. De ce point de vue MVS s'avère être d'une fiabilité remarquable, sans doute supérieure à celle des systèmes "frères" VM ou VSE.

Nous étudierons les moyens qu'emploie MVS pour se prémunir contre les erreurs logicielles (abends, boucles, waits) et matérielles (machine, E/S).

A) Les erreurs logicielles

Un ABEND (abnormal end, voir chapitre 6.F) est la fin anormale d'une unité de travail, provoquée par l'intermédiaire de la macro ABEND (SVC 13) par les routines du système ou du programme utilisateur.

Le composant de MVS chargé de traiter les fins normales ou anormales est RTM (recovery and termination manager). RTM se compose de RTM1, qui traite les erreurs de base: interruptions "program-" ou "machine-check", restart, SVC incorrecte, erreur dans le DAT ou la pagination; et de RTM2, appelé uniquement au travers de la SVC 13 (par RTM 1 ou toute routine décidant d'un abend) .

On peut se protéger d'une erreur par une FRR (functional recovery routine) pour les programmes du système (modes SRB ou TCB avec verrou, routines ininterruptibles ou routines en mode superviseur clé O) ou une ESTAE (extended specify task abnormal exit) pour les programmes habituels. La routine FRR ou ESTAE de reprise peut décider de continuer le traitement à un point donné (retry) ou se déclarer incompétente à résoudre le problème et passer le contrôle à une routine de reprise à un niveau supérieur (percolation); en effet, les FRR ou ESTAE peuvent être "empilées", de sorte que la plus récente est invoquée la première dès qu'il y a erreur. Cet empilement augmente les chances de résoudre l'erreur sans devoir arrêter le traitement.

Une boucle dans une routine est un problème plus délicat. Pour une raison inconnue une même séquence d'instructions se répète sans discontinuer. Il s'agit d'une anomalie gênante moins patente qu'une erreur "franche" et parfois plus difficile à diagnostiquer. On peut être dans le cas d'une boucle:

* interruptible: un processeur est utilisé à 100 %, il y a peu d'E/S, on garde encore le contrôle de la console MVS. Il suffit généralement de tuer l'espace-adresse fautif (commande CANCEL);

* ininterruptible: on n'a plus le contrôle de la console MVS. On doit se résoudre, après avoir pris les éléments nécessaires (enregistrement de la boucle notamment), à invoquer la fonction de RESTART pour éliminer le coupable.

Un cas de boucle ininterruptible en environnement multi-processeurs est celui du verrouillage par spin lock (voir chapitre 2.D.4). Normalement une routine en "spin loop" est très courte; si la boucle dure de façon anormale (à cause par exemple d'une défaillance d'un processeur) il faut intervenir. Le temps de détection est fixé à 15 secondes en MVS-SP1, 40 secondes en MVS-XA, et par défaut à 10 secondes en MVS-ESA (paramétrable grâce au membre EXSPATxx de la PARMLIB). MVS ne pouvant plus communiquer par la voie des E/S habituelles utilise un dispositif particulier, le DCCF (disabled console communication facility), pour accéder à une console système et indiquer les décisions à prendre.

Il peut aussi arriver que la machine soit en attente (wait) et n'effectue plus aucun travail productif. Là encore on peut distinguer:

* un wait interruptible: une partie seulement du système semble bloquée (JES, TSO, le réseau, etc.). Les causes peuvent être nombreuses: message au pupitre avec attente de réponse, enqueues, RESERVEs émis depuis un

autre système sur des disques cruciaux (spool, fichiers page, catalogues), mauvais fonctionnement d'un élément du dispositif d'E/S, manque de mémoire (storage shortage), etc.;

* un wait ininterruptible: un message confirme cet état et l'alarme de l'ordinateur se met à sonner. MVS entre en "disabled wait" quand une erreur grave (souvent d'origine matérielle) l'empêche de continuer et qu'il ne peut la signaler autrement (wait code dans le PSW). Dans quelques cas le RESTART relance le système. Le chapitre 10.A indique quelques types d'erreurs survenant à l'IPL.

Le checkpoint-restart est un dispositif logiciel qui permet aux programmes d'application de se prémunir contre un arrêt inopiné de MVS, en évitant de reprendre un traitement depuis le début. A intervalles réguliers le programme, par la macro CHKPT, sauvegarde certaines informations importantes sur un fichier checkpoint; un fichier "journal" est maintenu sur le spool par le système pour garder une copie des principaux blocs de contrôle du job (en SWA) à chaque modification importante (open/close de fichiers, allocation, etc.). Lors du restart, la SWA est reconstruite à partir du journal et on peut recommencer le traitement au dernier checkpoint connu, en supposant que les fichiers temporaires n'aient pas été détruits, ni les fichiers VIO (IPL avec warm start de MVS). Le checkpoint-restart à notre connaissance est très peu utilisé (il nécessite une programmation bien particulière); il n'aurait d'intérêt que pour des travaux très longs, pour accélérer la reprise. Il est préférable en pratique de créer dans un job un assez grand nombre de petites étapes pour faire du restart au niveau du step.

B) Les erreurs matérielles

Les erreurs processeur ou mémoire, quand elles ne peuvent être corrigées par le matériel, sont passées au gestionnaire d'erreur machine (MCH, machine check handler), qui les analyse, écrit un enregistrement sur la SYS1 .LOGREC, et appelle s'il y a lieu une FRR de reprise. Si la reprise est impossible, un autre processeur est averti par un SIGP "signal d'alarme"; en environnement mono- processeur, on aboutit à un wait ininterruptible.

En environnement multiprocesseur, I'ACR (alternate CPU recovery) permet à un processeur "sain" de reprendre un traitement en cours d'exécution sur un processeur défaillant, qu'il s'agisse d'erreur machine, d'erreur dans le MCH, de boucle "spin". Les FRRs de la tâche interrompue sont appelées par RTM, si elles existent (sinon la tâche est terminée par un abend système OF3); le processeur défaillant est mis hors jeu (offline) par une instruction SIGP STOP. Les tâches qui avaient une affinité avec ce processeur sont terminées (il s'agit des tâches déclarées en PPT pour s'exécuter uniquement sur un processeur).

L'ACR est une sécurité intéressante pour les ordinateurs à plusieurs processeurs. En MVS-SP1, où l'on est limité à 2 processeurs au maximum qui se répartissent les unités d'E/S, on risque de perdre l'accès aux unités "non symétriques" dédiées au processeur défaillant; les E/S qu'il a émises doivent être relancées, les RESERVEs sur les disques partagés annulés par un RESERVE "inconditionnel". L'ordinateur devient en fait un monoprocesseur; le jeu de canaux (channel set) du processeur en erreur peut être repris par le processeur restant grâce au "channel set switching", en alternance avec son propre jeu de canaux; cependant à un instant donné un seul ensemble de canaux est connecté, aussi l'impact sur les performances est sensible.

Vis-à-vis des erreurs liées aux unités d'E/S, MVS dispose de quelques moyens de prévention ou de détection:

* le MIH (missing interrupt handler), composant de l'IOS, inspecte les UCBs des unités pour détecter les demandes d'E/S ou de montage non satisfaites depuis "trop longtemps", par défaut: 15 secondes pour un disque, 3 minutes pour les autres types d'unités (délais de rembobinage des bandes). Le manque de "retour" (par interruption du processeur) d'une E/S peut être dû à un mauvais fonctionnement du matériel ou, plus souvent pour les disques, à un RESERVE émis sur l'unité depuis un autre système. L'opérateur est averti, et le MIH tente de relancer l'E/S. Le membre IEClOSxx de la PARMLIB permet en XA de changer les seuils de détection. Pour un MVS sous VM, il faut donner au MIH de VM et à celui de MVS des seuils assez différents, pour laisser à un seul des deux systèmes le soin de traiter le problème;

* le HOT I/O: il s'agit d'un problème inverse à celui du MIH. Le mauvais fonctionnement d'une unité de contrôle provoque des interruptions d'E/S inattendues et incessantes. Il y a risque notamment d'épuiser la zone de mémoire commune par création de blocs de contrôle d'E/S trop nombreux en réponse à ces interruptions. Le nombre d'interruptions seuil est par défaut de 100; MVS peut réagir en mettant "offline" l'unité ou le chemin

canal utilisé. Le membre IEClOSxx permet d'indiquer l'action à entreprendre la première fois que se produit le problème, et les fois suivantes;

* le DDR (dynamic device reconfiguration) permet de changer (swap) d'unité (dérouleur de bande, imprimante ou unité de disques pour les anciens disques démontables) quand une erreur d'entrée-sortie se produit sur une unité. Il est évident que le swap sera inefficace si le problème est lié au volume (data check) plutôt qu'à l'unité qui le supporte. Le swap peut être aussi demandé par l'opérateur pour récupérer une unité (commande SWAP de MVS).

12. LA SECURITE D'UN SYSTEME MVS

A) Généralités

A une époque où l'outil informatique, d'importance cruciale pour les entreprises, est universellement répandu, et où d'importants volumes de données circulent de par le monde sur les lignes de télécommunications, tout système informatique doit être efficacement protégé. Il faut éviter qu'une erreur involontaire autant qu'une attaque en règle ne puissent conduire à un blocage du système, à sa destruction, à la divulgation, l'altération ou la destruction de données importantes. Bien évidemment, la sécurité d'un site informatique est un tout, et englobe des éléments aussi dissemblables que la protection contre le feu, les inondations ou autres catastrophes, le contrôle d'accès à la salle machine, le cryptage des données, l'emploi d'un site informatique de secours, d'un lieu d'archivage, etc. Nous nous en tiendrons à la sécurité logique des informations en MVS.

Nous étudierons les moyens dont MVS dispose en standard (déjà rencontrés dans d'autres chapitres) et ceux, optionnels, qui existent pour mettre en oeuvre une plus grande protection.

B) La sécurité interne en MVS

1) MODE PROBLEME ET MODE SUPERVISEUR

Le mode **superviseur** permet d'exécuter les instructions **privilégiées** réservées normalement au système. Le mode **problème** est celui qu'utilise habituellement tout programme d'application sans privilège particulier. Le basculement vers le mode superviseur est automatique lors d'une interruption ou d'une SVC; il peut être établi par une SVC particulière (MODESET) si le programme est autorisé à le faire (voir plus loin). Ce système du tout ou rien (autorisé ou non-autorisé) n'est pas souple (d'autres systèmes d'exploitation prévoient différents états ou anneaux de protection procurant différents privilèges à mesure qu'on approche d'un anneau central) mais il a le mérite d'être clair; il oblige notamment l'utilisateur, pour réaliser une opération privilégiée (ouverture d'un fichier, acquisition de mémoire, etc.), à s'en remettre au système par le filtre des SVCs. Certaines instructions "semi-privilégiées" (liées surtout au cross-memory) peuvent être utilisées en mode problème selon la valeur d'un masque de clés (PKM).

2) LA PROTECTION MÉMOIRE

Chaque page mémoire comporte une **clé** (de 0 à 15) fonction du composant du système qui la détient (clé 8 pour l'utilisateur commun); la modification d'une page n'est possible que si la clé du PSW du programme correspond à la clé de la page. La clé 0 est réservée au système et ouvre l'accès à toute la mémoire. Ainsi, aux deux extrémités on trouvera d'une part le programme fonctionnant en mode superviseur clé 0, qui aura droit à toutes les opérations et l'accès à toute la mémoire; d'autre part l'utilisateur normal en mode problème et clé 8 qui n'aura accès en lecture/écriture qu'à son espace-adresse (pas dans sa totalité à cause des zones communes de clé système). La mémoire virtuelle sépare logiquement tous les utilisateurs de clé 8 puisque les jeux de tables de pages sont distincts; quant au cross-memory, il est suffisamment quantifié pour éviter les intrusions d'un espace-adresse dans un autre.

3) L'AUTORISATION APF

Un programme est considéré par MVS comme "*autorisé*" s'il s'exécute en mode superviseur (1 6e bit du PSW à 0), ou avec une clé système (de 0 à 7) ou si la tâche est APF-autorisée (bit JSCBAUTH). Ainsi une SVC reçoit le contrôle en mode superviseur clé 0, pleinement autorisée; cela ne l'empêche pas de pouvoir tester l'autorité de l'appelant (macro TESTAUTH, SVC protégée APF) ou d'obtenir sa clé de protection pour éviter tout abus. Le bit JSCBAUTH (lié au TCB) est positionné normalement quand la tâche est lancée avec un programme autorisé chargé depuis une bibliothèque autorisée (attention à certaines SVCs que l'on écrit par commodité pour mettre ce bit dans un environnement non autorisé, TSO ou ISPF par exemple).

Les bibliothèques APF-autorisées contiennent des modules autorisés à effectuer toutes les opérations, sous les conditions suivantes: code autorisation 1 indiqué à l'édition des liens; exécution avec une STEPLIB ou JOBLIB autorisée (s'il s'agit d'une concaténation, toutes les bibliothèques doivent être autorisées) .

Ces bibliothèques doivent être particulièrement surveillées et protégées le mieux possible (voir les méthodes au paragraphe C). Elles constituent la meilleure porte ouverte aux pirates, surtout quand la présence sur un site de nombreux logiciels réclamant l'autorisation APF a conduit à une prolifération de bibliothèques de modules APF-autorisées. Il suffit d'une seule bibliothèque APF insuffisamment protégée pour installer des programmes "pirates" qui pourront à leur guise examiner la mémoire (recherche des mots de passe ou d'autres zones intéressantes), modifier des blocs de contrôle (par exemple les blocs de contrôle du logiciel de sécurité, RACF ou autre, pour se donner toute autorité sur toutes les ressources), passer des commandes MVS à la place du pupitreur, charger en mémoire des exits pirates, ou simplement bloquer le système (en modifiant l'adresse de la CVT ou d'autres blocs fondamentaux) et même empêcher qu'il puisse redémarrer (il suffit d'altérer ou de détruire le fichier NUCLEUS). Pourquoi le pirate perdrait-il son temps à rechercher d'éventuelles failles dans le système MVS (manière de procéder du pirate besogneux ou optimiste) alors qu'on lui laisse une telle porte ouverte ? Il convient donc de protéger les bibliothèques APF et d'empêcher toute modification de la table des bibliothèques APF tant en mémoire que dans la PARMLIB.

4) LE PARTAGE

Les moyens dont dispose MVS pour contrôler le partage des ressources ont été vus dans les premiers chapitres:

- les verrous et l'instruction TEST AND SET en multitraitement (plusieurs processeurs);

- l'ENQUEUE/DEQUEUE pour sérialiser l'accès aux fichiers, aux volumes ou à toute ressource accessible par plusieurs tâches;

- les options de partage de VSAM (share options) garantissent l'intégrité des données en lecture ou en écriture. En fait elles génèrent des ENQUEUEs qui peuvent d'ailleurs être propagées à d'autres systèmes MVS avec un logiciel tel que GRS.

Le mécanisme d'ENQUEUE/DEQUEUE fournit une protection logique: comme pour les verrous, ce n'est pas la ressource elle-même qui est protégée mais une représentation symbolique. C'est le seul moyen en MVS d'empêcher par exemple un utilisateur de détruire un fichier tandis qu'un autre le lit. Toute allocation de fichier par une carte DD du JCL s'accompagne d'un ENQUEUE (de qname SYSDSN) sur le nom du fichier, indépendamment du volume sur lequel il réside. Ceci est parfois gênant quand on manipule des fichiers différents mais de même nom: on ne peut par exemple détruire l'un si l'autre est alloué par ailleurs.

Comme on l'a vu au chapitre 4, l'ENQUEUE avec RESERVE permet à un système MVS d'accaparer physiquement un volume disque entier au détriment des autres ordinateurs auxquels le disque est connecté. Cet ENQUEUE est utilisé notamment pour les accès aux catalogues ou aux VTOCs de disque quand le disque est déclaré "shared" (susceptible d'être accédé depuis plusieurs ordinateurs). Avec GRS ou un logiciel équivalent, on peut tempérer le RESERVE en le transformant en simple ENQUEUE, en partant du principe qu'il est inutile de bloquer tout un volume quand seul un fichier du volume est concerné.

C) Les autres moyens de protection

1) LES MOTS DE PASSE

Avec l'emploi de mots de passe, il s'agit de restreindre l'accès à des ressources protégées et d'authentifier l'accès des personnes autorisées.

