systèmes d'exploitation des ordinateurs

CROCUS

Principes de conception

CROCUS
Nom collectif de : J. BELLINO, C. BÉTOURNÉ, J. BRIAT, B. CANET, E. CLEEMANN, J.-C. DERNIAME, J. FERRIÉ, C. KAISER, S. KRAKOWIAK, J. MOSSIÈRE, J.-P. VERJUS

DUNOD
phase spécialité | informatique

AVANT-PROPOS

L'idée de cet ouvrage est née lors d'une Ecole d'Eté d'Informatique organisée à Alès en 1971, sous le patronage de l'Association Française pour la Cybernétique Economique et Technique, pour permettre à des enseignants et à des chercheurs de réfléchir en commun à une présentation pédagogique des matières de l'informatique.

Un groupe de travail s'y est constitué pour rédiger des notes d'enseignement sur les systèmes d'exploitation. Un plan de cours détaillé publié aux Etats-Unis en juin 1971 par le « Cosine Committee of the Commission on Education of the National Academy of Engineering » sous le titre « An Undergraduate Course on Operating Systems Principles » a servi de document de départ et a permis de dépasser le stade des notes de cours. Divers membres du groupe ont pris pour base de leur enseignement la rédaction obtenue, ce qui a permis d'en mettre au point la présentation pédagogique. Enfin, cet ouvrage a constitué la matière des cours de trois Ecoles de Printemps sur les Systèmes d'Exploitation (Les Arcs, 1973 ; Auron, 1974 et 1975).

Ce livre est le résultat d'un travail fait en commun du début à la fin de la rédaction. C'est pourquoi un nom collectif, Crocus, a été choisi comme nom d'auteur par le groupe constitué de : J. Bellino (Centre Scientifique IBM de Grenoble), C. Bétourné (Université de Toulouse III (*)), J. Briat (Université de Grenoble I), B. Canet (Université de Rennes I), E. Cleemann (Université de Grenoble I), J. C. Derniame (Université de Nancy I), J. Ferrié (Iria/Laboria), C. Kaiser (Iria/Laboria), S. Krakowiak (Université de Grenoble I (*)), J. Mossière (Université de Grenoble I (*)) et J. P. Verjus (Université de Rennes I). Des observateurs ont participé aux travaux du groupe. Ce sont : G. Bazerque (Université de Toulouse I), J. C. Boussard (Université de Nice), C. Girault (Université de Paris VI) et C. Carrez (Université de Lille I). Nous tenons à remercier plus particulièrement ce dernier pour son rôle de contestataire permanent.

Nous exprimons notre reconnaissance à toutes les secrétaires, en particulier à Mmes G. Perez et M. Suard qui ont assuré une grande partie de la frappe des nombreuses versions intermédiaires du manuscrit, ainsi qu'à M. J. Riguet, qui s'est chargé de l'exécution de toutes les figures.

Nous remercions enfin les organismes d'appartenance des différents membres du groupe pour leur soutien matériel au cours de l'élaboration de cet ouvrage.

(*) Anciennement Iria.

AVANT-PROPOS DE LA SECONDE ÉDITION

L'évolution rapide des recherches et des réalisations dans le domaine des systèmes d'exploitation des ordinateurs a posé aux auteurs un délicat problème pour la préparation de cette seconde édition : dans quelle mesure fallait-il tenir compte des progrès réalisés depuis 1973, date d'achèvement du texte initial ?

Nous avons choisi de nous en tenir à une simple correction des erreurs matérielles relevées depuis la parution de la première édition. Une raison principale a motivé ce choix : l'évolution de l'enseignement de l'informatique a fait que la matière de cet ouvrage est maintenant intégrée, pour une large part, aux programmes de formation de base en informatique : maîtrise, Instituts de Programmation, Ecoles d'Ingénieurs. Cette formation met l'accent sur des principes généraux de conception dont la présentation nous paraît, pour l'essentiel, toujours d'actualité. L'abondance des travaux récents dans plusieurs domaines (protection, modèles, structuration des systèmes, ...) aurait pu justifier une refonte complète de certaines parties de l'ouvrage. Mais nous pensons que les concepts récemment introduits n'ont pas encore atteint une stabilité suffisante pour être intégrés à un enseignement de base, et que leur présentation relève encore, pour un temps, des enseignements spécialisés ou de la préparation à la recherche.

