Chapitre 6 LES ENTREES-SORTIES

1. ENTREES-SORTIES PHYSIQUES

Les principes et les caractéristiques des périphériques ont été vus dans le cours de Technologie des Ordinateurs.

Dans certains ordinateurs, des instructions permettent la redirection des données issues de l'UC, non vers la mémoire, mais vers un périphérique (solution INTEL); ou bien un accès mémoire à une adresse réservée particulière est interprétée comme demande d'accès à un périphérique (solution Motorola).

Trois solutions sont utilisées pour faire communiquer l'UC et les périphériques :

1.1 Contrôleurs

Chaque périphérique est connecté à l'ordinateur par l'intermédiaire d'une carte électronique appelée **interface** ou **adaptateur** ou **contrôleur de périphérique**, qui transforme les signaux du périphérique en signaux adaptés à l'UC et vice-versa. Un contrôleur peut gérer un ou plusieurs périphériques. Le SE communique donc avec le contrôleur et non avec le périphérique lui-même. Les petits ordinateurs utilisent des liaison à **bus** entre les contrôleurs, la mémoire et l'UC.

Toutes les données transitent par l'UC qui gère toutes les phases de la transmission, du contrôle et de la synchronisation.

Exemple : imprimantes, clavier/écran sur PC.

1.2 DMA

Pour éviter le ralentissement de l'UC par ces tâches de bas niveau, on utilise parfois l'**accès direct à la mémoire (DMA)**. Il suffit de donner au contrôleur l'adresse où il doit accéder en mémoire, l'opération (lecture ou écriture), le nombre d'octets à transférer entre la mémoire principale et le tampon du contrôleur. Le contrôleur utilise les "temps morts" du bus (**vol de cycle** ou cycle stealing). L'UC conserve la responsabilité des fonctions de commande, de contrôle et une partie de la synchronisation. En fin de transfert, le contrôleur émet une interruption.

Exemple : disque sur PC.

1.3 Canal

Sur les gros ordinateurs, des **canaux d'E/S** allègent le travail du processeur principal pour sa communication avec les contrôleurs (contrôle et synchronisation). Un canal d'E/S est un processeur spécialisé qui gère soit un seul périphérique rapide, soit plusieurs périphériques multiplexés. Un canal utilise le DMA.

L'UC envoie au canal l'adresse en mémoire centrale du début du programme de transfert à exécuter (**programme canal**). L'UC peut interrompre l'exécution, scruter l'état d'avancement par consultation de registres du canal. Un canal est une solution adaptée aux périphériques rapides des mini- et gros ordinateurs.

2. PERIPHERIQUES VIRTUELS D'ENTREE-SORTIE

2.1 Mécanisme des périphériques virtuels

L'objectif est de rendre tout programme utilisateur indépendant des types de périphériques , nombreux et en évolution constante, et des contrôles qui en découlent. D'où la création de **périphériques virtuels** ou **flots d'E/S** ou **fichiers d'E/S**. Sous UNIX, par exemple, une console, un disque ou un fichier ont la même interface d'accès.

Les opérations possibles sur un flot sont :
- ouverture et association physique (open en C)
- fermeture et dissociation (close en C)
- lecture (read en C)
- écriture (write en C)
- paramétrage et contrôle (ioctl , seek en C)

Le système gère une table des flots par processus dans le BCP.

La concordance entre un périphérique virtuel et un périphérique physique associé est assurée par un **pilote de périphérique** (device driver) : soit un type de pilote par type de périphérique, soit un seul type de pilotes configurable par des **descripteurs de périphériques**. Un pilote est la seule partie du SE à connaître les registres du contrôleur, les secteurs et leur facteur d'entrelacement, les pistes, les cylindres, les têtes, les déplacements des bras, les moteurs, le temps de positionnement des têtes, etc..., ainsi que le traitement des erreurs.

2.2 Fonctionnement d'un périphérique virtuel

Un processus P demande une E/S avec un périphérique. La correspondance périphérique virtuel-périphérique physique est ainsi résolue :

- appel système : les paramètres de l'appel sont empilés et un appel système (Supervisor Call = requête SVC) est généré

- action du superviseur : il identifie la nature de l'appel et récupère les paramètres (nom du flot, nombre d'octets, adresse de transfert); puis il identifie le pilote concerné et le périphérique physique. Enfin, il appelle le pilote

- le pilote : le contexte du processus actif est commuté après sauvegarde. La procédure de traitement des E/S est exécutée. Ensuite, le pilote envoie une interruption pour le signaler. Le processus initial est rechargé et réactivé.