En MVS on peut protéger un fichier disque ou bande par un mot de passe en lecture ou en écriture: on emploie le sous-paramètre PASSWORD (protection en lecture/écriture) ou NOPWREAD (protection en écriture) de l'ordre LABEL de la carte DD, ou la commande TSO PROTECT. L'information "fichier protégé par mot de passe" est enregistrée dans la VTOC du disque ou sur le label de la bande. Le mot de passe est enregistré dans un fichier système, le fichier PASSWORD; quiconque veut accéder au fichier doit fournir le mot de passe du fichier. Ce mode de protection déjà ancien a les inconvénients suivants: les mots de passe sont stockés en clair dans le fichier PASSWORD, aussi ce dernier doit lui-même être protégé en lecture; le mot de passe doit être fourni par une réponse pupitreur (sauf sous TSO); enfin la connaissance du mot de passe donne l'accès au fichier à toute personne qui le possède, quel que soit son droit à y accéder par ailleurs.

Les fichiers VSAM peuvent aussi être protégés par mot de passe, avec 4 niveaux différents: le READPW donne l'accès en lecture, l'UPDATPW en écriture, le CONTROLPW au niveau control interval, le MASTERPW donne tous les accès. Le ou les mots de passe sont indiqués par les commandes DEFINE ou ALTER; on peut indiquer un nombre maximum de tentatives. Les mots de passe sont enregistrés au catalogue; lors de l'accès aux fichiers, ils peuvent être indiqués par le pupitreur, l'utilisateur TSO ou par le programme qui effectue l'accès.

Enfin on peut contrôler l'accès au système grâce au mot de passe, qui permet en principe de vérifier que l'utilisateur est bien qui il prétend être. Ainsi l'accès au système MVS par le logiciel de temps partagé TSO est contrôlé par l'entrée d'un nom d'utilisateur (userid) et d'un mot de passe connu de l'utilisateur seul. Dans le système de protection standard (hors logiciel de sécurité du type RACF), les mots de passe sont stockés dans le fichier SYS1 .UADS qui décrit les caractéristiques des utilisateurs TSO; le stockage est en clair, ce qui oblige à protéger ce fichier en lecture. La modification des mots de passe est malaisée (commande ACCOUNT). Les travaux batch ne sont pas reliés à l'utilisateur qui les a soumis, et ne bénéficient donc pas de ses degrés d'autorisation.

L'emploi des mots de passe pour contrôler la connexion est une procédure souple: plutôt que d'identifier une personne par quelque chose qui la caractérise seule (empreinte digitale, signature, etc.) ou quelque chose qu'elle possède seule (carte magnétique, badge, etc.), on l'identifie à partir de quelque chose qu'elle est seule à connaître (ou est censée l'être). Le danger est aussi à la mesure de la souplesse: un mot de passe peut être facile à deviner, car trop simple (le nom ou les initiales de la personne), dévoilé par mégarde ou inattention (retranscrit sur un autre support), ou assez court pour être trouvé en essayant toutes les combinaisons possibles (le système doit par précaution rejeter l'accès après un certain nombre de tentatives infructueuses; RACF quant à lui "révoque" purement et simplement l'utilisateur en lui

interdisant tout accès ultérieur). Enfin les mots de passe sont souvent stockés en mémoire par le système: par JES qui les reçoit dans la carte JOB (paramètre PASSWORD), par TSO qui les stocke par exemple pour l'exit de SUBMIT (sauf dans les versions récentes) et par la plupart des logiciels de multi-sessions. Les mécanismes de protection mémoire sont souvent insuffisants et d'autres méthodes doivent être étudiées.

2) LE CHIFFREMENT

Un niveau de sécurité supplémentaire peut être obtenu en chiffrant les mots de passe avant de les stocker. Chiffrer des données consiste à les brouiller de façon à les rendre illisibles par des personnes qui ne connaissent pas la méthode cryptographique employée ou certains de ses paramètres secrets (les clés). Ainsi la connaissance de ces données (les mots de passe une fois encodés par RACF) ne sert à rien, pourvu que l'algorithme de chiffrement ne puisse être inversé facilement. On contrôle alors l'accès en chiffrant le mot de passe fourni et en le comparant au mot de passe déjà stocké. Cette opération ne nécessite aucun moyen de déchiffrement. Il en va autrement quand il s'agit de chiffrer un texte pour le rendre secret avant de le communiquer à son destinataire; des logiciels ou des machines spécifiques existent à cet effet. Les algorithmes de chiffrement et de déchiffrement utilisent alors des "clés" analogues aux mots de passe et seules connues de l'émetteur et du destinataire.

Actuellement deux grands algorithmes de chiffrement existent:

- le **DES** (data encryption standard), créé par IBM (et très largement utilisé depuis dans les solutions de sécurité de la Compagnie) est depuis 1977 le standard de fait aux USA. Il permet de chiffrer des textes par blocs de 64 bits. La clé de chiffrement est de 64 bits (seuls 56 sont utilisés). Le texte est chiffré par permutation et application en 16 passes d'une fonction de chiffrement qui fait intervenir la clé. Cette fonction, facile à programmer, fait intervenir des tables, des permutations et des additions. Le déchiffrement s'effectue avec la même clé que pour le chiffrement, en appliquant au texte chiffré un procédé identique (algorithme symétrique). La protection découle évidemment non de l'algorithme, qui est public et universellement utilisé, mais de la clé secrète connue seulement des opérateurs. Suspecté quelquefois de faiblesse, le DES a encore de beaux jours devant lui en raison de sa simplicité de mise en oeuvre;

- le **RSA** (Rivest, Shamir, Adleman) utilise des clés différentes (clé publique, clé privée) pour le chiffrement et le déchiffrement. La clé de chiffrement peut être rendue publique, ce qui lui permet d'être employée par de nombreuses personnes. La clé de déchiffrement, connue seulement du destinataire, ne peut être obtenue aisément à partir de la clé publique, car il faudrait pour cela effectuer la mise en facteur du produit de deux très grands nombres premiers (inconnus). Inversement on peut rendre la clé de chiffrement secrète et la clé de déchiffrement publique; cela permet à n'importe qui d'authentifier l'identité de l'opérateur, seul capable de chiffrer un texte donné (signature "digitale"). Le RSA est souvent considéré comme plus sûr que le DES; en revanche il nécessite beaucoup plus de calculs et est donc beaucoup plus lent. Avec la souplesse des deux clés, le RSA est souvent mis à contribution dans les télécommunications, alors que le DES conduit à une gestion des clés des correspondants bien plus lourde.

En MVS on peut systématiser le chiffrement avec des unités cryptographiques et les logiciels adéquats (PCF, CUSP). La récente architecture cryptographique d'IBM (CCA) proposée sur les machines ES/9000 convient bien aux sites qui ont des besoins de chiffrement massifs : elle met en oeuvre le DES dans le matériel même (un TCM est dédié au processeur de chiffrement), avec une gestion très élaborée des clés. Le point le plus crucial est celui de la gestion des clés, puisqu'en définitive toute la sécurité repose sur elles.

Les fichiers peuvent être chiffrés (commande AMS REPRO avec les options ENCIPHER, DECIPHER) selon le standard DES; on peut aussi coder les télécommunications. On allonge évidemment les temps de traitement, aussi on ne doit employer ces moyens que si une confidentialité absolue est requise. RACF emploie le DES à seule fin de coder les mots de passe avant de les stocker sur disque (d'autres algorithmes peuvent être imposés sous forme d'exit). Il s'agit d'un chiffrement à sens unique, dans lequel le mot de passe n'est pas le texte que l'on chiffre; en fait il sert de clé (après quelques manipulations simples) pour chiffrer le userid (ce qui donne des mots de passe après chiffrement différents quand bien même tous les utilisateurs adopteraient le même mot de passe).

Nous pensons que le chiffrement, méthode finalement très ancienne de protection de l'information, sera de plus en plus employé en informatique pour répliquer aux risques de détournement et de piratage. Utilisés avec les précautions qui s'imposent, les procédés basés sur le DES ou le RSA offrent une protection presque absolue des données vitales.

3) LE LOGICIEL RACF

RACF est le produit-programme IBM de sécurité des données pour MVS (une version existe aussi pour VM).

Avec RACF on peut "industrialiser" la sécurité du système en évitant l'emploi des mesures ponctuelles vues plus haut (protection des fichiers par mots de passe). On définit d'une part des utilisateurs et des groupes d'utilisateurs; chaque utilisateur possède un identifiant (userid) et un mot de passe qui lui permet de se connecter aux différents logiciels (TSO, IMS, CICS, etc.) ou de transmettre les droits qu'il détient aux travaux qu'il soumet. L'accès au système et aux grands logiciels peut être ainsi contrôlé. On peut imposer des contraintes sur le mot de passe (longueur, format, changement périodique). D'autre part, on définit des ressources à protéger (une ressource non connue de RACF n'est pas protégée) et des classes de ressources: fichiers, volumes disques ou bandes, programmes, terminaux, transactions CICS ou IMS, etc., peuvent être protégés. Les règles de protection sont enfin déterminées: on donne un accès par défaut à la ressource (universal access); on dispense éventuellement à certains utilisateurs des autorisations: accès en lecture (READ), écriture (UPDATE, CONTROL), tous accès (ALTER, même la destruction est permise), aucun accès (NONE). Depuis RACF 1.8.1 un accès de type EXECUTE permet d'exécuter un programme quand bien même on n'a pas le droit de le lire. Les contrôles sont effectués juste avant l'accès à la ressource: OPEN pour un fichier, initialisation pour un travail, LOGON de TSO, SIGNON de CICS ou IMS, chargement de programme, appel de transaction. Le contrôle des fichiers est systématique depuis les premières versions de DFP avec RACF "always-call".

Les fichiers peuvent être protégés de manière logique, par un profil générique: le nom (ou le début du nom) du fichier est alors protégé, indépendamment de son support physique. On peut ainsi protéger tous les fichiers du système de base par un profil "SYS1.*". Au contraire, une protection physique est fournie par le profil discret: un indicateur "fichier protégé par RACF" est mis dans la VTOC du disque. La protection est ainsi conservée même après sauvegarde et restauration sur un autre site. Ce genre de protection est cependant beaucoup moins souple et tend à disparaître. La protection des fichiers par RACF, qu'elle soit ou non par génériques, remplace la protection traditionnelle par mots de passe.

Certaines options puissantes permettent d'affiner les contrôles. On peut imposer des périodes d'utilisation de certains userids. On peut empêcher la création de fichiers dont les règles de protection ne seraient pas définies (option PROTECTALL).

Avec RACF, MVS peut prétendre aux classifications C2 (protection discrétionnaire) pour MVS-XA et B1 (protection obligatoire) pour MVS-ESA (il s'agit des standards du département de la Défense américaine, répertoriés dans le "livre orange"). Le niveau B1 est obtenu par un contrôle affiné et rigoureux (MAC, mandatory access control). La "sensibilité" de l'objet protégé détermine ses besoins de

protection, avec un label de sécurité (top secret, secret, sensible, usage interne, etc.). L'ancien contrôle "discrétionnaire", basé sur l'identité du sujet, existe toujours, mais s'exerce seulement après le contrôle MAC : il représente ce qu'on appelle le "besoin d'en connaître". Les règles d'accès MAC découlent d'une comparaison entre les niveaux de sécurité (security label, seclabel) du sujet et de l'objet : selon les cas le sujet pourra lire seulement, écrire seulement, ou lire/écrire (propriété de confinement).

A partir de RACF 1.9, on peut identifier plus strictement tous les utilisateurs (par exemple les pupitreurs à la console), contrôler les commandes MVS, mieux protéger les SYSOUTs, passer à un utilisateur les autorités d'un autre (surrogate userids), etc. Avec ESA et SMS, RACF prend de plus en plus d'importance et centralise toutes les données sur les utilisateurs de l'informatique (remplacement de l'UADS de TSO) et sur les possibilités qui leur sont ouvertes dans le système (Hiperbatch, SMS, DFP, etc.).

D) Précautions élémentaires

La sécurité n'est jamais assurée à 100 %; en outre, plus on approche d'une limite idéale, plus les investissements deviennent colossaux. Un système étant une chaîne de composants en étroite intrication, sa sécurité est celle du maillon le plus faible : une approche totale est nécessaire. Cela dépasse rapidement le cadre étroit du système d'exploitation et nous vous renvoyons aux ouvrages spécialisés pour approfondir ce thème.

En nous fixant un but de protection du système en lui-même, avant celle du site informatique et des applications, nous proposons quelques dispositions capables d'assurer, même en cas de malveillance, erreur ou problème matériel, la bonne marche du système MVS. Voici une liste non exhaustive de ces quelques précautions:

- protéger tous les fichiers système, les bibliothèques APF, le MASTERCAT. Etre bien conscient des "points faibles" possibles: les bibliothèques APF, les SVCs utilisateur, les exits;

- éviter les passe-droits et donner à bon escient certains attributs tels que ACCOUNT ou OPER aux utilisateurs TSO, SPECIAL ou OPERATIONS aux users RACF;

- contrôler les userids "par défaut" (tel que IBMUSER, userid initial de RACF) ; modifier aussi les mots de passe "par défaut" livrés pour utiliser certains produits (par exemple Omegamon) ;

- contrôler l'accès à certains utilitaires tels que IEHINITT, AMASPZAP, ICKDSF (DSF), ADRDSSU (DFDSS), et aux moniteurs qui permettent de modifier la mémoire;

- tenir disponible une duplication du disque système et du MASTERCAT (mettre à jour un membre SYSCATxx du NUCLEUS pour pouvoir faire IPL à partir de ce MASTERCAT dupliqué);

- prévoir la possibilité d'un démarrage sans certains composants: démarrage sans JES, sans RACF, etc., afin de pouvoir disposer au moins de TSO en cas de problème grave (pour édition et soumission de travaux de restauration); prévoir donc en PROCLIB et PARMLIB des procédures TSO et VTAM (et une procédure de LOGON) capables de démarrer en sous-système de MVS, indépendamment de JES;

- créer éventuellement un "mini-système" sur un seul disque, ne comportant que les fichiers nécessaires: LINKLIB, LPALIB, NUCLEUS, SVCLIB, CMDLIB (pour les commandes TSO), PARMLIB, PROCLIB, UADS, BRODCAST, fichiers de pagination, MASTERCAT, SPOOL de

JES. Ce mini-système peut être sauvé sur bande ou cassette pour être restauré en autonome (DF-DSS stand alone) en cas de problème. Moins de 200 cylindres de 3380 peuvent y être consacrés;

- installer le dump autonome (stand-alone);

- disposer d'une bande ou cassette contenant DF-DSS autonome pour toute restauration complète de disque. Il est commode de créer des bandes de sauvegardes sans label comportant en premier fichier l'utilitaire DF-DSS stand alone et en second fichier la sauvegarde du disque. L'IPL sur le dérouleur ou l'unité à cassettes amène l'utilitaire en mémoire et le positionnement sur le fichier suivant permet de déclencher immédiatement la restauration;

- disposer (pour les ordinateurs à IOCP) d'une bande sans label contenant le texte de l'IOCP. On peut alors recréer la configuration même si le disque intégré au processeur est défaillant.

Certaines mesures ne seront utiles qu'en dernier ressort, et peut être ne serviront jamais; cependant, elles sont tellement simples et peuvent être si salvatrices qu'on aurait tort de les négliger. Elles sont peut-être moins importantes dans un site multi-ordinateurs avec disques partagés, où l'on a plus de chance de disposer d'un système encore actif pour réparer ce qui doit l'être.

L'entreprise peut vouloir pousser sa démarche sécuritaire plus loin et monter un centre de secours destiné à prendre le relais en cas de sinistre grave. Cela impose une planification: il faut déterminer les applications et les logiciels qui sont absolument indispensables pour la bonne marche de l'entreprise; prévoir ensuite les sauvegardes à utiliser pour restaurer les fichiers vitaux (les autres, faute de temps et de moyens matériels, ne le seront pas). Les sauvegardes à utiliser doivent être disponibles, identifiables, et synchronisées entre elles pour reconstituer un environnement cohérent et utilisable, ce qui peut être parfois complexe à gérer; le plus simple en la matière est de sécuriser à intervalles réguliers tous les volumes critiques, ce qui donne un jeu de sauvegarde cohérent et dont le déphasage avec la situation avant le sinistre est connu. On peut aussi vouloir mettre en place un réseau de télécommunication de secours. Il faut dans tous les cas connaître la configuration matérielle du site de secours, à moins que celui-ci ne dispose de VM, qui peut rendre transparente l'arrivée du système sinistré. La restauration du système MVS s'effectue par DF-DSS en autonome, et l'emploi d'un mini-système est à recommander pour restaurer ensuite le système "réel" par DF-DSS sous MVS, qui est bien plus rapide que sa version stand-alone.