Depuis l'achèvement de la première édition, plusieurs ouvrages didactiques ont été publiés sur les systèmes d'exploitation. Nous avons inclus dans un complément bibliographique ceux qui nous paraissent, à des titres divers, les plus intéressants.

Nous insistons de nouveau sur l'importance que nous attachons, dans la présentation pédagogique de la matière de cet ouvrage, aux études de cas menées en parallèle : études de systèmes existants, projets de systèmes ou parties de systèmes. Quelques indications à ce sujet sont données dans le complément bibliographique.

Nous tenons enfin à remercier tous ceux qui nous ont aidés par leurs suggestions et critiques, et en particulier les enseignants, étudiants et stagiaires des divers cycles de formation où notre ouvrage a été utilisé.

Octobre 1976

PRÉAMBULE

L'informatique met en œuvre des ressources importantes et coûteuses, tant en ce qui concerne le matériel que les programmes. Un souci d'économie conduit à rendre ces ressources communes à un groupe de traitements. Cette fonction est remplie par un ensemble de programmes et de dispositifs câblés qui constituent le système d'exploitation (nous dirons plus simplement : le système). Ce système a pour charge de mettre à la disposition d'un groupe d'utilisateurs les ressources qu'il administre. Bien que les programmes inclus dans un système ne soient pas essentiellement différents des autres programmes, ils se particularisent par leur aspect dynamique et par la nécessité d'assurer l'indépendance mutuelle d'un ensemble d'utilisateurs. Le déroulement des programmes d'un système dépend de la nature de leurs données et de l'occurrence d'événements externes, ce qui rend le comportement d'un système difficilement prévisible et reproductible. Les problèmes qu'impliquent la mise en commun d'objets, leur partage, leur protection, la nécessité de les nommer, la synchronisation des actions qui peuvent être entreprises sur eux, prennent donc plus d'importance dans un système que dans d'autres programmes. Ces problèmes peuvent être abordés à propos d'un système particulier. De nombreux exemples de réalisations sont exposés dans la littérature technique, mais il est malaisé de dégager de ces descriptions, dont l'abord est souvent difficile, des principes généraux de conception. C'est à un tel effort de synthèse que nous avons tenté de contribuer.

Nous ne proposons pas dans cet ouvrage des règles de construction des systèmes d'exploitation, mais simplement des éléments pour leur conception. Plus exactement nous tentons de dégager, chaque fois que cela est possible, les principes qui s'appliquent à la conception des systèmes ou qui semblent devoir y contribuer dans les années à venir.

Ces principes intéressent les concepteurs de systèmes mais aussi ceux qui participent de près à leur évolution, qu'ils en assurent la maintenance ou l'exploitation. Ce travail leur est donc destiné, ainsi qu'aux étudiants et aux chercheurs spécialisés dans les problèmes liés aux systèmes d'exploitation ou, plus généralement, préoccupés par la conception ou l'utilisation de grands programmes. De façon plus précise, cet ouvrage s'adresse aux concepteurs, aux programmeurs de systèmes, aux étudiants en conception de systèmes ou en programmation avancée, ainsi qu'à tous les enseignants en informatique.

Nous supposons acquises la connaissance de l'anatomie d'un système simple ainsi que l'expérience de la programmation et de l'utilisation d'un système. Le lecteur devra avoir une idée claire du rôle d'un assembleur, d'un compilateur, d'un chargeur, d'un éditeur de liens, d'un interpréteur et d'un système de gestion de fichiers.

De même nous supposons connues les notions suivantes :

— l'organisation de l'unité de commande et de l'unité de traitement, les techniques d'adressage, la structure d'une instruction, le fonctionnement des mécanismes câblés d'exécution des instructions, les modes de fonctionnement (maître-esclave), les notions d'interruption et de déroutement ;

— l'organisation et les caractéristiques des principaux types de mémoire (circuits intégrés, tores, tambours, disques, bandes,...) ;

— le fonctionnement des différents types d'organes d'accès et leurs rapports avec l'unité de commande ;

— l'emploi des structures usuelles de données (tables, listes, piles, arborescences) et leur représentation en machine ;

— la structuration des programmes (récursivité, réentrance).

