OBJETS ET FONCTIONS PRIMITIFS D'UN SYSTÈME D'EXPLOITATION

J.P. VERJUS

1975

Ce document est la dernière version, à cette date, d'un cours donné à Rennes (DEA : années 71/72 à 74/75), à l'école d'été de l'AFCET (Grenade, 1973), à Paris VI (DEA, 73/74) et à Lausanne (3ème Cycle Romand, 1975).

Il est certain qu'il doit recevoir encore de nombreuses modifications.

Il est publié conjointement par l'Université de Rennes (Département de mathématiques et informatique) et par l'Ecole Polytechnique Fédérale de Lausanne (Département de Mathématiques) dans le cadre du 3ème Cycle Romand d'Informatique.

Lausanne, Juillet 1975

AVANT - PROPOS

De quoi doit traiter un enseignant lorsqu'il envisage de faire un cours sur les systèmes informatiques. De la programmation des canaux d'entrée-sortie à la syntaxe et sémantique du langage de commande, il y a matière considérable.

Dans le Crocus*, nous avions choisi (en 71/72), après un premier tri, de retenir 6 grands thèmes, que nous rappelons :

(1) - Gestion des Processus (et du parallélisme)
(2) - Gestion de l'information
(3) - Gestion des ressources
(4) - Protection
(5) - Mesures et Modèles
(6) - Méthodologie de conception et de réalisation

La méthodologie de conception est le domaine sur lequel on trouve le plus de bavardages, donc le moins de résultats tangibles. Cette désignation englobe aussi bien les outils et les techniques de conception, écriture et mise au point de gros programmes que l'idée que l'on peut se faire d'un système. Très grossièrement, on dit qu'on conçoit et réalise un système selon une méthode ascendante si le système est vu comme l'entité capable de donner un accès facile à une machine donnée en multiplexant et enrichissant ses ressources. A l'inverse, on dit qu'on conçoit et réalise un système selon une méthode descendante si le système est vu comme une entité définissant certaines fonctions, indépendantes de tel calculateur, à mettre en oeuvre sur un calculateur donné.

---

* Crocus : Les systèmes d'exploitation des Calculateurs. Principes de Conception - Dunos (1975).

Selon la méthode ascendante, le système se compose d'un noyau dont le rôle est de générer des machines virtuelles sensiblement identiques à la machine réelle : ce noyau est donc un multiplexeur de ressources ; en outre il inhibe certaines fonctions et certains objets de la machine réelle et en définit d'autres considérés comme primitifs. Sur chaque machine virtuelle on peut alors construire un système offrant des fonctions analogues à celles qu'offre un système en monoprogrammation.

Selon cette optique, les seuls progrès attendus concernent la méthodologie de conception et de réalisation, ainsi que les techniques de validation qualitative et quantitatives de ces gros programmes, si possible "structurés". Il n'est pas certain que l'on puisse apprendre ou apporter quelque chose de nouveau en ce qui concerne la définition des fonctions et objets d'un système. Il ne fait pas de doute que l'architecture même des machines ne fasse de quelconque progrès.

On peut même aller un peu plus loin. Le multiplexage "aveugle" des ressources, qui consiste à banaliser les agents utilisateurs de ressources, est-il une bonne méthode d'allocation. Considérons les agents "programmes utilisateurs" et les ressources "mémoire centrale". Considérons la séquence de programme consistant en l'exécution de la procédure P suivante :

```
variables A, B ;
procédure P (paramètre X) co utilise la variable
        globale B co
    :
P(A)
    :
```

Si le texte de P occupe les deux pages p_i et p_{i+1} et que A et B appartiennent respectivement aux pages p_a et p_b, l'exécution de cette

séquence avec 3 cases de mémoire centrale entraîne une succession de déroutements imprévisibles lorsque le comportement du programme n'est considéré que comme une banale suite de références confiée à une heuristique (aussi bonne soit-elle) d'allocation. Par contre, l'exécution de cette séquence avec la même zone de mémoire centrale n'entraîne aucun déroutement si l'on a pris soin, à l'appel de la procédure, de recopier dans une tierce page idoine les variables A et B , que l'on prendra soin de remettre à jour au retour de procédure.

La recopie peut évidemment entraîner le chargement des pages contenant A et B , avant et après l'exécution de la procédure. Dans la première méthode ce chargement était nécessaire également pendant l'exécution, et peut-être plusieurs fois. D'autre part, l'exécution de P a pour but de mettre à jour A et B : il est donc probable que A et B servent dès la fin de l'exécution de P .