13. L'INSTALLATION D'UN SYSTEME MVS

A) La génération

L'opération de génération d'un système MVS (SYSGEN) consiste à créer un système opérationnel à partir des bibliothèques de distribution et de paramètres définis par le site. Il y a trois sortes de générations MVS:

- la génération complète (FULL SYSGEN) qui entraîne l'installation d'un nouveau système MVS;
- la génération d'E/S (IOGEN), utilisée lors de modifications dans la configuration des unités d'E/S;
- l'EDTGEN (eligible device table generation) qui prend en compte des changements dans les noms ésotériques d'unités (unitnames).

Les deux derniers types, qui sont des générations partielles, ont disparu en MVS-SP220 pour être remplacé par le MVSCP. Nous nous intéresserons uniquement à la FULL SYSGEN.

La génération se déroule en deux étapes:

- le STAGE1: les paramètres sont définis et assemblés. Ils permettent de définir la configuration, les méthodes d'accès souhaitées, la table des SVCs, les fichiers système et le disque système, etc. Au sortir du STAGE1 un JCL est créé;
- le STAGE2: le JCL créé par le STAGE1 est soumis. Il génère les bibliothèques système par copies, assemblages ou éditions des liens à partir des bibliothèques de distribution livrées par IBM.

La génération complète d'un MVS n'est nécessaire que pour une nouvelle installation. Une fois effectuée, elle ne sera plus refaite, sauf dans certains cas particuliers: besoin d'une méthode d'accès omise à l'installation ou d'autres modules non générés. Certaines PTFs ont besoin d'une génération pour être vraiment prises en compte (hold FULLGEN). Si une génération doit être refaite après installation, il faut veiller à ce que les PTFs appliquées aient aussi été acceptées, sans quoi les DLlBs ne reflètent plus le niveau précédent du système et le système généré sera d'un niveau inférieur au précédent.

La génération, procédé lourd et long, n'est pas le mode habituel d'installation des logiciels IBM; elle est réservée aux produits riches en fonctionnalités et pour lesquels un choix doit être effectué. Peu à peu les paramètres de génération de MVS sont d'ailleurs presque tous devenus obsolètes pour être remplacés par des options en SYS1.PARMLIB.

Passons en revue les paramètres de génération du STAGE1:

- AFFINITY: décrit la PPT (program properties table, voir chap. 6). Remplacée à partir de MVS/XA220 par le membre SCHEDxx de la Parmlib.
- CKPTREST: options du Checkpoint/Restart.
- CONSOLE: définit les consoles MVS. Remplacé par CONSOLxx en Parmlib, mais utile pour le MVSCP.
- CTRLPROG: définit les options fondamentales de MVS. Le paramètre ACRCODE=YES permet de prendre en compte les multi-processeurs (code pour "alternate CPU recovery"). Remplacé par IEASYSxx et CLOCKxx.
- DATAMGT: définit les méthodes d'accès supplémentaires à générer (par exemple ISAM, BTAM, etc.).
- DATASET: définit les fichiers du système.
- EDIT: donne les options de l'éditeur TSO.
- JES: permet de générer JES3.
- SCHEDULR: définit les sous-systèmes et le sous-système primaire.
- SVCTABLE: définit les SVCs actives. Remplacé par IEASVCxx.
- TSO: définit quelques paramètres marginaux de TSO.

Les paramètres IODEVICE, UNITNAME (et CHANNEL en MVS/SP) décrivent la configuration matérielle (ils sont remplacés par le MVSCP en version 220). Enfin GENERATE avec GENTYPE=ALL indique une génération complète et précise le volume système.

Dans les toutes dernières versions de MVS, seules les macros DATAMGT, DATASET et GENERATE sont supportées. Au cours des releases et des versions le paramétrage de génération a été peu à peu déporté vers la PARMLIB, d'emploi beaucoup plus souple.

B) L'installation

Les procédés d'installation de MVS sur un site 370/390 sont les suivants:

* le système "express": comme son nom l'indique, il est destiné à des clients pressés, qui ne disposent peut-être pas d'assez de temps ni de personnel pour installer MVS par leurs propres moyens ou qui installent MVS pour la première fois. Il s'agit d'un système créé chez le constructeur à partir des desiderata du client; des copies sur des bandes standard label ont été faites avec DF-DSS et après restauration sur le site on obtient un système opérationnel;

* le CBIPO (custom-built installation process offering), qui est la méthode standard pour installer un nouveau système. Le constructeur fournit un système à un niveau bien défini; ce niveau de service est intégré aux bibliothèques de distribution à partir desquelles sera généré le système cible. A la livraison le CBIPO comporte 3 volumes "logiques":

- la bande RIM (related installation material) qui contient la documentation d'installation et surtout des JCLs d'installation (bibliothèques IPO1);

- la bande DLIB, qui contient les bibliothèques de distribution et le CSI correspondant (il peut y avoir physiquement plusieurs bandes);

- la bande SMP (ou bande SERV) contient des PTFs d'un niveau supérieur à celui des DLIBs, ou des PTFs en erreur.

D'autres grands logiciels IBM sont distribués en CBIPO: NCP, CICS, DB2, IMS. Les sites qui ne disposent pas de système MVS peuvent obtenir un système jetable ("MVS driver") le temps de l'installation du CBIPO;

* le CBPDO (custom-built product delivery offering) est destiné à une remise à niveau de MVS, ou à l'installation de nouvelles fonctions, et non pas à l'installation complète du système. Le même volume logique contient la documentation, les produits, les PTFs. Les produits sont à un niveau donné, choisi par le client, les PTFs ultérieures sont livrées aussi, mais ne sont pas intégrées aux produits.

Le CBPDO semble plus adapté pour appliquer la maintenance ou effectuer certains changements de versions bénins; dans beaucoup de cas, le CBIPO peut être plus facile à installer avec les RIMs qui minimisent le travail, mais il produit un système entièrement neuf: il faut réinstaller les exits, les usermods (modifications système du site), les logiciels non inclus dans la commande, les logiciels non IBM, etc.

Le CBPDO, comme les bandes PUT, est créé à partir du profil du client. S'il ne s'agit que de mettre à niveau un logiciel, les autres possibilités sont l'application des bandes PUT adéquates ou d'une bande cumulative équivalente.

Les étapes d'installation de MVS par CBIPO sont les suivantes:

* création des fichiers système sur le futur disque système;

* création et copie à partir des bandes des fichiers RIM, qui contiendront les JCLs des étapes suivantes, avec principalement le fichier IPO1.INSTLIB;

* adaptation de ces JCLs au site par mise à jour des fichiers RIM;

* création du nouveau MASTERCAT où sont catalogués les fichiers système; les fichiers JES, PAGE et SMF sont aussi créés;

* éventuellement création de divers catalogues utilisateurs où seront définis les fichiers SMP et DLlBs;

* création et déchargement des fichiers DLlBs, qui vont servir à générer le système;

* création des fichiers SMP;

* réception des PTFs de la bande SERV.

Commence ensuite ce qu'lBM appelle le processus d'"IPOGEN":

* création du fichier SMP CSI cible; ce fichier est ensuite initialisé à partir du fichier CSI de distribution;

* assemblage du STAGE1 de la génération MVS; il en résulte un fichier IPO1.STAGE2;

* ce fichier n'est pas soumis directement comme dans le STAGE2 de la génération MVS; il est utilisé par le JCLIN de SMP pour mettre à jour les entrées de la zone SMP cible : ainsi le CSI cible connaît-il définitivement tous les éléments du nouveau système. Les logiciels non supportés par la génération ("non-sysgenables") pourront tout de même être installés, car le JCLIN les décrivant figure dans les fichiers SMP.

* la commande SMP GENERATE construit les JCLs d'installation définitive: un JCL réunit les travaux de copie depuis les DLlBs; l'autre réunit les étapes d'assemblage et édition des liens; autant de travaux sont créés que de bibliothèques système;

* la soumission de ces JCLs termine l'IPOGEN;

* la dernière étape est celle de POSTGEN ou d'adaptation au site (customization): création du texte d'lPL, du dump stand-alone, de la PARMLIB, la PROCLIB, etc. Enfin l'IPL permet de tester le nouveau système et les IVPs (installation verification procedures) vérifient le bon fonctionnement de chaque produit.

L'installation de MVS par le CBIPO nécessite en général au moins deux disques: I'un sera le disque système; il peut aussi contenir les fichiers RIM; I'autre est le disque de DLlBs, sur lequel peuvent être placés des fichiers de pagination ou le SPOOL de JES.

C) La migration VSE/MVS

La question de la migration du DOS vers l'OS se pose depuis les années 60 et reste toujours ouverte. La conversion DOS/VSE vers MVS coûte beaucoup de temps et de ressources, et les utilisateurs VSE (ils sont encore très nombreux) préfèrent garder ce système, qui est toujours supporté (la version 3 a été annoncée en 1986 et la version 4 en 1988), qu'lBM a amélioré dans un sens plus proche de MVS (depuis la version 2 en 1984), et qui bénéficie maintenant de l'architecture ESA.

VSE et MVS sont des systèmes hautement incompatibles, sauf en quelques points tels que les fichiers bande, les bases DL/1, les fichiers VSAM et catalogues VSAM (qui peuvent être accédés par l'un ou l'autre système sous certaines conditions - mais non partagés). Si le jeu d'instructions est le même, en revanche les appels superviseur sont complètement différents. Les principaux objets à convertir d'un système à l'autre sont:

* les programmes: les sources doivent être modifiés et recompilés. Il y a de nombreuses incompatibilités sur le plan des langages eux-mêmes. IBM fournit des aides à la conversion pour les programmes Cobol ou PL/1. Les programmes assembleur posent plus de problèmes (macro-instructions e1 conventions différentes). Enfin les programmes dont on ne dispose pas des sources doivent être ré-installés par le fournisseur ou réécrits;

* le JCL VSE doit être transformé en JCL MVS; il s'agit de langages de commande très différents. Le logiciel JCA (VSE JCL conversion aid) peut être employé;

* les fichiers VSE doivent être recréés en MVS (structures différentes). Par exemple les bases DL/1 doivent être déchargées en VSE pour être rechargées en IMS/VS sous MVS.

Le passage de VSE à MVS est un projet de longue haleine, qui doit être soigneusement préparé. Comme il s'agit de systèmes orientés tous deux vers le batch, les ressources nécessaires sont importantes, et on peut être amené à faire fonctionner la production un certain temps avec les deux systèmes en parallèle, pendant la conversion et après.

L'autre problème est celui de l'adaptation du personnel informatique à un nouveau système. Les standards sont différents en MVS, avec notamment l'importance des qualifiants des noms de fichiers catalogués. Le logiciel DITTO (data interfile transfer testing and operations) est remplacé par plusieurs utilitaires MVS (bien qu'il existe aussi en OS). La gestion du SPOOL avec JES diffère de celle de POWER; de même ICCF (interactive computing and control facility) diffère de TSO. Une formation à MVS et au JCL est très souvent nécessaire. VSE est destiné aux ordinateurs IBM de moyenne puissance, mais il est fréquent de le rencontrer sur de plus gros ordinateurs (en multiples exemplaires dans de nombreuses machines virtuelles VM); à l'évidence il est plus facile d'acquérir un ordinateur puissant que de faire l'effort de passer à un système d'exploitation plus performant mais complètement différent.

MVS est en général ressenti par le personnel venant du DOS/VSE comme un système bien plus riche en possibilités, mais aussi plus centralisé et plus rigoureux. Mais le passage à MVS est aussi un accès à un environnement plus ouvert, qui supporte les logiciels "hauts de gamme" d'lBM destinés aux grands systèmes (même si l'équivalent existe parfois en VSE) tels que NETVIEW, IMS, DB2, CICS, etc.

D) MVS et VM

1) DEUX SYSTEMES DIFFERENTS

VM a été conçu à l'origine pour être un hyperviseur destiné à supporter différents systèmes s'exécutant dans différentes "machines virtuelles". Le composant de base de VM, CP (control program), gère ces différentes machines (qui sont en fait des espaces-adresses analogues à ceux de MVS) en leur fournissant un environnement simulant une machine réelle: les machines disposent de registres virtuels, d'un PSW virtuel, de minidisques qui sont des parties de disques réels, et d'un espace de mémoire qu'elles voient comme étant leur mémoire réelle (mais qui est de la mémoire virtuelle paginable pour CP). CP agit en prenant le contrôle quand une machine émet des instructions privilégiées; en effet chaque machine virtuelle s'exécute sous VM en mode problème; le passage de la machine au mode privilégié provoque une interruption programme qui est traitée par CP.

Ce rôle d'hyperviseur est encore très utile dans le cas d'une migration de VSE à MVS, ou pour tester de nouvelles versions de système, par exemple MVS/370 et MVS/XA avec VM/XA (SF ou SP). Pour ce qui est du haut de gamme cependant, des dispositifs microcodés tels que le MDF d'Amdahl, MLPF de HDS ou PR/SM (processor resource/systems manager) d'lBM sur les 3090, qui permet de partager logiquement l'ordinateur en 7 machines, devraient rendre VM moins intéressant dans ce rôle.

Le point fort de VM découle plutôt de la possibilité de faire de l'informatique personnelle, avec CMS ("conversational monitoring system", à l'origine "Cambridge monitor system"). Une machine CMS correspond à un utilisateur sur un écran, disposant d'un certain espace disque et de logiciels, un peu comme s'il travaillait seul avec un micro-ordinateur (mais avec les ressources d'un grand système). CMS est souvent utilisé en infocentre pour l'utilisateur final, ou parfois pour les programmeurs DOS ou MVS qui le voient comme un environnement de développement et de test plus performant que TSO.

En revanche VM n'est pas adapté à la production et à la gestion de grosses masses de données, malgré l'existence de CMS batch ou de CICS-VM, et une personne habituée à MVS ou au DOS est assez surprise de

constater qu'il n'y a pas de catalogue en VM, pas de JCL, pas de notion de partage de fichier, etc. On peut sans doute "bricoler" des environnements de production en VM, mais telle n'est pas la destination de ce système. Aussi la question de la migration de VM à MVS ou l'inverse a peu de sens; s'agissant de systèmes bien spécialisés dans leur domaine, il y a très peu de chances qu'un jour IBM décide de ne plus supporter l'un ou l'autre et de contraindre ses clients à une migration aussi douloureuse que celle de DOS à MVS. MVS a cependant le privilège d'être le système de très grands sites de production, et IBM en est conscient: ainsi MVS-ESA est apparu bien avant VM-ESA.

Au reste, même si MVS et VM sont développés la plupart du temps par des laboratoires différents, IBM sait exploiter au mieux les réussites de l'un ou de l'autre. Ainsi le langage REXX (qui fait maintenant partie de la plate-forme SAA) fut développé d'abord en CMS avant de l'être en TSO et MVS. Inversement, SMP a inspiré le laboratoire VM pour créer un équivalent en VM: SES.

2) MVS SOUS VM

Quand on est amené à faire fonctionner MVS comme machine virtuelle de VM, les problèmes de performance doivent être examinés sérieusement. Les facteurs de dégradation identifiés sont:

* les deux niveaux de pagination (la mémoire "réelle" de MVS est paginée par VM);

* les E/S sont gérées par VM, de même que toutes les opérations privilégiées (privileged operation simulation); les CCWs notamment doivent être retraduits (les adresses des zones d'E/S sont des adresses "réelles" pour MVS, mais virtuelles pour VM);

* MVS n'est pas conscient qu'il s'exécute sous VM: sa notion d'écoulement du temps est fausse, elle ne correspond à un temps réel que lorsque VM dispatche MVS; ainsi SRM peut être conduit à prendre des décisions qui ne sont pas conformes à la réalité.

Diverses possibilités existent pour la machine virtuelle MVS sous VM:

* le fonctionnement en mode V = V (virtuel = virtuel), mode standard sous VM, le moins performant pour MVS; la fixation de la page 0 (qui dans tous les systèmes 370 est constamment utilisée) apporte un minimum de performance. Ce mode est suffisant pour une machine MVS uniquement destinée aux tests;

* V=V avec réservation de cadres de pages et emploi de tables particulières ("shadow tables") qui permettent de sauter un niveau de pagination: VM accède directement à la mémoire virtuelle d'un utilisateur MVS sans passer par les tables de MVS;

* le mode V = R (virtuel = réel) supprime complètement un niveau de pagination (et permet de se passer des "shadow tables"): VM ne pagine plus la mémoire "réelle" de MVS. On peut de plus dédier des canaux ou des unités à la machine MVS, qui sera seule à les gérer, et éliminer la traduction de CCWs. Une seule machine peut être en V= R, c'est la machine préférée (machine de production le plus souvent). Un autre mode intéressant en XA est le V= F (virtuel=fixé) qui, comme le V=R, élimine simplement la pagination de CP. Certains ordinateurs peuvent supporter en même temps jusqu'à 5 machines V=F et une V=R;

* certains dispositifs microcodés permettent de partager la machine en mode natif entre VM et MVS: il s'agit du PMA en 370 (preferred machine assist); le "SIE assist" de XA (start interpretive execution) permet d'intercepter les E/S ou les interruptions et de les traiter à la place de CP. Le V= R associé à ces dispositifs permet à MVS d'atteindre au moins 90 % des performances qu'on attendrait en mode natif. Enfin, avec VM-HPO associé au PMA ou avec VM/XA SP, il y a possibilité pour la machine MVS préférée de "survivre" même si VM connaît une défaillance irrémédiable.