Il est également nécessaire de connaître un langage de programmation. Nous utilisons un langage inspiré d'ALGOL 60 pour la description des algorithmes, mais la connaissance d'un langage de niveau équivalent est suffisante pour leur compréhension.

La bibliographie qui figure à la fin de ce préambule recouvre à peu près les connaissances prérequises.

Cet ouvrage peut constituer un guide pour l'étude des principes de conception des systèmes d'exploitation des ordinateurs, mais il ne saurait être à lui seul suffisant. Il doit être complété par l'étude pragmatique d'un système réel. En particulier, il est recommandé aux enseignants d'illustrer les concepts par des exemples pris dans un ou plusieurs systèmes.

Des exercices repérés dans l'ordre de difficulté croissante par un nombre de 1 à 3 figurent à la fin de chaque chapitre. Leur but est de fournir l'occasion d'appliquer les connaissances acquises dans le cours et d'approfondir certains points non traités dans le corps de l'ouvrage. Pour la plupart des exercices, des schémas de solution sont regroupés *in fine*.

L'ouvrage contient enfin une bibliographie générale, avec regroupement des références par chapitre, et un index des termes le plus couramment utilisés.

BIBLIOGRAPHIE POUR LES CONNAISSANCES PRÉREQUISES

Arsac J., *Les systèmes de conduite des ordinateurs*, Dunod (2e édition, 1970).

Hopgood F. R. A., *Compiling techniques*, Macdonald Computer Monographs (1969).

Knuth D. E., *The art of computer programming*, **vol. 1** : Fundamental algorithms, Addison-Wesley (1968), en particulier 2.1 à 2.4.

Meinadier J. P., *Structure et fonctionnement des ordinateurs*, Larousse (1971).

Profit A., *Structure et technologie des ordinateurs*, Armand Colin (1970).

TABLE DES MATIÈRES

1

INTRODUCTION

La variété des formes externes que peuvent prendre les systèmes d'exploitation des ordinateurs et la diversité des fonctions qu'ils assurent rendent malaisée toute tentative de définition générale ou de classification rigoureuse. C'est pourquoi, après avoir donné une idée des principales fonctions d'un système et des aspects externes le plus souvent rencontrés, nous tenterons de dégager quelques caractéristiques communes qui guideront notre étude.

1.1 FONCTIONS ET ASPECTS EXTERNES DES SYSTÈMES

1.11 FONCTIONS D'UN SYSTÈME

Un système peut être examiné sous des points de vue très divers. On peut considérer qu'il remplit, vis-à-vis de ses utilisateurs, un certain nombre de fonctions dont la liste ci-dessous n'est pas exhaustive.

— Gestion et conservation de l'information : il s'agit d'offrir aux utilisateurs des moyens de créer, de retrouver et de détruire les objets sur lesquels ils veulent effectuer des opérations.

— Gestion de l'ensemble des ressources pour permettre l'exécution d'un programme : le système a pour rôle de créer un environnement nécessaire à l'exécution d'un travail.

— Gestion et partage de l'ensemble des ressources : le système est alors chargé de répartir ces ressources (matériels, informations et programmes) entre les usagers. Pour cela, il doit réaliser un ordonnancement des travaux.

— Extension de la machine câblée : le système a ici pour rôle de masquer certaines limitations ou imperfections du matériel, ou de simuler une machine

différente de la machine réelle. L'utilisateur a alors à sa disposition une « machine virtuelle » munie d'un « langage étendu », c'est-à-dire d'un mode d'expression mieux adapté que les seules instructions câblées. A l'aide de ce langage, il peut commander l'exécution de ses programmes et en effectuer la mise au point. Font aussi partie du langage étendu les commandes de l'opérateur et les directives utilisées pour la « génération » du système.