Comme il doit y avoir possibilité d'E/S simultanées, un pilote doit être réentrant. En outre, chaque pilote gère une table des processus en attente.

3. PROBLEMES D'INTERBLOCAGE DES PERIPHERIQUES

Avec des périphériques partagés (disques, ...), des problèmes d'interblocage peuvent survenir. Citons deux exemples :

- un processus A demande le dérouleur et un processus B demande le traceur de courbe. Les deux demandes sont prises en compte. Puis A demande le traceur sans libérer le dérouleur et B demande le dérouleur sans libérer le traceur

- un processus A verrouille l'enregistrement R1 d'une BD pour éviter les conflits d'accès et un processus B verrouille l'enregistrement R2. Puis chacun essaie de verrouiller l'enregistrement verrouillé par l'autre

COFFMAN, ELPHICK et SHOSHANI ont montré en 1971 qu'il faut réunir 4 conditions pour provoquer un interblocage :

- l'exclusion mutuelle : chaque ressource est soit disponible, soit attribuée à un seul processus

- la détention et l'attente : les processus qui détiennent des ressources peuvent en demander de nouvelles

- pas de réquisition : les ressources obtenues par un processus ne peuvent pas lui être retirées, mais il doit les libérer explicitement

- l'attente circulaire : il doit y avoir au moins deux processus, chacun d'eux attendant une ressource détenue par l'un des autres

Il existe une solution à chacune des 4 conditions précédentes pour prévenir les interblocages :

3.1 Exclusion mutuelle : usage d'un spoule

Considérons une imprimante avec un fichier tampon associé. Un processus peut ouvrir ce fichier et ne rien imprimer pendant des heures, bloquant tous les autres processus voulant accéder à la ressource.

On crée un processus particulier appelé **démon** (daemon) et un répertoire spécial, le **répertoire de spoule** (spool). Pour imprimer un fichier, un processus doit d'abord le lier au répertoire de spoule. Le démon, seul processus à être autorisé à accéder au fichier spécial de l'imprimante, imprime les fichiers du répertoire de spoule.

Une technique identique peut être utilisée pour la transmission de fichiers à travers un réseau (démon de réseau, répertoire de spoule de réseau).

Mais le spoule ne peut pas résoudre tous les problèmes d'exclusion mutuelle (par exemple, spoule plein et processus inachevés) , ni s'appliquer à tous les périphériques. **On se reportera donc aux algorithmes du chapitre précédent.**

3.2 Détention et attente : demande préalable de ressources

Si l'on oblige tout processus à demander à l'avance les ressources nécessaires, on n'activera le processus que lorsque toutes les ressources seront disponibles. Cette condition est bien sûr difficile à mettre en pratique. Une alternative : n'accorder une nouvelle ressource que s'il y a libération des anciennes ressources.

3.3 Pas de réquisition : pas de solution !

3.4 Attente circulaire : l'ordonnancement numérique des ressources

On numérote toutes les ressources (exemple : 1 : imprimante, 2 : traceur, 3 : dérouleur, etc...). Un processus peut demander toutes les ressources qu'il souhaite à condition que les numéros des ressources soient croissants (variante : un processus ne peut demander une ressource de numéro inférieur à la plus haute ressource qu'il détient). Un interblocage se produirait si le processus A demande la ressource i détenue par B et si B demande la ressource k détenue par A. Si i > k, B ne pourra demander k et si i < k, A ne pourra demander i : il n'y aura pas interblocage.

3.5 Attente circulaire : l'algorithme du banquier

En 1965, DIJKSTRA a proposé un algorithme qui permet d'éviter l'interblocage : l'algorithme du banquier, qui reproduit le modèle de prêt à des clients par un banquier.

Considérons les deux tableaux suivants qui résument l'**état d'un ordinateur** à l'instant t :

proc.	P1	P2	P3	P4
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0
total	5	3	2	2

ressources actuellement attribuées

proc.	P1	P2	P3	P4
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

ressources encore demandées

ressources existantes : exist = (6 3 4 2)
ressources disponibles : dispo = exist - total = (1 0 2 0)

5 processus sont actifs (A,B,C,D,E) et il existe 4 catégories de périphériques P1 à P4 (exemple : P4 = imprimante). Le tableau de gauche donne les ressources déjà allouées et le tableau de droite les ressources qui seront encore demandées pour achever l'exécution.