E) Migration d'une version de MVS à une autre

Il s'agit du passage de MVS-SP1 à SP2 ou de SP2 à SP3. Ce genre de migration pose peu de problèmes dans la mesure où l'on bénéficie le plus souvent d'une compatibilité ascendante: les programmes fonctionnent sans changement dans une version de MVS plus récente que leur version d'origine, sauf cas particulier. Les changements de version de JES obligent très souvent à décharger puis à recharger le SPOOL. Pour certaines évolutions, il faut prévoir la coexistence de plusieurs versions, ce qui amène à appliquer sur la version la plus ancienne des PTFs de "tolérance": c'est le cas pour le passage à DFP2.2 (avec un plus grand choix de tailles de bloc physique possibles pour les fichiers VSAM) ou à DFP version 3, pour pouvoir partager les catalogues pré-existants; une certain compatibilité "descendante" est ainsi assurée.

Il faut aussi être conscient que le passage d'une version SP de MVS à une version XA ou ESA correspond à un changement d'architecture, à des performances et des possibilités bien différentes; il faut prendre garde aussi à certains "effets" imprévus: par exemple au fait que MVS-SP2 a besoin de plus de mémoire centrale que MVS-SP1, et SP3 que SP2 (au moins 1,5 Méga-octets de plus). MVS-SP4 réclamerait au moins 26 Méga-octets de mémoire réelle. De même MVS-ESA prend toute sa valeur avec la présence de la MAP.

Le passage de MVS-SP à MVS-XA est certainement le plus délicat, et il faut tenir compte des points suivants:

* la mémoire virtuelle a une structure différente; les segments représentent 1 Méga-octet de mémoire, et non plus 64K; pour cette raison la frontière basse de la CSA est alignée au Méga-octet immédiatement inférieur, ce qui risque de laisser une zone privée sous la "barre" plus petite que prévu (8 Méga-octets est la taille minimum de zone privée que l'on doit espérer en XA);

* la gestion des entrées-sorties est complètement revue et toutes les instructions d'entrée-sortie sont différentes. Il n'y a pas de changement visible pour le programmeur en langage "évolué", à part l'amélioration des performances;

* avec l'introduction de l'adressage à 31 bits, de nouvelles instructions de branchement apparaissent, qui permettent de changer le mode d'adressage (AMODE): BSM, BASSM. Les programmes de mode d'adressage 24 bits ne peuvent accéder aux données ou appeler des programmes résidant au-dessus des 16 Méga-octets. Les instructions de branchement BAL, BALR et l'instruction LA (load address) ont aussi un "comportement" différent;

* certains utilitaires, certaines unités périphériques ne sont plus supportés (les nouvelles versions de logiciel sont aussi une occasion d'épurer le système de produits obsolètes...).

Indiquons, pour information, comment IBM distingue les différentes évolutions de ses logiciels. Chaque niveau de maintenance est repéré par 3 composantes:

* la version (numéro supérieur ou égal à 1) : une nouvelle version apporte de nouvelles fonctions et modifie sensiblement le produit (de façon peut-être incompatible avec la version précédente);

* la release (numéro supérieur ou égal à 1): des corrections ("apar fixes") ont été apportées et éventuellement quelques fonctions ajoutées;

* le level, niveau de modification (supérieur ou égal à 0), correspond à un simple enrichissement du produit de PTFs apparues depuis le dernier level.

Par exemple, MVS/SP 3.1.3 désigne le troisième niveau de maintenance de la release 1 de MVS/SP version 3 (MVS/ESA).

F) PR/SM

PR/SM (ou son équivalent chez les constructeurs compatibles) est une fonction "hardware" qui permet de créer des "partitions" entre lesquelles on répartira les ressources de la machine (mémoire, MAP, canaux, processeurs). On parle parfois de PR/SM comme d'un "VM microcodé". Il s'agit en fait d'une fonction

programmée (LIC, licensed internal code chez IBM) moins proche du matériel que du logiciel (on peut d'ailleurs constater que PR/SM s'exécute avec une clé de protection 3), avec l'existence d'un véritable distributeur qui affecte les ressources (processeurs logiques) aux partitions selon certains algorithmes et en fonction du paramétrage. Avec PR/SM, l'ordinateur dispose d'un mode LPAR (logical partitioning) par contraste avec le mode "basic" d'un système natif.

Les partitions sont définies par l'opérateur à la console par des "frames" spéciales : on leur attribue une quantité de mémoire centrale, de MAP et un nombre de processeurs logiques. Les processeurs physiques ("réels") peuvent être "dédiés" à une partition ou partagés entre plusieurs partitions ; dans ce dernier cas, PR/SM allouera le processeur aux partitions selon leur importance (leur "poids") et pour un intervalle de temps précisé (time slice). Le poids et l'intervalle peuvent être changés dynamiquement. Le fait de dédier un processeur à une partition génère moins de déperdition tandis que le partage permet plus de souplesse et augmente la fiabilité en cas de panne d'un processeur physique. La répartition des canaux découle à l'origine de l'IOCP, par exemple :

```
CHPID  PATH=00,TYPE=BL,PARTITION=(MVSPROD,REC)
```

réserve le canal 00 à la partition MVSPROD ; le canal est reconfigurable (on pourra par commande MVS CONFIG l'affecter à une autre partition). Avec l'avènement des canaux "flottants" toute partition sera capable d'utiliser dynamiquement n'importe quel canal pour accéder aux unités. Les éléments de mémoire et de MAP peuvent aussi être reconfigurés plus ou moins aisément. La mémoire centrale est partagée entre les partitions un peu à la façon d'OS/MFT ; dans les réalisations d'IBM la partition "basse" (située à partir de l'adresse 0) est favorisée (favored) puisque la traduction d'adresse par PR/SM est alors triviale.

La distribution de la ressource processeur entre partitions, en mettant de côté le cas très simple des processeurs dédiés, repose sur certains paramètres. Le mode "event driven" (WAIT COMPLETION=NO) permet à PR/SM de profiter de l'inactivité d'une partition pour affecter la CPU à toute autre partition prioritaire qui en a besoin; le mode "time driven" (WAIT COMPLETION=YES) permet à une partition, selon son poids, de monopoliser la ressource même si elle n'en a plus besoin : on respecte ainsi de façon stricte les règles de poids, au risque parfois d'impacter les temps de réponse. Enfin la fonction de "capping" limite la consommation CPU d'une partition, au prix d'une certaine déperdition.

Comme sous VM, certains effets inattendus sont à prévoir pour MVS sous PR/SM, comme le risque d' I/O "elongation": l'E/S est terminée mais la partition MVS n'étant pas "dispatchée" ne se voit pas encore présenter l'interruption d'E/S ; une évaluation incorrecte par MVS et SRM de la consommation CPU, car le "wait involontaire", dû au fait que le distributeur de PR/SM a soustrait la ressource CPU à la partition, est ignoré par le système, MVS accumulant un "wait time" (LCCAWTIM) qui est la mesure des attentes volontaires, alors qu'il faudrait considérer le "partition dispatch time" pour mesurer la consommation CPU. D'autres phénomènes, tels que le "low utilization effect" (une déperdition CPU inattendue, due à PR/SM, quand les partitions consomment peu de ressource processeur), ont conduit IBM à mettre en évidence dans les comptes-rendus RMF toutes les déperditions engendrées par PR/SM.

Avant de mettre en place PR/SM, il faut tenir compte de contraintes telles qu'un certain overhead (7 à 10%), une quantité de mémoire de quelques MO prise en HSA par le code, et un changement de format des numéros de CPU (qui peut impacter certains logiciels non IBM qui se protègent des copies non autorisées par contrôle de la date et des numéros de série des processeurs). L'IOCP doit être remanié pour répartir les canaux entre les partitions. Les pupitreurs doivent faire connaissance avec de nouvelles "frames" pour gérer PR/SM et le découpage en partitions.

Finalement PR/SM (standard sur les ES/9000 et présent sur certains 3090) permet de faire fonctionner plusieurs systèmes sur une seule machine physique (ce qui peut amener certaines économies sur le coût des logiciels et des matériels) avec une souplesse convenable, pour une consommation minime.

14. MVS et son environnement

Un système d'exploitation n'est pas une fin en soi, mais bien le support de logiciels immédiatement utiles et rentables pour une entreprise, le but d'un service informatique étant, au travers d'un système performant, de traiter et de mettre à la disposition des personnes concernées les données qui lui sont confiées. Aussi il nous a paru intéressant de présenter quelques logiciels "phares", récents ou moins récents, correspondant à l'état de l'art et qui font partie intégrante de la sphère de MVS, même si cela déborde du cadre strict du système d'exploitation.

A) Les SGBDs

1) Fonctionnement général

La gestion d'entreprise débouche sur le traitement de grandes masses de données, organisées en fichiers ou en "bases de données". Une base de données (database) est un fichier centralisé contrôlé par un SGBD (système de gestion de base de données). Un SGBD (DB/DC) comprend une partie gestion des données (DB), qui consiste à maintenir l'intégrité des bases, gérer les accès et le partage, et une partie télétraitement (DC) qui met les données à la disposition des utilisateurs du réseau pour répondre à leurs requêtes (transactions).

Un SGBD présente les propriétés suivantes, qui sont un "plus" par rapport à ce que permettent les méthodes traditionnelles de traitement des fichiers:

* indépendance des données vis-à-vis de l'applicatif: la base a sa propre structure, indépendante des programmes qui la traitent. Les structures possibles découlent des modèles hiérarchiques (chemins d'accès en arbre avec des relations "1 pour 1" ou "1 pour n" dans un seul sens), en réseau (qui permet des relations "n pour n" ou "n pour 1") et relationnels (relations dynamiques suivant une logique ensembliste). Il y a souvent une nette séparation entre le langage de description de la base, celui qui permet l'accès à la base, et le langage de programmation courant.

* non-redondance des données et cohérence accrue ; on peut répertorier les données existantes (dictionnaire de données) et contrôler la cohérence des données nouvelles avec les anciennes (intégrité de référence).

* partage des données entre de nombreux utilisateurs, avec la possibilité de masquer une partie des données vis-à-vis de certains utilisateurs ("views" de DB2, PCB d'IMS).

Les problèmes pratiques qui surviennent dès qu'on met en commun des informations sont traités par les SGBDs de la façon suivante :

- intégrité des données : il faut garder les bases dans un état cohérent entre elles. Cette cohérence est une contrainte applicative opérant à un niveau qui dépasse celui de l'organisation physique. Si on imagine par exemple une transaction effectuant un virement d'un compte bancaire à un autre, une interruption de la transaction (abend, "plantage" du SGBD ou du système) risquerait de laisser débité le compte origine alors que le compte destinataire n'a pas encore été crédité. Si cela se produit, le SGBD doit être capable par la suite (et au plus vite) de restaurer le compte origine dans son état initial (backout de CICS ou IMS, rollback d'IDMS, etc.). Le même processus de recouvrement se produit au redémarrage d'un SGBD suite à un problème grave (emergency restart de CICS ou IMS, warmstart d'IDMS). Pour être capable de restituer la cohérence d'origine, le SGBD consigne les "images avant" et "images après" des données dans un fichier spécifique (active logs de DB2, dynamic log d'IMS, journaux de CICS ou IDMS), de manière à pouvoir réappliquer les images des données avant modification si la transaction a échoué. A l'inverse, une transaction qui s'exécute correctement du début à la fin conduit à un "point de synchronisation" (synchpoint, commit) à partir duquel le SGBD peut estimer que le traitement a abouti à un nouvel état des bases aussi cohérent que l'état précédent. Les fichiers log ou journaux peuvent également se révéler indispensables quand toute une période de mise à jour en "online" a

été "perdue" par suite de problèmes matériels et qu'on ne dispose plus que des sauvegardes de fin de nuit et des logs pour restaurer les bases.

Les problèmes de cohérence des données deviennent plus complexes quand plusieurs SGBD entrent en jeu (classiquement il s'agit de DB2 et d'IMS ou de DB2 et CICS). Les mises à jour sont alors coordonnées entre les différents systèmes par un protocole à deux temps (two-phase commit).

- le partage des données obéit à un système de verrous qui interdit toute lecture ou mise à jour concurrente. En DB2 le verrouillage se fait au niveau de l'enregistrement ou de la page de 4K ; en IMS le "program isolation" verrouille le DBR (database record) durant la mise à jour et jusqu'au point de synchro. Ce mécanisme, comparable aux enqueues de MVS, est efficace mais présente les mêmes risques : transactions retardées par "lock suspension" ; inter-blocage (deadlock) conduisant obligatoirement à l'abandon d'une des transactions au bout d'un certain temps d'attente (stall purge de CICS, timeout de DB2, etc.).

- contrôle d'accès et sécurité : une transaction étant souvent destinée à des utilisateurs qui ne sont pas des professionnels de l'informatique, la pratique d'un SGBD ne fait pas appel à des compétences techniques exceptionnelles. L'accès au SGBD, aux transactions et aux bases peut être contrôlé par un produit de sécurité tel que RACF ou un système d'autorisation propre (cas de DB2). L'administrateur de bases de données (DBA) peut de plus limiter l'accès aux données par le moyen de vues partielles des bases.

2) IMS

IMS est un puissant SGBD, écrit spécialement pour MVS, qui met en oeuvre l'organisation hiérarchique DL/1. Les transactions IMS sont traitées en parallèle dans différents espaces-adresses, les régions de message (MPP) ou plus rarement les régions batch (BMP). Des espaces-adresses sont dédiés à la région de contrôle, à DL/1 et à DBRC pour les reprises. IMS, par ses performances (avec notamment l'option Fastpath) et son adaptation au multi-traitement convient tout à fait aux grands sites qui réclament de gros "débits" transactionnels.

Au coeur d'IMS est DL/1 avec ses différentes méthodes d'accès : séquentielles (OSAM, SHSAM et SHISAM, HSAM et HISAM) et directes (HDAM, HIDAM). Les données sont organisées en DBRs (database records) ; un DBR est un ensemble de segments organisés hiérarchiquement au-dessous d'un segment racine. La souplesse requise, qui peut manquer dans un modèle hiérarchique pur, est apportée par les index secondaires (pour l'accès par d'autres segments que la racine) et les bases logiques (groupement de segments d'une ou plusieurs bases physiques).

Les programmes d'application accèdent aux segments en invoquant l'interface DL/1 (par CALLs directs ou par EXEC DLI en CICS). En online, les programmes répondent à des "messages", émis dès qu'un utilisateur frappe un code transaction, par accès aux bases et échange d'écrans gérés par MFS (message format services). Les fonctions MSC (multiple systems coupling) ou ISC (intersystem communication) permettent d'engager un traitement coopératif entre plusieurs IMS ou même entre IMS et CICS (avec ISC).

La "customisation" d'IMS fut longtemps une procédure lourde en comparaison par exemple avec la souplesse des "tables" CICS : il s'agit d'une "génération" qui oblige à définir les terminaux (le réseau doit être décrit à IMS sous la forme de LTERMs, logical terminals), les transactions, les programmes, les bases, les formats d'écran, la sécurité, etc. La plupart de ces objets peuvent à présent être créés ou modifiés en cours de fonctionnement d'IMS. Les bases sont décrites par des blocs DBD (database description) et la vue qu'en ont les programmes d'application, par des PSBs (program specification blocks).

3) CICS

CICS, beaucoup plus courant, est (chez IBM) l'alternative à IMS. Participant à SAA (il existe aussi en VSE, en VM, en OS/2), il supporte les fichiers VSAM, BDAM et les bases DL/1. Il est sans doute moins lourd qu'lMS, mais aussi moins puissant, puisque monotâche. Il dispose d'une capacité de multi-traitement grâce à l'option MRO (multiple region option) avec plusieurs CICS concurrents, dédiés à certaines applications ou à la gestion des terminaux (comme pour IMS, une fonction ISC permet à plusieurs CICS de communiquer entre eux).