Ce (contre-) exemple est évidemment construit à dessein. Son seul but est de montrer que le style de schéma d'exécution des programmes, imposé à l'utilisateur (ou aux compilateurs), n'est pas indifférent d'un point de vue système. Si la notion de sous-programme (ou procédure) n'est plus une composition purement logicielle mais une fonction primitive du calculateur, on peut améliorer beaucoup de chose : outre l'allocation de ressources, citons la fiabilité et la protection dues à la modularité ainsi obtenue.

On peut ainsi discuter et revoir de nombreux aspects concernant l'allocation de ressources. Le coût du matériel étant en nette régression par rapport au coût du logiciel, il est clair qu'on peut être tenté de multiplier les ressources critiques : il paraît plus intelligent (et plus prometteur) de tenter de définir un matériel plus approprié au fonctionnement d'un système.

Dans cette optique, il faut donc appréhender d'abord ce qu'est un système, puis tenter de le définir précisément. Ce faisant, on peut espérer mettre

Les deux exemples suivants sont une illustration de la nécessité de reconsidérer la frontière entre les fonctions "langages" et les fonctions "systèmes".

EXEMPLES

1) Au stade de la compilation d'un programme, il est généralement impossible de connaître les emplacements de mémoire que va utiliser le programme généré par le compilateur.
De la même manière, les liens entre programmes, écrits dans des langages différents, posent de tels problèmes qu'ils sont souvent confiés au compilateur de l'un des langages. Par exemple, les procédures "en code" de nombreux compilateurs Algol ou Fortran. Une telle politique est coûteuse et va à l'encontre de la modularité (c'est-à-dire l'indépendance des programmes).
Pour résoudre les problèmes posés dans cet exemple, il suffit de transmettre certaines tâches, dévolues traditionnellement aux compilateurs, à un éditeur de lien. On dit qu'on retarde l'établissement de certaines liaisons (delay binding time). Ainsi un éditeur de lien peut avoir à résoudre des liaisons entre paramètres formels et effectifs.

2) L'expression, dans un langage, des fonctions d'accès aux fichiers a été pratiquement toujours l'objet de l'une des deux démarches suivantes.

   a) Ces fonctions sont définies dans le langage (Modèles Fortran, Entrées-Sorties ISO d'Algol) puis tant bien que mal adaptées aux fonctions d'accès définies par les programmes généraux d'entrées-sorties d'un système particulier.

   b) Ces fonctions ne sont pas définies et les concepteurs du système et du compilateur les définissent à partir de celles qui sont offertes par les programmes généraux d'entrées-sorties.

   Dans les deux cas, la définition est conforme à des modèles traditionnels (un fichier à l'origine sur une bande doit être traité séquentiellement même si, au moment de l'exécution, le système l'a transféré sur un disque ou en mémoire centrale) et n'autorise pas le compilateur à traiter les

accès aux éléments du fichier (en effet, la structure éventuelle du fichier est inconnue du compilateur et les accès aux éléments du fichier font l'objet d'un traitement spécial).

Pour résoudre les problèmes posés par ce second exemple, il semble plus homogène que le système définisse des objets de type "fichier" dont la structure soit harmonisable avec celle des autres objets des langages et dont les fonctions d'accès soient définissables dans ces langages. Une fois de plus, il est alors nécessaire de transférer certaines tâches du compilateur à l'exécution.

Ces observations nous conduisent à regrouper différemment les fonctions d'un système. On dégage ainsi un axe utilisation et un axe mise en oeuvre. Un système définit, par ses fonctions d'utilisation, un langage étendu : ce langage est à la disposition d'un ensemble d'utilisateurs susceptibles de travailler simultanément.

Toutes les fonctions définies par ce langage, ainsi que tous les objets créés grâce à lui ne sont pas forcément utilisables par tous les utilisateurs dans les mêmes conditions. Le langage doit exprimer de telles contraintes.

Pour mettre en oeuvre un langage étendu, on procède en deux temps :

- définition d'un langage primitif extensible à partir duquel on puisse générer le langage étendu. Ce langage primitif définit une machine abstraite ou noyau du système inspiré à priori par les seules considérations de satisfaction des désirs et des droits des utilisateurs.

- définition des programmes qui mettent en oeuvre le noyau sur une machine donnée en utilisant au mieux les ressources physiques de cette machine.

Le concepteur doit donc appréhender en premier lieu les besoins des utilisateurs.