Un état est dit **sûr** s'il existe une suite d'états ultérieurs qui permette à tous les processus d'obtenir toutes leurs ressources et de se terminer. L'algorithme suivant détermine si un état est sûr :

1. Trouver dans le tableau de droite une ligne L dont les ressources demandées sont toutes inférieures à celles de dispo ($L_i <= dispo_i, \forall i$). S'il n'existe pas L vérifiant cette condition, il y a interblocage

2. Supposer que le processus associé à L obtient les ressources et se termine. Supprimer sa ligne et actualiser dispo

3. Répéter 1 et 2 jusqu'à ce que tous les processus soient terminés (l'état initial était donc sûr) ou jusqu'à un interblocage (l'état initial n'était pas sûr)

Ici, par exemple, l'état actuel est sûr car :
- on allouera à D les ressources demandées et il s'achèvera
- puis on allouera à A ou E les ressources demandées et A ou E s'achèvera
- enfin les autres

L'inconvénient de cet algorithme est le caractère irréaliste de la connaissance préalable des ressources nécessaires à l'achèvement d'un processus. Dans bien des systèmes, ce besoin évolue dynamiquement.

4. LES DISQUES

4.1 Caractéristiques physiques

Un disque est logiquement organisé en cylindres. Un cylindre contient autant de pistes qu'il y a de têtes. Chaque piste est divisée en secteurs(un nombre constant et de taille constante quelque soit le rayon de la piste). Un contrôleur de disques peut effectuer des recherches simultanées sur plusieurs disques, ou bien lire ou écrire sur un disque et attendre des données d'un ou plusieurs autres disques.

Pour le système, une adresse sur disque est formée du triplet : (n° de cylindre, n° de piste, n° de secteur).

Le temps moyen de lecture/écriture sur un secteur est égal à la somme de trois durées élémentaires :

- le temps de recherche : positionnement de la tête sur le bon cylindre
- le temps de latence ou délai rotationnel :pour atteindre le bon secteur (en moyenne 1/2 tour de rotation)
- le temps de transfert réel de l'information

4.2 Ordonnancement des accès

Algorithme FCFS (FIFO) : la gestion la plus simple du mouvement du bras du disque consiste à satisfaire les requêtes d'accès dans l'ordre où elles surviennent. Bien entendu, cet algorithme donne généralement de mauvais temps de réponse. Ainsi, si une suite de requêtes concerne les pistes : 1, 36, 16, 34, 9, 12, alors que les têtes sont sur le cylindre 11, il faudra se déplacer au total de 111 cylindres. Il s'agit de l'algorithme First Come First Served.

Algorithme SSF (ou PCD) : ici, le pilote choisit la requête la plus proche de sa position actuelle (Shortest Seek First, plus courte d'abord, variante du PCTR déjà étudié au chapitre 3, § 3). Avec l'exemple ci-dessus, les cylindres seront traités dans l'ordre : 12, 9, 16, 1, 34, 36. On se sera déplacé de 61 cylindres seulement. Mais les requêtes portant sur des cylindres éloignés peuvent être durablement différées, si d'autres requêtes surviennent pendant le traitement de la liste : l'équitabilité peut souffrir de la réduction du temps de réponse.

Algorithme de l'ascenseur (Look) : pour éviter cet inconvénient, on déplace la tête dans une direction donnée en traitant toutes les requêtes rencontrées, puis le sens du balayage s'inverse et on traite les requêtes rencontrées, etc... Dans l'exemple traité, on obtiendrait l'ordre : 12, 16, 34, 36, 9, 1 et un parcours total de 60 cylindres.

Une variante a été proposée par T.J. TEOREY en 1972 (C-Look, C pour circulaire) : la dernière piste est considérée comme adjacente à la première. La tête, lorsqu'elle est arrivée à une extrémité, retourne immédiatement à l'autre sans traiter de requête. On obtiendrait ici l'ordre : 12, 16, 34, 36, 1, 9, soit 68 cylindres parcourus.

Algorithme PCTL : lorsqu'un disque est fortement sollicité, on trouve fréquemment plusieurs références à une même piste ou à un même cylindre. Les requêtes doivent être ordonnées selon le secteur recherché pour réduire le temps de latence. L'algorithme PCTL (plus court temps de latence) traite les requêtes dans l'ordre de défilement des secteurs concernés sous la tête, pour une piste donnée, quel que soit leur ordre d'arrivée. Par exemple, si la tête se trouve au-dessus du secteur n° 2 et que des requêtes concernent les secteurs 11, 5, 8 et 7, on traitera dans l'ordre les secteurs 5, 7, 8,

11ce qui évitera d'attendre plus d'un tour pour traiter les secteurs 5, 8, 7. On peut parfois gagner en efficacité en entrelaçant les secteurs sur les pistes.