Le distributeur de CICS donne le contrôle à la tâche de plus haute priorité, mais, à la différence de MVS, une tâche ne peut être interrompue qu'au moment où elle repasse le contrôle à CICS (lors d'une demande de service par EXEC CICS). Les programmes doivent être "quasi-réentrants", c'est-à-dire réutilisables en série entre deux appels à CICS (avec COBOL, chaque transaction exécutant un programme dispose d'une "working storage" qui lui est propre). L'opérateur contrôle CICS par un ensemble de transactions spécifiques (et non par commandes, comme avec IMS).

L'espace-adresse CICS contient à la fois le code de CICS, les tables de paramétrage, et un espace destiné aux transactions (DSA, dynamic storage area). Depuis peu, l'option SSSP permet de protéger plus efficacement le code CICS des écrasements de mémoire dus aux applications (un très vieux souci). Les modules spécialisés de CICS assument la gestion des tâches (KCP), des programmes (PCP), des terminaux (TCP), des fichiers (FCP), de la mémoire dynamique (SCP), des journaux (JCP) et des données temporaires ou transitoires. BMS (basic mapping support) fournit l'interface programme - terminal.

4) DB2

DB2 (Database 2) est un SGBD plus récent que CICS ou IMS, qui offre une organisation relationnelle des données, sous le format de "tables". Il se présente en MVS sous la forme de 3 ou 4 espaces-adresses: le maître (services systèmes, gestion des logs), le gestionnaire de bases de données, IRLM pour gérer les partages et éventuellement DDF (distributed data facility) pour traiter des requêtes éloignées et communiquer par VTAM avec d'autres DB2. DB2 diffère essentiellement d'lMS et de CICS en ce qu'on n'accède pas directement aux données qu'il gère: c'est un serveur que l'on invoque grâce aux interfaces TSO, IMS, CICS ou batch. Il est défini comme sous-système de MVS et fonctionne en clé 7 (comme IMS).

Les objets DB2 consistent en "tablespaces" (fichiers VSAM linéaires contenant les tables) regroupés à un niveau plus élevé en "databases" (définition plus applicative que physique). Une table est une combinaison de colonnes (champs de la table) et de lignes (enregistrements). Un champ peut être clé primaire ou clé "foreign" (correspondant à la clé primaire d'une autre table). Un jeu d'index stocké séparément dans la même database permet d'accéder plus vite aux lignes de la table. Tous les objets de DB2 sont eux-aussi décrits dans une database "catalogue" (DSNDB06), tandis qu'une database "directory" (DSNDB01) conserve les informations de logging et de fonctionnement courant. Le stockage physique sur les volumes disques est assuré par des "storage groups" (stogroups) qu'on peut rapprocher des storage groups de SMS.

Le langage qui sert à la fois à définir les objets DB2 et à traiter les tables est le SQL (Structured Query Language), langage non-procédural (on exprime ce qu'on veut obtenir sans indiquer le moyen de le réaliser, ceci étant à la charge de DB2), qui est devenu depuis plusieurs années un standard (normes ISO et ANSI) dont l'emploi dépasse largement le cadre du SGBD DB2. SQL peut être mis directement à contribution dans les programmes applicatifs, sous TSO avec un outil tel que SPUFI (SQL processor using file input) ou avec QMF (Query Management Facility), produit de reporting destiné aux non-professionnels.

Avec l'intégration à SAA, le standard SQL, les bases de données distribuées (ou "réparties") depuis la version 2.2, et la possibilité de faire communiquer (par sessions LU 6.2) plusieurs DB2 sur des sites différents, il est clair depuis plusieurs années que DB2 est devenu un produit-programme stratégique pour IBM, sinon pour ses clients (sans doute 5000 licences dans le monde, et plus de 300 en France).

B) La gestion des fichiers (DFHSM, DFSMS)

Longtemps en MVS la gestion des fichiers (création, déplacement, sauvegarde, etc.) fut manuelle ou peu automatisée. DFHSM apporta une contribution fondamentale à la gestion automatique de l'espace magnétique, en assurant une certaine quantité d'espace disponible sur les supports disques (organisés en pools de volumes primaires) et en prenant en charge certaines opérations de sauvegarde. Les fichiers inactifs sont "migrés" par HSM vers des supports magnétiques (disque ou cassette) prévus à cet effet (migration level 1 ou 2), le but étant de maintenir une certaine quantité d'espace libre sur les volumes primaires ; ils sont rapatriés sur ces volumes primaires dès qu'un utilisateur en a besoin (RECALL). HSM sauvegarde les fichiers (BACKUP) par

sauvegardes incrémentales (on peut ne sauver que les fichiers qui ont été modifiés depuis la dernière sauvegarde) ainsi que les volumes disques (DUMP) ; il dirige lui-même les restaurations de fichiers ou de volumes (RECOVER). La rotation des cassettes utilisées pour le backup doit être menée par l'opération de RECYCLE.

Cependant DFHSM ne s'intéresse principalement qu'aux volumes. Avec des disques tels que les 3390 modèles 3 (triple densité), qui peuvent contenir de très nombreux fichiers de nature différente, DFHSM manque quelque peu de discernement. On souhaiterait un traitement plus individualisé pour les fichiers.

Le concept SMS (system managed storage) apparu avec MVS/ESA (ou MVS/XA 223) et DFP v3 est un pas décisif vers une automatisation et une centralisation rendues nécessaires par l'accroissement des masses de données sur disque à gérer (souvent 30 % par an) et le besoin d'indépendance par rapport au support physique. Ce que cette gestion avait encore d'"artisanal" devient "industriel". Une fonction ABARS (aggregate backup and recovery support) complète HSM pour ce qui est des sauvegardes applicatives.

Avec DFSMS, toutes les étapes de la vie du fichier peuvent être contrôlées, depuis la création jusqu'à la destruction. Le JCL peut être allégé de paramètres "physiques" tels que DCB, UNIT, VOL et SPACE exprimé en pistes ou cylindres (voir aussi le chapitre 5,E). On peut créer des fichiers VSAM temporaires ou permanents par JCL sans passer par AMS. Le site est à même de contrôler l'allocation des fichiers et d'automatiser leur suivi par la mise en place de "profils types" (constructs) déterminant leurs caractéristiques physiques à la création (data class), les performances souhaitées découlant de l'emploi des contrôleurs à cache (storage class), la gestion des sauvegardes ou des migrations du côté de DFHSM (management class) et le "pool" de volumes sur lequel le fichier peut être créé (storage group). Parmi les 3 classes, seule la "storage class" est nécessaire pour un fichier SMS ; elle permet de doser l'utilisation du cache pour chaque fichier (dataset level caching, dynamic cache management) plutôt qu'au niveau volume comme auparavant. La "management class" affine le travail de HSM en précisant pour chaque fichier les règles de migration, de rétention et de sauvegarde. Les storage groups doivent être choisis peu nombreux ; ils réunissent des disques de même géométrie et de caractéristiques identiques pour SMS.

Les routines ACS (automatic class selection) attribuent les profils en fonction des nombreux paramètres mis à la disposition du gestionnaire d'espace magnétique, et la normalisation des noms de fichiers prend encore plus d'importance qu'auparavant. La mise en place de DFSMS s'accompagne de plusieurs changements dans la structure des VTOCs (bit SMS dans les DSCBs de type 1 ou 4, généralisation des VTOCs indexées) et des catalogues (profils notés dans les BCS et les VVDS). DFSMS s'appuie sur DFDSS (le "data mover") pour les transferts physiques, sur DFHSM pour les sauvegardes (backups ou dumps de disques), les migrations ou les restaurations, et sur ISMF, qui devient son interface privilégiée sous ISPF.

La migration à SMS peut s'effectuer avec DF/DSS au fil de l'eau, petit à petit, ou lors de l'arrivée de nouveaux disques, par copie de fichiers ou conversion en place (CONVERTV). Le principal problème (comme souvent en informatique) est un problème d'organisation : il faut arriver à se mettre d'accord avec les utilisateurs sur des règles de gestion des fichiers qui soient claires et définitives. Après coup, le bénéfice est important en gain d'espace et en automatisation. Ajoutons que SMS est un produit stratégique pour les centres informatiques, et n'en témoigne pas seulement le fait qu'IBM ait établi SMS comme un produit à part entière (appelé DFSMS/MVS) intégrant DFP (le minimum requis), DF/DSS, DFHSM (rebaptisés respectivement DFSMSdfp, DFSMSdss, DFSMShsm) ainsi que RMM (DFSMSrmm), composant qui gère les volumes bandes (bandothèque 3495) ; il y a aussi la possibilité pour SMS de centraliser la gestion de données se trouvant sur stations de travail ou réseaux locaux, avec DFDSM (distributed storage manager). L'ambition d'IBM est d'imposer SMS comme gestionnaire global des fichiers du centre informatique, quel que soit le système et le site de résidence.

C) TSO et ISPF

Dans des systèmes d'exploitation prévus à l'origine pour le traitement par lots le besoin d'interactivité est faible: il faut seulement pouvoir accéder à certains fichiers (fonction d'"édition"), puis lancer et suivre les travaux à exécuter par la machine. TSO (time sharing option), le "temps partagé" de MVS, existait déjà sous MVT... Ses fonctions de base sont assez limitées et se résument en quelques commandes simulant celles du JCL

(ALLOCATE, CALL, SUBMIT, etc.) et en un éditeur ligne à ligne, souvenir du temps où les terminaux n'étaient que des espèces de machines à écrire rudimentaires.

Chaque utilisateur TSO (informaticien ou du moins utilisateur averti) dispose d'un espace-adresse qui lui ouvre l'accès à toutes les fonctions standards de MVS. Le langage CLIST est un langage interprété prévu à l'origine pour exécuter en une seule fois de nombreuses commandes TSO (stockées dans un fichier) et qui a évolué pour devenir un langage de programmation capable d'une certaine interactivité. REXX (restructured extended executor language), annoncé en 1983 sous VM/SP, disponible sous CMS ainsi que TSO/E version 2 depuis 1988, est un langage de programmation universel bien plus puissant, presque entièrement portable (inclus dans la plate-forme SAA), et dont l'emploi en MVS peut être étendu à tous les espaces-adresses (TSO, batch, NETVIEW, etc.).

ISPF (interactive system productivity facility) est un gestionnaire de dialogue sous TSO, qui permet de réaliser des applications à base de panels (images d'écrans préformatées), messages, tables, squelettes (modèles paramétrables de JCL par exemple), commandes Clists et macros (Clists sous l'éditeur). Une interface ISPLINK (ISPEXEC) permet de mettre en oeuvre les principales fonctions d'ISPF par programme, Clist ou exec REXX.

ISPF/PDF (program development facility) constitue un jeu de dialogues sous ISPF offrant notamment un éditeur pleine page tout à fait correct. La majeure partie du temps passé aujourd'hui sous TSO par le programmeur l'est souvent sous ISPF/PDF, qui est un outil de développement irremplaçable, et dont la convivialité approche peu à peu celle de certains logiciels sur micro-ordinateur, avec dans les dernières versions le fenêtrage, les menus déroulants, les barres d'action, etc. Les principaux outils de l'informaticien travaillant sous MVS sont disponibles sous ISPF/PDF : ISMF, SDSF, SPUFI, RMF moniteur III, INFO/MVS, etc.

D) Les réseaux

Les réseaux de communications prennent de plus en plus d'importance, avec ce facteur nouveau depuis plusieurs années que représente le développement d'une informatique départementale et répartie destinée dans certains cas à contourner ou prolonger une informatique centralisée autour d'un unique ordinateur "host" arrivée à bout de souffle et dont la convivialité et la modularité sont loin d'égaler d'autres architectures récentes.

En MVS (de même en VM et VSE) le gestionnaire de réseau est VTAM (virtual telecommunication access method), qui, comme son nom l'indique, peut servir de méthode d'accès à un programme d'application, avec un ensemble de macro-instructions pour l'invoquer. Cependant VTAM est bien plus qu'une méthode d'accès du type ancien BTAM (basique, avec emploi de READ et WRITE) ou TCAM (queued, dédiée surtout à TSO), puisqu'il se présente comme un espace-adresse autonome. VTAM est une mise en oeuvre de l'architecture SNA (Systems Network Architecture) d'lBM, lancée en 1974. L'idée initiale de SNA était de fonder une base commune pour tous les produits de télécommunications d'IBM. Les limitations des méthodes d'accès QTAM, BTAM ou TCAM devaient être levées pour pouvoir gérer des terminaux de plus en plus intelligents et des réseaux modernes souvent interconnectés entre eux.

Avec VTAM et SNA, les terminaux gagnent en indépendance tant vis-à-vis du host que des applications. Identifiés par une adresse virtuelle, ils ne sont plus dédiés à un CICS, un IMS, un TSO, etc., mais ils accèdent à de multiples applications avec une certaine indépendance relativement au réseau tant du point de vue physique que logique (notion de LU, de session, couches spécialisées de SNA). VTAM contrôle les communications entre programmes d'application (CICS, TSO, IMS, etc.) et terminaux en utilisant les ressources du réseau pour répondre aux besoins de communication.

La gestion des terminaux (mis à part les terminaux locaux) est confiée dans les réseaux un tant soit peu importants à un contrôleur de communications 37XX (3705, 3725, 3745, etc.) et à son logiciel NCP (network control program), qui déchargent l'ordinateur central de la gestion des transmissions. NCP prend en charge la scrutation (polling), le contrôle des liaisons, l'adressage, la bufferisation et la reprise en cas d'erreur.

Dans des environnements complexes (multi-domaines ou multi-réseaux), un outil de contrôle et de suivi du réseau est indispensable: il s'agit de NETVIEW. L'ouverture au trafic non SNA et au monde non IBM (par exemple les Minitels) se fait au travers de fonctions spécialisées, telles que NPSI pour X25.

Une nouveauté récente est l'introduction du traitement coopératif avec l'APPC/MVS. Le but (idéal) du traitement coopératif est de rendre accessible n'importe quelle donnée du réseau depuis n'importe quelle plate-forme connectée, et ceci de façon transparente pour l'utilisateur. Le traitement coopératif se décline en plusieurs variantes moins ambitieuses que ce but ultime, depuis le simple "revamping" des applications (habillage reposant sur la convivialité des micros) jusqu'au mode client-serveur et à l'appel à distance (RPC, remote procedure call). Le traitement coopératif conduit à scinder une application en processus indépendants s'exécutant sur différents systèmes (MVS ou autres). L'APPC/MVS se place au niveau de la couche "services de présentation" de SNA et supporte l'interface SAA de communication CPI-C, le protocole LU 6.2, interface d'égal à égal (peer-to-peer) entre applications, mais aussi une interface propre à MVS, avec un scheduler (espace-adresse ASCH) qui distribue les processus coopératifs à d'autres espaces-adresses subordonnés.

Les réseaux jouent de plus en plus un rôle de fédérateurs des systèmes informatiques, pour faire communiquer entre eux des systèmes dissemblables ou des ordinateurs d'architectures différentes. On passe peu à peu d'une structure verticale hiérarchique (celle des débuts de SNA) vers une structure de réseaux maillés accueillant les systèmes départementaux, les stations de travail, les LAN, les micros. IBM va dans cette direction, avec l'évolution vers la LU 6.2, avec SAA, garant de la portabilité et de la standardisation des applications. Dans ce contexte, MVS aura toujours sa place, celle d'un serveur puissant, centralisant une grande masse de données et assurant l'axe vertical du réseau.