1.12 ASPECTS EXTERNES DES SYSTÈMES

Les systèmes se présentent sous un grand nombre d'aspects externes ; cette diversité reflète la variété des tâches à remplir et des caractéristiques des matériels utilisés. En dehors d'une classification historique [Rosin, 69], on peut classer les systèmes suivant la fonction principale qu'ils remplissent. On pourrait ainsi distinguer :

a) les systèmes orientés vers la commande de processus industriels (exemples : système de conduite d'un haut fourneau, système de guidage d'une fusée, central téléphonique),

b) les systèmes orientés vers la conservation et la gestion de grandes quantités d'information (exemples : systèmes de documentation automatique, système de gestion de comptes bancaires, systèmes de réservation de places),

c) les systèmes destinés à la création et à l'exploitation de programmes. Ces derniers systèmes peuvent eux-mêmes être classés suivant le degré d'interaction possible d'un utilisateur avec ses programmes (systèmes de traitement par trains ou systèmes conversationnels), suivant le mode d'entrée des programmes (local ou à distance, par fournées ou continu), suivant le mode de partage des ressources (mono- ou multiprogrammation), suivant les possibilités du langage étendu (langage unique ou langages multiples). Ces systèmes peuvent présenter à un degré variable certains aspects du type *a*) ou *b*) : ainsi, les contraintes de temps sont importantes pour un système comportant des usagers interactifs.

1.2 CARACTÉRISTIQUES COMMUNES

Les divers systèmes énumérés précédemment posent à leur concepteur les mêmes types de problèmes, bien qu'ils diffèrent par leurs objectifs, par leurs contraintes et par leur aspect externe. L'analyse de leur structure et de leur fonctionnement permet en effet de dégager un certain nombre de caractéristiques communes :

— gestion et partage d'un ensemble de ressources,
— désignation des objets et accès à l'information,
— coopération entre processus parallèles,
— protection des informations et fiabilité des programmes.

Dans le corps de l'ouvrage, nous tentons pour chacun de ces aspects de dégager les concepts utiles à la résolution des problèmes rencontrés. Lorsque l'état des connaissances le permet, nous proposons une approche aussi générale que possible, illustrée par des exemples pris dans des systèmes existants ; dans le cas contraire, nous suivons une démarche plus pragmatique en partant de réalisations particulières.

Les exemples sont, en général, empruntés à des systèmes importants fonctionnant sur des matériels moyens ou gros. Toutefois, les notions présentées s'appliquent également aux petits systèmes.

1.21 PARTAGE DES RESSOURCES PHYSIQUES

Des raisons économiques amènent les utilisateurs d'ordinateurs à mettre en commun leur matériel, ce qui pose le problème de la planification de son utilisation et de son partage. Une manière immédiate de résoudre ce problème consiste à admettre un seul programme à la fois en mémoire ; ce programme dispose donc pendant son déroulement de toutes les ressources de l'installation laissées libres par le système. L'utilisation de l'ordinateur peut alors être entièrement planifiée par le service d'exploitation : il suffit de réserver un temps suffisant à chacun des programmes et de prévoir l'ordre dans lequel ils seront exécutés.

Des considérations d'efficacité conduisent à adopter des méthodes de partage plus complexes.

Exemple. Soit un système de traitement par trains en monoprogrammation avec gestion simultanée des entrées-sorties, organisé comme suit : un seul programme d'utilisateur est présent à la fois en mémoire ; les entrées-sorties (y compris l'entrée des programmes eux-mêmes) ont lieu depuis (ou vers) une zone sur disque réservée à cet effet et sont effectuées par des programmes appelés « symbionts ».

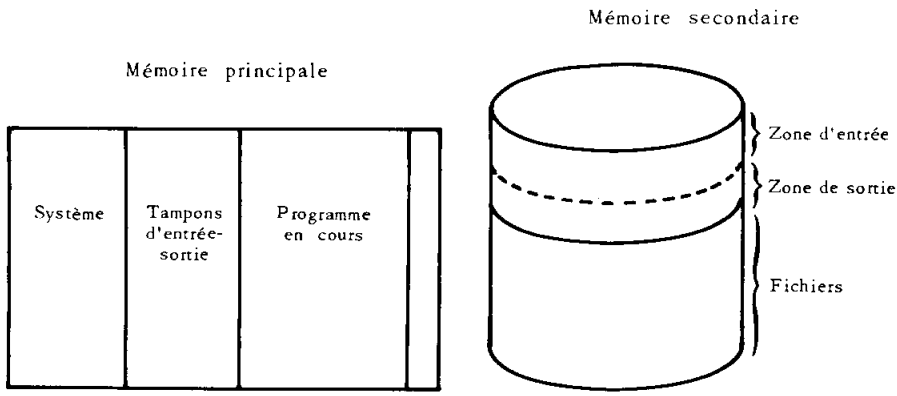

Figure 1. Exemple de partage de la mémoire.