Souvent, les modèles probabilistes sont plus réalistes que les modèles déterministes. Dans les problèmes d'ordonnancement, en particulier, les modèles des files d'attente sont souvent performants. Ils sont bien traités dans BEAUQUIER et BERARD, ch. 10.

4.3 Mémoire cache pour les pistes du disque

En général, le temps de recherche est très supérieur au temps de transfert. Il importe donc peu de lire un secteur ou bien une piste complète pour simplifier le fonctionnement du contrôleur. On utilise une **mémoire cache** pour stocker une piste : mémoire à accès rapide, à accès direct par le contrôleur et par l'UC.

Il est préférable d'implanter cette antémémoire dans le contrôleur, plutôt que dans le pilote, pour que le transfert puisse se faire par DMA

4.4 Traitement des erreurs

Le fonctionnement des disques est soumis à de nombreuses sources d'erreurs :

- erreur de programmation (par exemple : accès à un secteur inexistant) nécessitant un arrêt de la requête
- erreur du contrôle de parité (checksum) : on déclare le secteur endommagé si l'erreur persiste au bout de plusieurs essais. On tient à jour un fichier des secteurs endommagés à ne jamais allouer et à ne jamais copier lors d'une sauvegarde
- erreur de positionnement du bras : un programme de recalibrage est lancé
- erreur de contrôleur

4.5 Disque virtuel

Un disque virtuel utilise une portion de la mémoire centrale pour sauvegarder des secteurs. Il convient bien pour stocker programmes et données fréquemment utilisées, qui seront ainsi accessibles très rapidement.

5. LES TERMINAUX

Il existe de très nombreuses variétés de terminaux. C'est donc au pilote de terminal de masquer ces différences pour qu'un programme soit le plus possible indépendant du terminal utilisé.

5.1 Caractéristiques physiques

Du point de vue du SE, on distingue deux catégories de terminaux :

5.1.1 Les terminaux à interface RS-232 standard

Ils communiquent avec une interface série et son dotés d'un connecteur à 25 broches (une broche pour transmettre les données, une pour en recevoir, une pour la masse et quelques autres pour les contrôles). Généralement un UART (Universal Asynchronous Receiver Transmitter) assure la conversion parrallèle-série des signaux et série-parallèle. Le pilote envoie octet par octet à l'UART et se bloque entre deux transferts (jusqu'à 19200 bits/s).

Les terminaux intelligents actuels possèdent un processeur puissant, une mémoire de grande capacité et peuvent télécharger des programmes à partir de l'ordinateur.

5.1.2 Les terminaux mappés en mémoire (memory-mapped terminals)

Ces terminaux ne communiquent pas avec l'ordinateur par une liaison série, mais font partie de l'ordinateur. Ils sont interfacés par une mémoire particulière (RAM vidéo) qui fait partie de la mémoire de l'ordinateur et est adressée par le processeur comme n'importe quelle partie de la mémoire. Cette solution es plus rapide que la précédente. Un autre composant, le contrôleur vidéo, retire des octets de la RAM vidéo et génère le signal vidéo qui contrôle l'affichage à l'écran.

Dans le cas des terminaux bitmap, chaque bit de la RAM vidéo contrôle un pixel de l'écran.

Le clavier est interfacé par une liaison parallèle, voire une liaison série. A chaque frappe de touche et à chaque relâchement, une interruption est déclenchée et le code de l'emplacement de la touche est placé dans un tampon.

5.2 Le logiciel du clavier

Deux types de pilotes sont envisageables :

- le pilote lit chaque octet du tampon, le convertit en code ASCII à l'aide de tables internes, et le transmet sans interprétation, ou plus souvent le stocke dans un tampon : il s'agit d'une approche caractère (**mode raw**)

- le pilote prend en compte tout ce qui se rapporte à l'édition de ce qui précède la validation (par RC ou NL) : BS, etc... et ne transmet que les lignes corrigées : il s'agit d'une approche ligne (**mode cooked**)

Sous UNIX, le mode **cbreak** est un compromis entre les modes raw et cooked : les caractères sont transmis au processus sans attendre la validation (comme en mode raw), mais DEL, CTRL-S, CTRL-Q et CTRL-\ sont traités comme en mode cooked. La fonction **ioctl** permet de changer de mode.

Il existe deux façons de gérer les tampons des claviers :

- le pilote réserve un ensemble de tampons de taille standard. A chaque terminal est associé un pointeur sur le premier tampon utilisé; l'ensemble des tampons associés à un terminal est organisé en une liste chaînée. Cette méthode est utilisée sur les grosses machines possédant beaucoup de terminaux

- au sein de la structure de données associée à chaque terminal dans la mémoire, on implante un tampon propre au terminal. Le pilote est plus simple. Cette solution convient bien aux petits systèmes ou aux PC.