ANNEXES

1) Exemple 1 : programme utilisant le cross-memory

2) Exemple 2 : programme utilisant l'interface sous-systŠme

3) Exemple 3 : programme avec SCHEDULE de SRB

4) Exemple 4 : programme effectuant des EXCPs

5) Exemple 5 : programme invoquant la m,thode d'accŠs BSAM pour un PDS

6) Exemple 6 : programme cr,ant un dataspace ESA

```
* BUT DU PROGRAMME : AFFICHER LA DERNIERE COMMANDE ENTREE A LA CONSOLE
* PAR LE PUPITREUR ; IL FAUT POUR CELA CHERCHER L'INFORMATION DANS
* L'ESPACE-ADRESSE MVS "CONSOLE" PAR CROSS-MEMORY
*     (RDCM ET TDCM SONT DANS LA LSQA DE L'ESPACE-ADRESSE 'CONSOLE')
* CVT -> UCM BASE -> UCM PREFIX -> UCME MASTER CONSOLE -> RDCM -> TDCM
EXEMPLE1 CSECT
         SAVE (14,12),,EXEMPLE1-&SYSDATE-&SYSTIME
         LR   R12,R15          R12 = R15 = ADRESSE POINT ENTREE
         USING EXEMPLE1,R12    R12 EST LE REGISTRE DE BASE DE LA CSECT
         LR   R9,R13           SAUVEGARDER LE R13 INITIAL
         LA   R13,SAVE         MAINTENANT R13 POINTE SUR NOTRE SAVE
         ST   R13,8(R9)        ADRESSE DE NOTRE SAVE MISE CHEZ APPELANT
         ST   R9,4(R13)        ADRESSE SAVE APPELANT MISE CHEZ NOUS
* ACCES … LA CVT
         L    R1,CVTPTR         PRENDRE L'ADRESSE DE LA CVT
         USING CVT,R1           ADRESSER LA CVT
         L    R1,CVTCUCB        POINTER VERS UCM BASE
* ACCES … L'UCM BASE
         DROP R1                R1 N'EST PLUS UTILISE POUR L'ADRESSAGE
         USING UCM,R1           ADRESSER UCM BASE
         LH   R2,UCMCTID        PRENDRE ASID ADDRESS-SPACE CONSOLE
         ST   R2,ASIDF          LE STOCKER CHEZ NOUS
         DROP R1                R1 N'EST PLUS UTILISE POUR L'ADRESSAGE
* ACCES … L'UCM PREFIX
         SH   R1,=H'4'          RECULER DE 4 OCTETS
         L    R1,0(R1)          POUR PRENDRE ADRESSE UCM MCS PREFIX
         L    R1,0(R1)          PRENDRE ADRESSE UCM MASTERCONS UCMMCENT
* ACCES … L'UCME DE LA MASTER (UCM INDIVIDUAL DEVICE ENTRY MAP)
         USING UCMECB,R1        ADRESSER L'UCME
         L    R4,UCMXB          PRENDRE ADRESSE DU RDCM
         DROP R1                R1 N'EST PLUS UTILISE POUR L'ADRESSAGE
* ACCES AU RDCM QUI EST DANS L'A.S. "CONSOLE"
         MODESET KEY=ZERO,MODE=SUP   PASSER EN MODE SUPERVISEUR CLE 0
         ESAR R2                OBTENIR L'ASID DE NOTRE ESPACE-ADRESSE
         ST   R2,NOTRASID       LE SAUVER
         LA   R2,1              AUTORISATION INDEX DE 1 (TOUT PERMIS)
         AXSET AX=(2)           AUTORISER NOTRE ESPACE-ADRESSE
         L    R2,ASIDF          PRENDRE L'ASID DE L'ESPACE-ADRESSE VIS,
         SSAR R2                L'ETABLIR COMME ESPACE-ADR. SECONDAIRE
* ACCES AUX DONNEES DE L'AUTRE ESPACE-ADRESSE
         XR   R2,R2             R2 = 0
         LA   R2,4              4 OCTETS A MOUVEMENTER
         XR   R1,R1             R1 = 0
         MVCP ADDR1(R2),0(R4),R1    PRENDRE ADRESSE DU TDCM
* ACCES AU TDCM DANS L'A.S. CONSOLE
         L    R5,ADDR1          ADRESSE TDCM DANS A.S. CONSOLE
```

```
         LA   R2,128            LONGUEUR DE LA COMMANDE
         XR   R1,R1             R1 = 0
         MVCP LASTCMD(R2),640(R5),R1    COPIER  DERNIERE COMMANDE
* SORTIR DU MODE CROSS-MEMORY
         L    R2,NOTRASID       PRENDRE ASID DE NOTRE ESPACE-ADRESSE
         SSAR R2
         XR   R2,R2             R2 = 0
         AXSET AX=(2)
         MODESET KEY=NZERO,MODE=PROB      REPASSER EN MODE PROBLEME
* AFFICHER LES 30 PREMIERS CARACTERES
         MVC  WTO1+8(30),LASTCMD  INDIQUER LA ZONE A AFFICHER
WTO1     WTO  '                                        ',ROUTCDE=11
FIN      L       R13,4(R13)     REPRENDRE ADRESSE SAVE DE L'APPELANT
         RETURN  (14,12),T,RC=0
SAVE     DS   18F    18 MOTS DE SAVE AREA
NOTRASID DC   F'0'   NUMERO DE NOTRE ESPACE-ADRESSE
ASIDF    DC   F'0'   NUMERO D'ESPACE-ADRESSE VISE
ADDR1    DC   F'0'   ADRESSE TDCM
LASTCMD  DS   CL128  DERNIERE COMMANDE PASSEE PAR LE PUPITREUR
         CVT   DSECT=YES
         IEECUCM
         IEECDCM
         END


*
* EXEMPLE D'APPEL DE L'INTERFACE SOUS-SYSTEME
* LISTER TOUS LES JOBS COMMENCANT PAR LE NOM D'UN USER
*
         IEFJESCT
         CVT  DSECT=YES
EXEMPLE2 CSECT
         SAVE (14,12),,EXEMPLE2-&SYSDATE-&SYSTIME
         LR   R12,R15           R12 = R15 = ADRESSE POINT ENTREE
         USING EXEMPLE2,R12     R12 EST LE REGISTRE DE BASE DE LA CSECT
         LR   R9,R13            SAUVEGARDER LE R13 INITIAL
         LA   R13,SAVE          MAINTENANT R13 POINTE SUR NOTRE SAVE
         ST   R13,8(R9)         ADRESSE DE NOTRE SAVE MISE CHEZ APPELANT
         ST   R9,4(R13)         ADRESSE SAVE APPELANT MISE CHEZ NOUS
* PRENDRE LE PARAMETRE PASSE AU PROGRAMME : EXEC PGM=EXEMPLE2,PARM=ZZZZ
         L    R1,0(R1)
         LH   R2,0(R1)          PRENDRE LA LONGUEUR DU PARAMETRE
         BCTR R2,0              R2 - 1 POUR INSTRUCTION EXECUTE (MOVE)
         EX   R2,MOVE           FAIRE UN MOVE DE LONGUEUR VARIABLE
*
         MODESET MODE=SUP,KEY=ZERO
         LA    R1,PARAM         ADRESSER LE PARAMETRE
         OI    PARAM,X'80'      SIGNER L'ADRESSE
         IEFSSREQ               APPEL DU SSI
         ST    R15,RETCODE      STOCKER LE CODE RETOUR
         MODESET MODE=PROB,KEY=NZERO
*
         L       R13,4(R13)
         RETURN  (14,12),T,RC=0
*
MOVE     MVC  SSCSJOBN(1),2(R1)   MOVE USERID INDIQUE EN PARAMETRE
RETCODE  DC  F'0'
PARAM    DC  A(SSOB)
*
SSOB     DS  0F
SSOBID   DC  C'SSOB'   IDENTIFIEUR DU BLOC DE CONTROLE SSOB
SSOBLEN  DC  H'20'     LONGUEUR DU SSOB
SSOBFUNC DC  H'3'      FONCTION DEMANDEE : STATUS
SSOBSSIB DC  F'0'      ADRESSE DE SSIB = 0
```

```
SSOBRETN DC  F'0'      CODE RETOUR SOUS-SYSTEME
* RC=4  JOBNAME NOT FOUND     RC=8   INVALID JOBNAME/JOBID COMBINAISON
* RC=12 JOB NOT CANCELLED     RC=16  ARRAY TOO SMALL
SSOBINDV DC  A(SSCS)   ADRESSE EXTENSION
         DC  D'0'      2 MOTS DE TRAVAIL SSI
*
SSCS     DS  0F        EXTENSION DU SSOB POUR FONCTION CANCEL/STATUS
SSCSLEN  DC  H'184'    LONGUEUR DU SSCS
SSCSFLGS DC  X'80'     FLAG : USERID IS IN JOBNAME FIELD
SSCSULEN DC  X'07'     LONGUEUR DU USERID
SSCSJOBN DC  CL8' '    JOBNAME
SSCSJOBI DC  CL8' '    JOBID
SSCSDIMP DC  H'160'    TAILLE DE TABLE MISE A DISPOSITION DU SSI
SSCSDIMR DC  H'0'      POSITIONNE PAR SSI POUR DIRE SI TAILLE SUFFIT
SSCSARAY DS  0CL160    16*10
ARAY1    DS  CL80
ARAY2    DS  CL80
SAVE     DS  18F
         LTORG
         END
```

```
* CE PGM REND UN ESP.-ADR. NON SWAPPABLE (SCHEDULE DE SRB, APPEL A SRM)
         CVT   DSECT=YES
         IHASRB
EXEMPLE3 CSECT
         SAVE (14,12),,EXEMPLE3-&SYSDATE-&SYSTIME
         LR   R12,R15          R12 = R15 = ADRESSE POINT ENTREE
         USING EXEMPLE3,R12    R12 EST LE REGISTRE DE BASE DE LA CSECT
         LR   R9,R13           SAUVEGARDER LE R13 INITIAL
         LA   R13,SAVE         MAINTENANT R13 POINTE SUR NOTRE SAVE
         ST   R13,8(R9)        ADRESSE DE NOTRE SAVE MISE CHEZ APPELANT
         ST   R9,4(R13)        ADRESSE SAVE APPELANT MISE CHEZ NOUS
* LE PROGRAMME  RECOIT DANS R1 LE NUMERO D'ASID DE L'ESPACE-ADRESSE
         ST   R1,THEASCB       STOCKER L'ASID NUMBER
         USING SRBSECT,R5      ADRESSER LE SRB, COMMENCER A LE PREPARER
         LA   R5,MYSRB
         XC   MYSRB,MYSRB      MISE DU SRB A ZERO BINAIRE
         MVC  SRBID,=CL4'SRB'  ACRONYME
         MVC  SRBASCB,THEASCB  ASCB DE L'A.S. VISE POUR EXEC SRB
* ACQUISITION UNE PAGE EN SQA  (POUR SRB ET ROUTINE SRB)
         MODESET MODE=SUP,KEY=ZERO         PASSAGE MODE SUPERVISEUR
         GETMAIN  RC,LV=4096,BNDRY=PAGE,SP=226   GETMAIN EN SQA
         LR   R8,R1            ADRESSE DE LA ZONE EN SQA
         ST   R8,ADDSRB        LA SAUVEGARDER
         LA   R1,(ENTREE-PAGE) LONGUEUR DU SRB
         AR   R1,R8            ADRESSE ROUTINE SRB
         ST   R1,SRBEP         ENTRY POINT
         NI   SRBEP,B'01111111'   1ER BIT FORCE A ZERO (AMODE 24)
* MOVE DU SRB ET DE LA ROUTINE SRB EN SQA
         L    R9,=F'4096'      LONGUEUR DE ZONE A MOUVEMENTER
         LR   R11,R9           EGALE LONGUEUR ZONE EMETTRICE
         LA   R10,PAGE         ADRESSE ZONE ORIGINE
         MVCL R8,R10           'MOVE CHARACTERS' LONG : MOVE CODE + SRB
         L    R8,ADDSRB        RESTAURER R8
* ENVOI DU SRB AU DISPATCHER
         SCHEDULE SRB=(8),SCOPE=GLOBAL,LLOCK=YES,FRR=NO
         MODESET MODE=PROB,KEY=NZERO       RETOUR MODE PROBLEME
FIN      L       R13,4(R13)
         RETURN  (14,12),T,RC=0
ADDSRB   DS   F               ADRESSE DU SRB EN SQA
THEASCB  DS   F               ASCB DE L'A.S. A METTRE NON SWAPPABLE
```

```
PAGE     DS   CL4096           DEFINIR L'IMAGE SRB + ROUTINE SUR 1 PAGE
         ORG  PAGE
MYSRB    DS   CL(SRBSIZE)      ********   S  R  B   *******
ENTREE   DS   0F               ** R O U T I N E    S R B **
* A L'ENTREE DANS LA ROUTINE SRB, LES REGISTRES SONT UTILISES AINSI :
* R0 = ADRESSE DU SRB, R1 = USER DATA, R14 = RETURN ADD, R15 = ENTREE
         USING ENTREE,R7       R7 REGISTRE DE BASE
         LR   R6,R0            SAUVER R0
         LR   R7,R15           SAUVER ADRESSE D'ENTREE DE LA ROUTINE
         LR   R8,R14           SAUVER ADRESSE DE RETOUR
         XR   R0,R0            R0 = 0
         ICM  R0,3,CODE        CODE POUR LE SYSEVENT DE TRANSWAP
         XR   R1,R1            R1 = 0  (PAS DE ECB)
         L    R15,CVTPTR       ADRESSE DE LA CVT
         L    R15,CVTOPTE-CVT(,R15)   SRM ENTRY
* APPEL SRM
         LA   R13,SAVEAREA     SAVE AREA DE 72 OCTETS POUR SYSEVENT
         BALR R14,R15          ON INVOQUE L'INTERFACE SRM
         FREEMAIN R,LV=4096,A=(6),BRANCH=YES,SP=226  LIBERER MEMOIRE
         BR   R8               RETOUR APPELANT
SAVEAREA DS   18F              SAVE AREA POUR SRM
CODE     DC   X'000E'          TRANSWAP SYSEVENT
LONGROUT EQU  *-ENTREE         LONGUEUR DE LA ROUTINE SRB
         ORG
         LTORG
         END



*  CE PROGRAMME LIT UN BLOC D'UN FICHIER BANDE (EXCP)
EXEMPLE4 CSECT
         SAVE (14,12),,EXEMPLE4-&SYSDATE-&SYSTIME
         LR   R12,R15          R12 = R15 = ADRESSE POINT ENTREE
         USING EXEMPLE4,R12    R12 EST LE REGISTRE DE BASE DE LA CSECT
         LR   R9,R13           SAUVEGARDER LE R13 INITIAL
         LA   R13,SAVE         MAINTENANT R13 POINTE SUR NOTRE SAVE
         ST   R13,8(R9)        ADRESSE DE NOTRE SAVE MISE CHEZ APPELANT
         ST   R9,4(R13)        ADRESSE SAVE APPELANT MISE CHEZ NOUS
* OBTENIR UNE ZONE MEMOIRE DE LONGUEUR = BLKSIZE (AU MOINS)
         LH   R3,MAXCNT        NOMBRE D'OCTETS A OBTENIR
         GETMAIN  RU,LV=(3)    NB: ON NE FERA PAS DE FREEMAIN
         LR   R5,R1            PRENDRE ADRESSE DE LA ZONE OBTENUE
         ST   R5,CCW1          DATA ADDRESS
         MVI  CCW1,X'02'       COMMAND CODE : LECTURE
         OPEN (TAPEIN,(INPUT)) OUVRIR LE FICHIER
         MVC  CCW1CNT,MAXCNT   MAXIMUM A TRANSFERER
         EXCP IOB1             DECLENCHER L'EXCP
         WAIT ECB=ECB          ATTENDRE LA FIN DE L'OPERATION
         TM   ECB,X'7F'        TESTER SI IO-ERROR
         BO   NOERROR          PAS D'ERREUR, ON CONTINUE
         ABEND 99,DUMP         SI ERREUR, ON FAIT ABEND
NOERROR  DS   0H
         CLOSE (TAPEIN)
* NOMBRE D'OCTETS TRANSFERES = X'7FFF' MOINS LE "RESIDUAL COUNT"
         LH   R15,MAXCNT       R15 = X'7FFF'
         LH   R2,IOBCSWBC      R2 = RESIDUAL BYTE COUNT DU CSW
         SR   R15,R2           MAX - RESIDUAL BYTE COUNT DU CSW
RET      L    R13,4(R13)
         RETURN  (14,12),T,RC=(15)   CODE RETOUR DANS LE REGISTRE 15
*
TAPEIN   DCB  MACRF=(E),DDNAME=TAPEIN,DSORG=PS
ECB      DC   F'0'
BIDON    DS   0D
CCW1     CCW  X'02',BIDON,X'20',80   LECTURE BANDE/ZONE POINTEE PAR R5
```

```
         ORG  CCW1             ** CCW DE FORMAT 0 **
CCW1CMD  DS   CL1              CCW - COMMAND CODE
CCW1ADDR DS   CL3              CCW - DATA ADDRESS
CCW1FLAG DS   CL1              CCW - FLAGS
CCW1RES  DS   CL1              CCW - INUTILISE
CCW1CNT  DS   H                CCW - BYTE COUNT
         ORG
         CNOP 0,4
IOB1     DS   0CL32            ** IOB **
IOBFLAG1 DS   CL1
IOBFLAG2 DS   CL1
IOBSENS0 DS   CL1
IOBSENS1 DS   CL1
IOBECB   DC   A(ECB)           ADRESSE DE L'ECB
IOBFLAG3 DS   CL1
IOBCSW   DS   0CL7             CHANNEL STATUS WORD (7 DERNIERS OCTETS)
         DS   CL5
IOBCSWBC DS   CL2              CSW : RESIDUAL BYTE COUNT
IOBSTART DC   A(CCW1)          ADRESSE DU CCW
IOBDCB   DC   A(TAPEIN)        ADRESSE DU DCB
         DS   CL12             LE RESTE DE L'IOB
MAXCNT   DC   H'32767'         MAXIMUM BLOCKSIZE POSSIBLE SUR 2 OCTETS
SAVE     DS   18F
         LTORG
         END
```

```
* CE PROGRAMME LIT UN FICHIER PARTITIONNE MEMBRE PAR MEMBRE
* IL AFFICHE (SOUS TSO) LE NOM DU MEMBRE, LA LONGUEUR
* ET LES PREMIERS OCTETS DE CHAQUE BLOC DE MEMBRE
* LE PDS DOIT ETRE ALLOUE EN SYSLIB
EXEMPLE5 CSECT
         SAVE (14,12),,EXEMPLE5-&SYSDATE-&SYSTIME
         LR    R12,R15          R12 = R15 = ADRESSE POINT ENTREE
         USING EXEMPLE5,R12     R12 EST LE REGISTRE DE BASE DE LA CSECT
         LR    R9,R13           SAUVEGARDER LE R13 INITIAL
         LA    R13,SAVE         MAINTENANT R13 POINTE SUR NOTRE SAVE
         ST    R13,8(R9)        ADRESSE NOTRE SAVE MISE CHEZ APPELANT
         ST    R9,4(R13)        ADRESSE SAVE APPELANT MISE CHEZ NOUS
         USING IHADCB,R5
         LA    R5,SYSLIB        DCB DU PDS
         OPEN  (SYSLIB,,LIB)    OUVRIR LES FICHIERS
* OBTENIR DYNAMIQUEMENT UNE ZONE DE MEMOIRE (POUR UN BLOC D'UN MEMBRE)
         L     R6,LONG
         GETMAIN RU,LV=(6)      DEMANDE DE MEMOIRE
         ST    R1,BUFAD         STOCKER ADRESSE DE LA ZONE OBTENUE
NEXTMBR  DS    0H
         XC    TTRN,TTRN
         CLC   LONGBLK,OFFSET   COMPARER TOTAL AVEC COMPTEUR
         BH    TRAITER          BLOC DE DIRECTORY DEJA LU, L'EXPLOITER
* LIRE UN BLOC DE DIRECTORY
         GET   LIB,DIRZONE      LECTURE
         LA    R1,DIRZONE       POINTER DEBUT ZONE
         MVC   LONGBLK,0(1)     MOVE LONGUEUR TOTALE UTILE DU BLOC
         MVC   OFFSET,=H'2'     OFFSET = 2 OCTETS A CAUSE COUNT
* EXPLOITER UNE ENTREE DU BLOC DE DIRECTORY
TRAITER  DS    0H               EXPLOITATION BLOC DE DIRECTORY
         LA    R1,DIRZONE       POINTER DEBUT ZONE
         LH    R0,OFFSET        CHARGER COMPTEUR
         AR    R1,R0            ACCEDER MEMBRE SUIVANT
         MVC   MBRNAME(8),0(R1) ** NOM DU MEMBRE **
         MVC   TTRN(3),8(R1)    ** TTR DU MEMBRE **
         XR    R0,R0            REG 0 = 0
```

```
         IC    R0,11(R1)         CHARGER LONGUEUR USER DATA EN 1/2 MOTS
         N     R0,=X'0000001F'   GARDER LES 5 DERNIERS BITS
         SLL   R0,1              R0*2 = NOMBRE D'OCTETS DE USER DATA
         LR    R1,R0             LONGUEUR DES USER DATA
         LA    R1,12(0,R1)       +12 OCTETS : NAME, TTR, LONGUEUR UDATA
         LH    R0,OFFSET         CHARGER COMPTEUR ANCIEN
         AR    R1,R0             AJOUTER OFFSET
         STH   R1,OFFSET         STOCKER COMPTEUR NOUVEAU
         CLC   FF,MBRNAME
         BE    CLOSE
         POINT SYSLIB,TTRN          POSITIONNEMENT SUR LE MEMBRE
         L     R6,BUFAD             CHARGER ADRESSE BUFFER PREVU
* LIRE UN BLOC DU MEMBRE : R6 POINTE SUR LE BLOC
NEXTBLK  READ  DECB1,SF,SYSLIB,(6),'S'   LIRE UN BLOC
         CHECK DECB1               ATTENDRE LA FIN DE L'E/S
         LH    R1,DCBBLKSI         BLOCKSIZE
         L     R2,DECB1+16         IOB ADDRESS
         SH    R1,14(R2)           BLKSIZE-RESIDUAL COUNT=LONGUEUR
         ST    R1,LONGBLOC         LONGUEUR UTILE DE CE BLOC
         CVD   R1,DBLEWORD         CONVERTIR LONGUEUR EN PACKE
         UNPK  LONGBL,DBLEWORD     PUIS EN DECIMAL ZONE
         OI    LONGBL+L'LONGBL-1,X'F0'    NORMALISER LE SIGNE
         MVC   DEBUTBL,0(R6)       COPIER LE DEBUT DU BLOC POUR INFO
         TPUT  MESSAGE,L'MESSAGE   AFFICHER A L'ECRAN TSO LES INFOS
         B     NEXTBLK             LIRE LE BLOC SUIVANT
FINMEMBR DS    0H
         B     NEXTMBR             PASSER AU MEMBRE SUIVANT
CLOSE    DS    0H
         CLOSE (SYSLIB)
         CLOSE (LIB)
FIN      L     R13,4(R13)
         RETURN  (14,12),T,RC=0
DBLEWORD DS    D
SAVE     DS    18F
TTRN     DS    CL4
LONG     DC    F'32767'          LONGUEUR MAXIMALE D'UN BLOC DE FICHIER
LONGBLOC DC    F'0'              LONGUEUR UTILE DU BLOC DE MEMBRE
BUFAD    DC    F'0'
OFFSET   DC    H'99'             OFFSET MBR DEPUIS DEBUT BLOC DIRECTORY
LONGBLK  DC    H'0'              LONGUEUR DU BLOC DE DIRECTORY (BINAIRE)
DIRZONE  DS    CL256             ZONE IO BLOC DIRECTORY
LIB      DCB   DDNAME=SYSLIB,MACRF=GM,BLKSIZE=256,                    X
               RECFM=F,DSORG=PS,LRECL=256
SYSLIB   DCB   DDNAME=SYSLIB,MACRF=R,DSORG=PO,EODAD=FINMEMBR
FF       DC    X'FFFFFFFFFFFFFFFF'  MARQUE DE FIN DE DIRECTORY
*
MESSAGE  DS    0CL70
         DC    C'MEMBRE '
MBRNAME  DS    CL8               NOM DU MEMBRE
         DC    C'  BLOC DE LONGUEUR '
LONGBL   DC    CL5' '            LONGUEUR DU BLOC DE MEMBRE
         DC    C' '              BLANC SEPARATEUR
DEBUTBL  DC    CL(70-*+MESSAGE)' '   DEBUT DONNEES DU BLOC
*
         LTORG
         DCBD  DSORG=PS
         END



* CE PROGRAMME CREE UN DATASPACE ESA, Y ACCEDE, PUIS LE DETRUIT
EXEMPLE6 CSECT
EXEMPLE6 AMODE 31
```

```
EXEMPLE6 RMODE ANY
         SAVE (14,12),,EXEMPLE6-&SYSDATE-&SYSTIME
         LR   R12,R15          R12 = R15 = ADRESSE POINT ENTREE
         USING EXEMPLE6,R12    R12 EST LE REGISTRE DE BASE DE LA CSECT
         LR   R9,R13           SAUVEGARDER LE R13 INITIAL
         LA   R13,SAVE         MAINTENANT R13 POINTE SUR NOTRE SAVE
         ST   R13,8(R9)        ADRESSE DE NOTRE SAVE MISE CHEZ APPELANT
         ST   R9,4(R13)        ADRESSE SAVE APPELANT MISE CHEZ NOUS
* CREER UN DATA SPACE
         DSPSERV   CREATE,         CREATION D'UN DATASPACE             *
               NAME=NOM,           NOM DU DATASPACE                    *
               BLOCKS=TAILLE,      TAILLE EN NOMBRE DE BLOCS           *
               ORIGIN=ORIGINE,     OU COMMENCE-T-IL (A RETOURNER)      *
               SCOPE=SINGLE,       DATASPACE A USAGE PRIVE             *
               STOKEN=JETON        JETON POUR Y ACCEDER (A RETOURNER)
         LTR  R15,R15          CREATION EFFECTUEE ?
         BNZ  FIN
         WTO  'DSPSERV CREATE SUCCESSFUL',ROUTCDE=11
* RENDRE ACCESSIBLE LE DATASPACE PAR MISE A JOUR DE NOTRE ACCESS-LIST
         ALESERV   ADD,            AJOUTER UNE ENTREE EN ACCESS-LIST   *
               ALET=ALET,          ALET (A RETOURNER)                  *
               STOKEN=JETON        JETON DE L'ESPACE-ADRESSE
         LTR  R15,R15
         BNZ  FIN
         WTO  'ALESERV ADD SUCCESSFUL',ROUTCDE=11
* PASSAGE A L'"ACCESS MODE REGISTER" POUR ACCES AU DATASPACE (VIA R4)
         SAC  512              PASSAGE A L'AR MODE
         LAM  R4,R4,ALET       CHARGER L'ACCESS-REGISTER 4
         L    R4,ORIGINE       DEBUT DU DATA-SPACE
         L    R5,TAILLE        TAILLE EN BLOCS DE 4K
         SLL  R5,12            X (2 PUISSANCE 12) C'EST-A-DIRE X 4096
         XR   R0,R0            PAS DE DONNEES ORIGINE
         XR   R1,R1            PAS DE DONNEES ORIGINE
         MVCL R4,R0            REMISE A ZERO DE TOUT LE DATA-SPACE
* ACCEDER AU DATASPACE (LES 500 PREMIERES PAGES)
         LA   R9,500
LOOP     OC   0(8,R4),0(R4)    DONNEES INCHANGEES
         LA   R4,4095(0,R4)    PAGE SUIVANTE
         LA   R4,1(0,R4)
         BCT  R9,LOOP
* SORTIE DE L'"ACCESS MODE REGISTER"
         SAC  0                RETOUR AU PRIMARY MODE
* SUPPRIMER LES DONNEES QUE CONTIENT LE DATASPACE (= MVCL)
         DSPSERV   RELEASE,                                            *
               BLOCKS=TAILLE,                                          *
               STOKEN=JETON,                                           *
               START=ORIGINE
* SUPPRIMER LE DATASPACE
         DSPSERV   DELETE,                                             *
               STOKEN=JETON
         XR   R15,R15
*
FIN      L       R13,4(R13)
         RETURN  (14,12),T,RC=(15)
SAVE     DS   18F              SAVE AREA
NOM      DC   CL8'TSTDATSP'    NOM DU DATA-SPACE
TAILLE   DC   F'400000'        TAILLE = 400000 BLOCS DE 4K = 1600M
ALET     DC   F'0'
* ZONES RENVOYEES PAR DSPSERV (ADD) :
ORIGINE  DC   F'0'
JETON    DS   CL8
         END
```