Le disque est divisé en deux régions : l'une est réservée aux entrées-sorties, l'autre à la conservation des informations des utilisateurs (fichiers). De même, la partie de mémoire principale non occupée par le système est divisée en une zone de tampons d'entrée-sortie et une zone réservée au programme en cours d'exécution.

Une telle organisation, qui suppose l'existence d'une unité d'échange pouvant fonctionner en parallèle avec l'unité centrale, permet de réduire le temps global de traitement d'un ensemble de programmes.

Le système gère plusieurs files d'attente :

— la file des travaux en attente d'exécution (ces travaux sont rangés dans la zone « entrée » sur disque,

— les files des informations à sortir (il y a une file par type de périphérique).

Cet exemple simple permet de mettre en évidence des notions communes à de nombreux systèmes :

— la demande simultanée d'une même ressource par plusieurs utilisateurs conduit à un partage qui peut être séquentiel (exemple : l'unité centrale) ou simultané (exemple : le disque, la mémoire principale). Dans le cas de la mémoire, le partage peut être statique (les limites des différentes zones sont fixées une fois pour toutes) ou dynamique (les limites peuvent varier),

— à un instant donné, l'ensemble des demandes relatives à une ressource peut excéder la quantité disponible de cette ressource, et cette situation provoque l'attente des demandeurs.

D'une façon plus générale, il existe dans un système un ensemble de ressources utilisables et un ensemble de travaux (ou charge) dont le traitement entraîne des demandes de ces ressources. Le système est chargé de l'attribution des ressources suivant les objectifs qui ont été fixés à sa conception et qui peuvent consister à :

— mieux utiliser le matériel ou certaines parties du matériel,

— mieux satisfaire les utilisateurs, ce qui peut s'exprimer sous diverses formes (réduire le temps de réponse, respecter les échéances...).

Ces objectifs peuvent être contradictoires. La définition du système implique un certain nombre de choix de conception, qui influent sur les performances finales. Ainsi, dans l'exemple ci-dessus, les choix importants concernent :

— le mode de partage de la mémoire principale et de la mémoire secondaire (statique ou dynamique ? si dynamique, selon quel critère ?),

— le mode de gestion des différentes files d'attente (avec ou sans priorité ?).

Il est commode, pour chacune des ressources importantes d'un système (processeurs, mémoire centrale, mémoire secondaire), d'étudier séparément les stratégies individuelles permettant de la gérer. Les résultats d'une telle étude sont applicables dans les cas où les problèmes d'allocation des divers types de ressources peuvent être découplés ; mais le plus souvent ces problèmes interfèrent.

La mise en œuvre de plusieurs stratégies particulières indépendantes dans un système comportant plusieurs types de ressources peut conduire à des

conflits si elle est faite sans précautions. En particulier peuvent apparaître deux types de phénomènes :

— l'écroulement, ou la dégradation des performances, dû à une mauvaise gestion de l'ensemble mémoire-processeur dans un système multiprogrammé,

— l'interblocage d'un groupe de processus, situation dans laquelle chaque processus est en attente d'une ressource possédée par un autre processus du groupe.

Toute politique d'allocation de ressources doit tenir compte de ces dangers ; elle doit donc être conçue de manière globale.

1.22 GESTION DE L'INFORMATION

L'utilisateur d'un système informatique désire effectuer des traitements sur des objets ; ces objets peuvent être, entre autres, les fichiers contenant des programmes ou des données, les segments dans les systèmes à mémoire segmentée, les variables, tableaux et structures définis dans divers langages de programmation.

L'utilisateur désigne les objets qu'il veut employer grâce à des identificateurs. Pour qu'un objet puisse être traité dans un ordinateur, il faut lui associer des informations le désignant sans ambiguïté ; de toute manière, il faut qu'au moment du traitement effectif, l'objet puisse être localisé par le processeur chargé de ce traitement. Ces différentes informations de localisation ou de désignation constituent les noms de l'objet. Au cours de son existence, l'objet peut être désigné par des noms différents.