Les pilotes de claviers doivent également s'acquitter de bien d'autres tâches : gérer la présence ou l'absence d'écho à l'écran, gérer les lignes de longueur supérieure à une ligne d'écran, gérer les caractères de tabulation, produire les caractères RC **et** NL lors d'une validation avec une convention donnée, gérer les caractères spéciaux (CTRL-D, CTRL-Q, CTRL-S, DEL, etc....). Sous UNIX, ces caractères

5.3 Le logiciel de l'écran

Dans le cas d'un terminal RS-232, le programme ou la fonction d'écho de la frappe envoie dans le tampon de l'écran la suite des octets à afficher. Si le tampon est plein ou bien si toutes les données

sont transmises, le premier caractère est envoyé au terminal, le pilote est endormi. Une interruption provoque la transmission du caractère suivant, etc...

Dans le cas de terminaux mappés en mémoire, les caractères à afficher sont retirés un à un de l'espace utilisateur et placés dans la RAM vidéo (avec traitement particulier pour certains caractères, comme BELL ou les séquences classiques d'échappement).

Le pilote gère la position courante dans la RAM vidéo, en tenant compte des caractères tels que RC, NL, BS; il gère aussi le défilement (scrolling) lorsqu'un passage à la ligne en bas d'écran survient. Pour cela, il met à jour un pointeur, géré par le contrôleur vidéo, sur le premier caractère de la première ligne à l'écran. Le pilote gère aussi le curseur en tenant à jour un pointeur sur sa position.

Exercices :

1. Soit l'état suivant d'un ordinateur à l'instant t :

proc.	P1	P2	P3	P4
A	3	0	1	1
B	0	1	1	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0
total	5	3	3	2

ressources actuellement attribuées

proc.	P1	P2	P3	P4
A	1	1	0	0
B	0	1	0	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

ressources encore demandées

ressources existantes : exist = (6 3 4 2)
ressources disponibles : dispo = exist - total = (0 1 0)

L'état du système est-il sûr ?
Si le processus E fait la demande (0, 0, 1, 0) le système qui utilise l'algorithme du banquier lui attribuera-t-il la ressource P3 ?

2. On exécute un ensemble S de n processus indépendants en parallèle :

$S = p_1 // p_2 // ... // p_n$

Le processus p_1 est constitué de 4 tâches séquentielles : $p_1 = T_1(1)\ T_2(1)\ T_3(1)\ T_4(1)$
Le processus p_2 est constitué de 2 tâches séquentielles : $p_2 = T_1(2)\ T_2(2)$
Le processus p_3 est constitué de 3 tâches séquentielles : $p_3 = T_1(3)\ T_2(3)\ T_3(3)$

Trois ressources sont disponibles : R_1 en quantité $N_1 = 7$, R_2 en quantité $N_2 = 8$,
R_3 en quantité $N_3 = 6$

A chaque tâche $T_i(k)$, on associe deux événements :

$d_i(k)$: initialisation de la tâche (lecture des paramètres d'entrée, acquisition de ressources, chargement d'information)

$f_i(k)$: terminaison de la tâche (écriture des résultats, libération de ressources)

On note $d_i(k)$ $(r_1, ..., r_n)$ l'événement correspondant à l'initialisation de la tâche $T_i(k)$ avec demande de r_1 ressources R_1 ,..., r_n ressources R_n et $f_i(k)$ $(r_1, ..., r_n)$ l'événement correspondant à la terminaison de la tâche $T_i(k)$ avec libération de r_1 ressources R_1 ,..., r_n ressources R_n .
Le comportement suivant est-il valide :

$w = d_1(1)(1,2,3)\ d_1(3)(1,1,1)\ f_1(1)(0,1,2)\ d_1(2)(0,2,1)\ f_1(2)(0,1,0)\ d_2(2)(1,2,0)$
$f_1(3)(0,0,1)\ d_2(1)(3,2,4)\ f_2(2)(1,3,1)\ f_2(1)(4,2,5)\ d_2(3)(3,0,1)\ d_3(1)(2,3,1)$
$f_3(1)(2,4,0)\ d_4(1)(2,0,0)\ f_2(3)(3,0,1)\ d_3(3)(2,1,0)\ f_3(3)(3,2,0)\ f_4(1)(2,0,1)$

Quelles sont les valeurs maximales de N_1, N_2, N_3 de telle façon à ce que si le mot était valide, il ne le soit plus ? Quelles sont les valeurs minimales de N_1, N_2, N_3 de telle façon à ce que si le mot n'était pas valide, il le soit ?