GLOSSAIRE des sigles employés dans cet ouvrage

ABEND (abnormal end): fin anormale d'un programme due à une erreur.

ACB (access control block): bloc de contrôle des méthodes d'accès VSAM et VTAM.

ACL (automatic cartridge loader) : chargeur de cassettes.

ACR (alternate CPU recovery): transfert de contrôle d'un processeur défaillant à un autre processeur.

ACS (automatic class selection): routines qui déterminent le profil d'un fichier en SMS.

AFP (advanced function printing) : fonctions avancées d'impression disponibles avec les imprimantes "intelligentes".

AFQ (available frame queue): réserve de cadres de pages disponibles.

AMS (access method services): langage de commande pour la gestion des fichiers VSAM.

APA (all-points addressability) : qualité des imprimantes "évoluées".

APAR (authorized program analysis report): correction d'une erreur logicielle, livrée par IBM et applicable par SMP.

APF (authorized program facility): possibilité de déclarer "autorisés" à émettre des instructions privilégiées un programme ou une bibliothèque de modules.

APG (automatic priority group): ensemble des priorités de distribution contrôlées par SRM.

APPC (application program to program communication) : mise en oeuvre du traitement coopératif, de programme à programme.

ASCB (address-space control block): bloc de contrôle représentant un espace-adresse vis-à-vis du système MVS.

ASCII (American standard code for information interchange) code binaire à 7 éléments.

ASID (address-space identifier): numéro affecté à un espace-adresse par MVS.

ASM (auxiliary storage manager): gestionnaire de la mémoire auxiliaire.

AUK (authorized user key): subpools 229 et 230 de la zone de mémoire virtuelle privée.

BCP (base control program) : désigne le système d'exploitation de base.

BCS (basic catalog structure): partie statique d'un catalogue ICF.

BLDL (build directory entry list): en MVS/SP, liste de répertoires de bibliothèques système gardée en mémoire.

BSAM (basic sequential access method): méthode d'accès séquentielle pour fichiers.

BSC (binary synchronous communication): protocole de communication synchrone, orienté caractère, mis au point par IBM (BSC type 2780 pour les transferts de fichiers, BSC 3270 pour les terminaux).

BTAM (Basic Telecommunications Access Method) : ancienne (1964) méthode d'accès de télécommunication de bas niveau.

CA (control area): groupe de CIs constituant un fichier VSAM.

CBIPO (custom-built installation process offering): méthode d'installation standard d'un système MVS.

CBPDO (custom-built product delivery offering) méthode de maintenance d'un système MVS.

CCA (common cryptographic architecture) : architecture de cryptographie d'IBM sur les processeurs S/390.

CCW (channel command word): ordre faisant partie d'un programme canal dans une opération d'entrée-sortie périphérique.

CFW (Cache Fast Write) : utilisation du cache des contrôleurs de disque pour stocker des fichiers temporaires (tri).

CI (control interval): unité de transfert et de stockage pour les enregistrements des fichiers VSAM.

CI (contention index): facteur utilisé par SRM pour mesurer l'importance d'un domaine dans l'ajustement du MPL.

CICS (Customer Information Control System): moniteur de télétraitement d'IBM.

CLIST (command list): langage interactif, à l'origine groupe structuré de commandes élémentaires (TSO, NETVIEW, etc.).

CMS (conversational monitoring system): moniteur interactif sous VM.

CPU (central processing unit): unité centrale, processeur.

CSA (common storage area): zone virtuelle commune à tous les espaces- adresses de MVS.

CSI (consolidated software inventory): fichier VSAM utilisé par SMP pour décrire les relations entre les éléments de base d'un logiciel.

CTC (channel to channel): liaison canal à canal entre ordinateurs.

CUA (common user access) : règles de l'interface d'accès aux applications SAA.

CVOL (control volume): ancien type de catalogue OS.

DADSM (direct access device space management): composant de DFP gérant l'espace disque.

DAE (dump analysis and elimination): composant gérant les fichiers DUMP en MVS-XA.

DAT (dynamic address translation): dispositif de traduction d'une adresse virtuelle en adresse réelle.

DBA (database administrator) : personne chargée de la conception et de la définition des bases de données.

DBRC (data base recovery control) : composant d'IMS dédié au recouvrement des bases de données DL/1 en cas d'erreur.

DB/DC (data base, data communication) : les deux composantes d'un SGBD (gestion des bases de données et des communications).

DB2 (Database 2): SGBD IBM de type relationnel.

DCB (data control block): bloc de contrôle décrivant un fichier et le moyen d'y accéder.

DCF (Document Composition Facility) : logiciel utilisé pour formatter et structurer un texte (SCRIPT/DCF).

DDR (dynamic device reconfiguration): possibilité de changer dynamiquement l'unité sur laquelle est monté un volume.

DEB (data extent-block): extension du DCB, résidant en AUK.

DES (data encryption standard): algorithme de chiffrement mis au point par IBM (standard U.S.).

DFP (data facility product): gestionnaire des données en MVS.

DFW (Dasd Fast Write) : utilisation du cache des contrôleurs de disque pour les opérations d'écriture.

DIE (disabled interrupt exit): exit asynchrone déclenché par les fonctions de timing.

DIM (Data in Memory) : évolution de MVS consistant à garder en mémoire les données les plus actives.