L'exemple qui suit, emprunté au langage de commande du système SIRIS 7 sur CII 10070, permet de préciser l'établissement de la correspondance entre identificateurs, noms et objets.

Exemple.

! Fortran si,go	Compiler le programme qui suit et ranger le résultat de la compilation dans le fichier appelé *go*.
⋮	
x = sin (y + z)	
Write (273) x	Ecrire la valeur de *x* sur le support associé au descripteur *273*.
⋮	
①- - - - - - - - - - - - -	
! Assign bib, fil, (nam,f4lib), (sts,old)	Assignation au descripteur *bib* du fichier existant (bibliothèque *Fortran*).
! Link	Editer le programme contenu dans le fichier *go*, en allant chercher les références externes dans le fichier associé au descripteur *bib* ; placer le résultat (module de chargement) dans le fichier *go* (option par défaut).
: Option (unsat, bib)	
②- - - - - - - - - - - - -	

! Assign 273, fil, (nam,resul), (unt,mt, (vol, 450))	Assignation au descripteur d'identificateur *273* d'un nouveau fichier *resul*, à créer sur la bande magnétique n° *450*.
③- - - - - - - - - - - - -	
! Run	Charger le programme édité contenu dans le fichier *go* et l'exécuter.
④- - - - - - - - - - - - -	

En 1, le compilateur a produit un programme translatable rangé dans le fichier *go*. Dans ce programme :

— le nom de *x* est une adresse relative (déplacement) par rapport à l'origine du programme,

— le nom *sin* n'est pas défini : l'identificateur *sin* figure dans une liste de références non satisfaites attachée au programme,

— le descripteur *273* n'a pas de valeur.

En 2, l'éditeur de liens a produit, dans le fichier *go*, un module de chargement (translatable) constitué en réunissant le texte initialement contenu dans le fichier *go* à ceux du fichier *f4lib* dont les identificateurs figuraient parmi les références non satisfaites de *go* : par exemple, *sin* est désigné maintenant par un déplacement relatif à l'origine du module de chargement et toutes les références à *sin* se font par ce nom.

En 3, le descripteur *273* a pour valeur la chaîne de caractères *'resul'* ; la correspondance entre l'identificateur *273* et le fichier appelé *resul* pourra donc être complètement établie à l'exécution.

En 4, les adresses en mémoire ont été fixées pour le module de chargement contenu dans *go* : maintenant *x* et *sin* désignent effectivement des objets.

Sur cet exemple, on peut constater que la correspondance entre identificateur et objet est établie en plusieurs étapes : on dit que l'identificateur est progressivement lié à l'objet qu'il désigne.

Le but de cette opération de liaison (« binding ») est essentiellement d'associer, de façon plus ou moins durable, l'objet à des emplacements adressables par un processeur, ce qui est la seule façon de le consulter ou de le modifier. La liaison est établie à l'aide d'une chaîne de noms partant de l'identificateur et aboutissant à l'objet désigné. Cette chaîne peut être construite en respectant le sens de l'identificateur vers l'objet : c'est le cas, dans l'exemple ci-dessus de la variable *x*. Elle peut aussi être construite dans un ordre différent : c'est le cas, dans notre exemple, de la liaison de l'identificateur *sin* lors de l'édition de liens, où sont reliées deux parties de la chaîne constituées à l'avance.

L'opération d'« assignation » des descripteurs de fichiers fournit le moyen de retarder jusqu'au stade de l'exécution le choix du fichier utilisé : un même programme peut être exécuté plusieurs fois avec des fichiers différents sans avoir à modifier son texte. De façon plus générale, le principe consistant à retarder les liaisons (« delay binding time ») permet une plus grande souplesse d'utilisation, qui se paye par une plus grande complexité et, parfois, une perte d'efficacité.

Dans de nombreux systèmes, d'autres opérations de liaison peuvent encore intervenir pendant l'exécution du programme : les adresses obtenues à l'édition de liens sont des adresses « virtuelles » et le mécanisme de transformation de ces adresses virtuelles en adresses réelles peut être plus complexe qu'une simple translation statique : production dynamique d'adresses par des mécanismes de segmentation par exemple.