DIV (data in virtual): méthode d'accès utilisée pour les fichiers VSAM linéaires.

DJC (dependent job control): composant de JES3 permettant de gérer des travaux interdépendants.

DLF (data lookaside facility) : support d'hiperbatch.

DLIB (distribution library): fichiers de distribution des logiciels.

DLS (device level selection): fonction supportée par certains disques 3380, offrant à chaque disque 2 voies d'accès et permettant à chaque tête de string d'accéder à tous les disques de la batterie (16 adresses).

DLSE (DLS extended): DLS étendu (4 voies d'accès pour chaque disque, chaque tête de string peut accéder à 32 adresses).

DRM (dynamic reconfiguration management) : composant de MVS permettant la modification directe de la configuration I/O.

DSCB (data set control block): poste de la VTOC décrivant un fichier disque.

DSF (device support facility): programme ICKDSF utilisé pour initialiser ou analyser les disques.

DSP (dynamic support process): utilitaire JES.

EBCDIC (extended binary coded decimal interchange code): code à 8 bits utilisé sur de nombreux ordinateurs.

EC (engineering change) : changement de niveau technique d'une machine.

ECB (event control block): bloc de contrôle indiquant l'état (survenu ou non survenu) d'un événement.

EDTGEN (eligible device table generation): génération de la table des noms "ésotériques" ou génériques d'unités d'entrée-sortie.

ESA (Enterprise Systems Architecture): architecture d'ordinateurs IBM 370 apparue en 1988.

ESCD : ESCON director.

ESCM : ESCON manager.

ESCON (Enterprise Systems connectivity) : nouvelle architecture d'E/S sur fibre optique (mode série).

ESDS (entry-sequenced data set): organisation séquentielle des fichiers VSAM.

ESO (expanded storage only) : type d'hiperspaces qui résident uniquement en MAP (MVS/ESA).

ESTAE (extended, specify task abnormal exit): routine de reprise d'une erreur dans un programme.

ETR (external time reference) : unité 9037 de synchronisation pour le SYSPLEX.

EXCP (execute channel program): lancement d'un ordre d'entrée-sortie représenté par un programme canal constitué de CCWs.

FBA (fixed block architecture) : organisation des données sur disque par blocs (secteurs) de longueur fixe.

FLIH (first level interrupt handler): routine de traitement d'interruption.

FLPA (fixed LPA): partie de la zone LPA fixée en mémoire.

FM (facilities management) : service offert par une société extérieure consistant à gérer l'exploitation informatique d'une entreprise (sous-traitance).

FRR (functional recovery routine): routine de reprise en cas d'erreur dans un programme système.

FSS (functional subsystem) : interface avec JES, notamment pour les impressions AFP.

GDG (generation data group): groupe de fichiers classés par ordre chronologique.

GRS (global resource serialization): composant de sérialisation des accès en mono- ou multi-système MVS.

GTF (generalized trace facility): composant de trace d'événements pour le débogage.

HCD (hardware configuration definition) : outil permettant de définir la configuration périphérique à partir de MVS/ESA SP4.

HDA (head disk assembly) : partie du disque comportant les plateaux, les bras et les têtes de lecture.

Hiper (high impact-pervasive) : qualifie une PTF jugée très importante par IBM.

Hiperbatch (high performance batch) : fonction "data in memory" destinée au batch.

HSA (hardware storage area): zone de mémoire réelle réservée au matériel.

HSM (hierarchical storage manager): logiciel de gestion des fichiers et volumes disques.

ICF (integrated catalog facility): structure de catalogue destinée à remplacer les CVOLs et catalogues VSAM.

ICS (installation control specification): paramétrage associant des profils de performance à des espaces-adresses.

IDRC (improved data recording capability) : technique de compression des données sur cassette effectuée par microcode.

IML (initial microcode load): opération de chargement du microcode dans les systèmes 370 (ou pour d'autres matériels IBM) intervenant au démarrage de l'ordinateur.

IMS (Information Management System): moniteur de télétraitement fondé sur le SGBD hiérarchique DL/1.

IOCDS (input-output configuration data set): fichier intégré à l'ordinateur contenant la description de la configuration matérielle périphérique.

IOCP (input-output configuration program): programme utilisé pour décrire la configuration matérielle, qui aboutit à la création des IOCDS.

IOGEN (input-output generation): génération dans MVS des programmes et tables reflétant la configuration matérielle.

IOS (input-output supervisor): composant du noyau de MVS chargé de la gestion des opérations d'entrée-sortie.

IPCS (interactive problem control system): logiciel d'étude des dumps MVS fonctionnant sous TSO ou ISPF.

IPDS (intelligent printer data stream) : protocole réglant le dialogue entre PSF et les unités de contrôle des imprimantes.

IPL (initial program load): processus de démarrage d'un système MVS.

IPS (installation performance specification): définition des profils de performances des espaces-adresses.

IRLM (IMS resource lock manager) : logiciel gestionnaire des verrous pour accès aux bases DB2 ou IMS.

ISAM (indexed sequential access method): méthode d'accès pour fichiers sur disque fondée sur un index.

ISPF (interactive system productivity facility): logiciel interactif sous TSO.

ISV (interval service value): consommation minimale de ressources d'un espace-adresse avant qu'un vidage puisse avoir lieu.

JCL (job control language): langage de commande pour la description des travaux.

JES (job entry subsystem): composant d'accueil des travaux dans MVS.

KSDS (key sequenced data set): fichier VSAM à accès direct, par clé.

LAN (local area network) : réseau local.

LCMP (loosely coupled multiprocessing): réunion d'ordinateurs en couplage lâche, avec communication par CTC.

LCU (logical control unit): en XA, groupe d'unités de contrôle auxquelles sont rattachées en commun plusieurs unités périphériques.

LDS (linear data set): fichier VSAM de type linéaire, constitué de données sans informations de contrôle, accessible par DIV.

LLA (linklist lookaside, ou: library lookaside): répertoires de PDS chargés en mémoire.

LPA (link pack area): zone de mémoire virtuelle où résident de nombreux programmes réentrants de MVS (SVC, méthodes d'accès).

LPAR : logical partitioning (voir PR/SM)

LRU (least recently used): algorithme de remplacement des cadres de pages en MVS.

LSQA (local system queue area): zone virtuelle privée réservée aux blocs de contrôle système.

LU (logical unit) : entité logique (terminal, application, programme, etc.) participant à un réseau SNA, capable d'entrer en session avec une autre entité du réseau.

MAC (mandatory access control) : niveau de sécurité requis pour avoir le label de sécurité B1.

MAP (mémoire d'arrière-plan): mémoire étendue d'accès rapide présente sur certains ordinateurs.

MAS (multi-access spool): complexe JES2 dans lequel plusieurs systèmes MVS se partagent le même spool.

MDF (multiple domain facility): partitionnement logique des ordinateurs AMDAHL (voir PR/SM).

MDS (main device scheduling): composant d'allocation de JES3.

MIH (missing interrupt handler): composant de l'IOS traitant les entrées-sorties non satisfaites.

MLPA (modified LPA): zone de LPA pour tester ou remplacer des modules de PLPA ou FLPA.

MLPF (multiple logical processor facility): partitionnement logique pour les ordinateurs HDS (voir PR/SM).

MPF (message processing facility): composant de traitement automatique des messages.

MPL (multiprogramming level): nombre d'utilisateurs présents simultanément en mémoire, géré par SRM.

MTTW (mean time to wait): mode de gestion dynamique des priorités de distribution .

MVS (multiple virtual storage): système d'exploitation IBM fondé sur la mémoire virtuelle et le multitraitement.

MVSCP (MVS configuration program): programme de description de la configuration périphérique, destiné à remplacer l'IOGEN et l'EDTGEN.

NCP (network control program): logiciel des contrôleurs de communication (37XX principalement).

NIP (nucleus initialization program): composant d'initialisation de MVS succédant à l'IPL.

NLS (national language support) : support par un produit des particularités linguistiques d'un pays (autre que les Etats-Unis).

NPSI (Network Packet Switching Interface) : support réseau de la commutation par paquets.

NVS (non-volatile storage) : mémoire tampon des 3990-3, alimentée par une batterie.

OAM : object access method (méthode d'accès pour disques optiques).

OCO (object-code only) : politique d'IBM consistant à ne plus fournir les programmes sources de ses principaux logiciels.

OGL (Overlay Generation Language) : logiciel utilisé pour préparer des "overlays" électronique (pré-imprimés).

OS (operating system): system d'exploitation des ordinateurs IBM 360 et 370.

PARMLIB (parameter library): bibliothèque des paramètres de MVS (utilisée notamment pour le démarrage du système).

PDS (partitioned data set): fichier constitué de "membres" et d'un répertoire (directory) permettant l'accès direct aux membres.

pel (picture element) : "point" adressable dans les impressions AFP.

PKM (PSW key mask) : zone du registre de contrôle 3 qui indique à quelles clés le programme est autorisé.

PLPA (pageable LPA): partie paginable de la zone LPA.

POP : Principles of Operations, brochure décrivant l'architecture de programmation de la machine.

PPFA (Page Printer Formatting Aid) : logiciel d'aide à la création des FORMDEFs et PAGEDEFs.

PPT (program properties table): table indiquant les caractéristiques de certains programmes.

PR/SM (processor resource systems manager): dispositif microcodé permettant le partitionnement logique de certains ordinateurs IBM en plusieurs systèmes indépendants.

PSA (prefixed storage area): zone de mémoire réservée à un processeur pour le traitement des interruptions.

PSF (print services facilities) : gestionnaire des impressions AFP.

PSP (preventive service planning) : informations d'aide à l'installation de PTFs ou CBPDO.

PSW (program status word): zone décrivant l'état d'un processus en cours d'exécution.

PTF (program temporary fix): correction, amélioration ou apport de fonction nouvelle livrée au client par un fournisseur de logiciels.

PUT (program update tape) : bande livrée par IBM tous les un à deux mois et contenant des PTFs portant sur les produits du client.

QSAM (queued sequential access method): méthode d'accès fichier séquentielle, extension de BSAM, caractérisée par le buffering.

QTAM (Queued Telecommunications Access Method) : ancienne (1963) méthode d'accès de télécommunication.

RACF (resource access control facility): logiciel de sécurité IBM.

RAID (redundant array of inexpensive/independant disks) : organisation de disques en grappe.

RAS (reliability, availability, serviceability) : fiabilité, disponibilité, facilité d'emploi, qualités vers lesquelles doit tendre un système.

RCT (region control task): première tâche d'un espace-adresse.

RECFM (record format): format d'un enregistrement (bloqué ou non, spanné, de longueur fixe ou variable).

REXX (restructured extended executor language): langage de programmation utilisé à l'origine en CMS, puis en TSO.

RMF (resource measurement facility): logiciel de mesure permettant d'estimer les performances d'un système MVS.

RMM (removable media management) : composant de SMS qui gère les volumes cassettes.

ROT (rule of thumb): "règle du pouce", règle empirique permettant d'évaluer grossièrement ce que devraient être les performances d'un système.

RPS (rotational position sensing): fonction disque permettant à l'unité, durant son délai de positionnement sur un secteur du disque, de libérer le canal.

RRDS (relative record data set): fichier VSAM pour lequel les enregistrements sont accessibles par numéro d'ordre.

RS (recommandation de swap): facteur déterminé par un objectif de performance, qui mesure l'importance pour SRM d'un espace-adresse en vue d'un swap.

RSA (Rivest Shamir Adleman): algorithme de chiffrement à clés multiples.

RSM (real storage manager): gestionnaire de la mémoire réelle en MVS.

RTM (recovery and termination manager): composant de reprise et de terminaison des travaux.

SAA (system architecture applications): projet d'architecture unifiée IBM.

SAF (system authorization facility) : interface de sécurité qui peut passer le contrôle à RACF ou à d'autres routines de sécurité.

SDB (system-determined blocksize) : calcul automatique du blocksize pour les fichiers bloqués.

SDSF (spool ou system display and search facility): utilitaire TSO/ISPF d'accès au spool, en JES2.

SGBD (système de gestion de bases de données): logiciel de gestion intégrée des bases de données (DL/1, DB2, etc.).

SLIH (second level interrupt handler): gestionnaire de second niveau des interruptions, intervenant après le FLIH.

SLR (service level reporter): logiciel de collecte d'informations pour le suivi des performances et du niveau de service.

SMF (system management facility): composant MVS pour le suivi et la comptabilité de MVS.

SMP (system modification program): logiciel de support à la maintenance du système MVS.

SMS (system managed storage): produit d'automatisation de la gestion des fichiers.

SNA (systems network architecture): architecture de réseau IBM.

SoD (statement of direction) : intention de développement d'IBM.

SPE (small product enhancement) : amélioration apportée à un produit (souvent sous la forme d'une PTF importante).

SPOOL (simultaneous peripheral operations on-line): fichier sur disque utilisé par JES pour stocker les travaux, leurs paramètres et impressions.

SQA (system queue area): zone de mémoire virtuelle commune orientée système.

SQL (structured query language): langage de définition et de manipulation de bases de données (voir DB2).

SRB (service request block): unité de travail indépendante en MVS.

SRM (system resources manager): composant de MVS chargé d'optimiser l'utilisation des ressources.

SSD (solid-state device/disk) : "disque" non rotationnel consistant en mémoires électroniques.

SSI (subsystem interface): interface de communication standard avec un sous-système MVS.

SSSP (subsystem storage protection) : dispositif permettant d'affecter des clés de protection différentes pour un sous-système (CICS) et les applications qu'il gère, de manière à éviter l'écrasement du code CICS.

STC (started task control): tâche de gestion des travaux démarrés par la commande START de MVS.

SVC (supervisor call): routine du système appelée par l'instruction SVC.

SWA (scheduler work area): zone virtuelle privée orientée scheduler (JES).

Sysplex (system complex) : complexe de systèmes MVS synchronisés.

TCAM (Telecommunications Access Method) : ancienne méthode d'accès de télécommunication, destinée alors (en 1968) à remplacer QTAM et BTAM.

TCB (task control block): bloc de contrôle représentant une unité de travail liée à une tâche d'un espace-adresse.

TCM (thermal conduction module) : composant faisant partie du "packaging" de l'ordinateur.

TCMP (tightly coupled multiprocessing): couplage de processeurs dans un même ordinateur.

TLB (translation lookaside buffer): mémoire rapide destinée à accélérer la traduction d'adresse virtuelle.

TMP (terminal monitor program): tâche de gestion des utilisateurs TSO.

TOD (time of day) clock: horloge de grande précision intégrée au processeur.

TP (teleprocessing) : partie "online" d'un SGBD, permettant son emploi à distance (voir DB/DC).

TPI (test pending interrupt): instruction de l'architecture XA destinée à accélérer le traitement des entrées-sorties.

TRQ (true ready queue): file d'attente des ASCBs des espaces-adresses réellement prêts à être dispatchés.

TSO (time sharing option): moniteur de temps partagé d'emploi très répandu sous MVS.

TTR (track-track-record): mode d'adressage relatif d'un fichier sur disque, contrairement au MBBCCHHR qui est absolu.

UCB (unit control block): bloc de contrôle représentant une unité périphérique.

UIC (unreferenced interval count): compteur permettant d'évaluer le "vieillissement" d'une page en mémoire.

VIO (virtual input/output): entrée-sortie virtuelle, privilégiant la mémoire pour éviter les accès aux périphériques.

VLF (virtual lookaside facility): possibilité de stocker en mémoire des éléments très accédés.

VM (virtual machine): hyperviseur d'architecture IBM 370.

VSAM (virtual storage access method): méthode d'accès des systèmes IBM 370.

VSCR (virtual storage constraint relief) : évolution de MVS et de ses sous-systèmes visant à se libérer de la limitation de la mémoire virtuelle en-dessous des 16 Mo, par mise de données au-dessus de la "barre".

VSE (virtual storage extended): système d'exploitation IBM pour moyenne gamme.

VSM (virtual storage manager): gestionnaire de la mémoire virtuelle en MVS.

VTAM (virtual telecommunication access method): logiciel de télécommunication IBM supportant SNA.

VTOC (volume table of contents): fichier disque indiquant le contenu du volume.

VVDS (VSAM volume data set): partie d'un catalogue ICF liée à un volume disque.

XA (extended architecture): architecture étendue IBM 370 (extension de l'adressage à 31 bits au lieu de 24).

XCF (cross system coupling facility) : service de MVS/ESA permettant de basculer un composant logiciel d'un processeur défaillant vers un processeur disponible.

XRF (extended recovery facility) : désigne divers dispositifs de recouvrement de pannes logicielles ou matérielles utilisés pour IMS ou CICS.