On demande souvent à un système de permettre à plusieurs utilisateurs d'accéder à des informations communes. Ce partage pose des problèmes supplémentaires puisque l'indépendance des utilisateurs doit toujours être assurée. On peut concevoir deux façons de partager un objet : constituer de cet objet autant d'exemplaires distincts que nécessaire, ou bien permettre à chaque utilisateur d'accéder à l'exemplaire unique de l'objet. Dans le premier cas, il faut assurer la cohérence des différents exemplaires ; dans le second cas, on a encore le choix entre l'affectation à l'objet d'un nom différent pour chaque utilisateur, ou l'utilisation d'un nom commun.

Exemple. Une procédure partagée peut être recopiée en autant d'exemplaires qu'il y a de programmes qui l'utilisent. Une autre solution consiste à n'avoir qu'un exemplaire réentrant. Dans ce dernier cas, les différents noms qu'elle possède pour les différents programmes qui l'utilisent doivent en dernier ressort désigner le même objet.

1.23 COOPÉRATION DES PROCESSUS

Dans un système, plusieurs activités peuvent se dérouler simultanément. Ces activités résultent de l'exécution de programmes. Nous utiliserons pour les désigner le terme de processus.

Reprenons l'exemple, introduit en 1.21, d'un système de monoprogrammation avec « symbiont ». A un instant donné, on peut observer l'exécution d'un programme par l'unité centrale et d'une entrée-sortie par l'unité d'échange. Chacune de ces activités fait partie d'un processus. Le déroulement de chaque processus est déterminé par la suite d'instructions exécutée par l'organe actif correspondant, ou processeur (unité centrale ou unité d'échange).

Il est commode d'introduire un processus distinct pour une activité que l'on veut considérer comme indépendante. Une telle décomposition ne tient pas compte du fait que ces processus peuvent ou non se dérouler simultanément ; en particulier, elle ne tient pas compte du nombre de processeurs. On dit alors que les processus ainsi définis sont logiquement parallèles.

Notons que les notions de programme et de processus sont distinctes : chaque exécution d'un même programme correspond à un processus distinct ; si de plus ce programme est réentrant, ces exécutions peuvent être simultanées.

Dans l'exemple du 1.21, l'exécution du train de programmes d'utilisateurs et l'exécution du « symbiont » peuvent être considérés comme des processus logiquement parallèles : le processus *travail* et le processus *symbiont*. Toutefois, ils ne se déroulent pas toujours simultanément ; lorsque *symbiont* utilise l'unité

centrale, l'exécution de *travail* est suspendue. En dehors de ce conflit dû à une insuffisance de ressources, les deux processus ont d'autres interactions :

— le processus *symbiont* ne doit pas pouvoir accéder à un tampon que le processus *travail* est en train de remplir,

— lorsqu'un tampon de sortie est plein, *travail* doit réveiller *symbiont* si ce dernier est inactif,

— lorsqu'un nouveau programme est introduit dans la file d'entrée, *symbiont* doit réveiller *travail* si ce dernier est inactif.

Cet exemple met en évidence l'existence de différents types d'interaction entre processus :

— conflit pour l'accès à une ressource (unité centrale ou tampon d'entrée-sortie) qui ne peut être utilisée que par un seul processus à la fois (exclusion mutuelle),

— action directe (synchronisation) d'un processus sur un autre : mise en attente ou réveil.

On rencontre dans un système bien d'autres formes de parallélisme : par exemple les demandes de service faites par des utilisateurs depuis des consoles d'accès direct correspondent à des processus logiquement parallèles dont le déroulement peut être assuré par un système de multiprogrammation à un ou plusieurs processeurs.

On peut considérer un système d'exploitation comme un ensemble de processus parallèles pouvant interagir. Pour mettre en œuvre ces processus, on a deux problèmes à résoudre :

— écrire les programmes décrivant chaque activité individuelle,

— concevoir des mécanismes d'interactions permettant les différents types de coopération : exclusion mutuelle, synchronisation, communication d'information.

1.24 PROTECTION

La coexistence, à l'intérieur d'un système, d'informations appartenant à différents utilisateurs impose la protection de ces informations contre les erreurs de programmation ou contre les malveillances. Par exemple, les informations utilisées pour la gestion du système lui-même doivent être inaccessibles aux programmes des utilisateurs ; un utilisateur peut souhaiter n'autoriser la consultation ou la modification de ses informations privées qu'à certains utilisateurs explicitement spécifiés ; plusieurs utilisateurs peuvent ainsi avoir des droits différents sur une même ressource.

Plus généralement, le rôle d'un système de protection est de garantir, dans tous les cas, l'intégrité de certaines ressources protégées. Cette protection peut être mise en œuvre par différents mécanismes câblés ou programmés. Par exemple, de nombreux ordinateurs comportent deux modes d'exécution : maître et esclave, et certaines instructions (entrée-sortie, ...) ne peuvent être exécutées qu'en mode maître.

1.3 PROBLÈMES DE CONCEPTION ET D'ÉVALUATION

La conception des systèmes se trouve actuellement à une étape intermédiaire entre un stade empirique où elle est basée sur le savoir-faire et l'intuition, et un stade scientifique où elle pourrait s'appuyer sur des études théoriques conduisant à des méthodes de construction des systèmes. Deux approches, entre autres, sont envisageables :

1) faciliter la conception par une meilleure connaissance du comportement des systèmes existants, soit à l'aide de mesures, soit par la construction de modèles de comportement,

2) faciliter la réalisation proprement dite d'un système en définissant des techniques d'écriture pour les gros programmes dont la réalisation pose d'importants problèmes de documentation et de communication entre les participants.

1.31 MESURES ET MODÈLES DE SYSTÈMES

La conception et la mise au point d'un système sont facilitées par la connaissance d'informations quantitatives sur le système lui-même ou sur des systèmes existants analogues.

Différentes techniques de mesures, câblées ou programmées, permettent d'obtenir des informations :

— sur le comportement d'un système (temps de réponse, débit des travaux, utilisation des ressources, fréquence de certains événements),

— sur le comportement d'un utilisateur en mode interactif ou sur le comportement dynamique d'un programme.

Ces informations peuvent également être importantes pour choisir un matériel et un système, pour modifier une configuration, pour assurer la comptabilité de l'utilisation des ressources et pour optimiser les programmes.

Des renseignements sur le comportement d'un système peuvent également être obtenus par l'utilisation de modèles qui en fournissent une image approchée. Ces modèles peuvent être traités par le calcul, si leur complexité le permet, et fournir ainsi des formules directement utilisables ; si leur complexité est trop grande, ils peuvent être traités par simulation.

1.32 MÉTHODOLOGIE DE CONCEPTION

La réalisation d'un système nécessite l'intervention de nombreuses personnes et peut durer longtemps. Etant donné l'absence de techniques automatiques de construction, la fiabilité d'un système dépend beaucoup de la méthode suivie pour sa réalisation. Ainsi, une mauvaise documentation des programmes ou l'absence de conventions précises de liaison entre les constituants du système sont des sources importantes d'erreur et diminuent donc considérablement la fiabilité du produit.

La construction d'un programme peut être simplifiée si ses constituants sont décrits sous la forme de modules pouvant être combinés sans avoir à connaître les détails de leur réalisation interne.

Chaque module ne doit communiquer avec les autres qu'en suivant des règles bien définies : les spécifications d'interface. Ces règles doivent en particulier imposer une représentation cohérente des informations communes.

Il est souhaitable de disposer de méthodes d'analyse permettant la décomposition d'un système en modules. Deux méthodes peuvent être employées :

— une méthode de conception descendante consistant à définir l'implantation de la solution par étapes ; au cours de chacune d'elles, on complète les définitions de fonctions ou d'informations utilisées aux étapes précédentes,

— une méthode de conception ascendante qui utilise des fonctions ou des informations déjà décrites pour la réalisation de nouvelles fonctions.

Dans la pratique, on utilise alternativement l'une et l'autre méthode.

1.4 ORGANISATION DE L'OUVRAGE

Les divers aspects des systèmes qui viennent d'être considérés sont traités dans l'ordre suivant :

Chapitre 2 : Processus.
Chapitre 3 : Gestion de l'information.
Chapitre 4 : Gestion des ressources physiques.
Chapitre 5 : Protection.
Chapitre 6 : Mesures et modèles de systèmes.
Chapitre 7 : Méthodologie de conception et de réalisation.