Travaux dirigés de système

Fabien GARREAU

Janvier 2019

1 Historique et structure d'un ordinateur

Question 1.1 Quel est le rôle d'un système d'exploitation ? Expliquer pourquoi celui-ci n'est pas indispensable dans les appareils électroménagers actuels et dans quelle condition il pourrait devenir nécessaire.

> Un système d'exploitation a deux fonctions principales. La première vise à proposer une abstraction du matériel au programmeur. La seconde vise à gérer les ressources matérielles de l'ordinateur.
> Les appareils électroménagers sont principalement monotâches, il n'est donc pas nécessaire d'avoir un système pouvant gérer plusieurs processus. Il pourrait devenir nécessaire d'ajouter un système d'exploitation dès lors que l'appareil propose des fonctions avancées (accès à internet, commandes vocales, ...)

Question 1.2 Qu'est-ce que le spooling ?

> le spooling est une technique qui consiste à mettre des informations dans une file d'attente (spool) avant de les envoyer à un périphérique. Cette technique est utilisée depuis 1960 pour exploiter des périphériques lents tels que les imprimantes, lecteurs de carte perforée, modems, etc. Source : https ://fr.wikipedia.org/wiki/spooling

Question 1.3 Quels sont les avantages du transistor qui équipe les ordinateurs actuels par rapport à son prédécesseur le tube à vide ? Le transistor évolue-t-il encore aujourd'hui ?

> Le transistor remplace le tube à vide et permet la rentabilité par son faible coût comparé aux tubes à vide. Moins fragiles pour une utilisation informatique, ils se distinguent aussi par leur petite taille (ils suivent la loi de Moore). Moore prédit que le nombre de transistors des microprocesseurs sur une puce de silicium double tous les deux ans. Bien qu'il ne s'agisse pas d'une loi physique mais seulement d'une extrapolation empirique, cette prédiction s'est révélée étonnamment exacte. Entre 1971 et 2001, la densité des transistors a doublé chaque 1,96 année. En conséquence, les machines électroniques sont devenues de moins en moins coûteuses et de plus en plus puissantes. Source : https ://fr.wikipedia.org/wiki/Loi_de_Moore

Question 1.4 Lister la liste des composants nécessaires au fonctionnement d'un ordinateur actuel ?

> Carte mère, Processeur, Disque Dur/SSD, Mémoire vive (RAM), Écran, Clavier, Souris. Optionnel : Carte graphique

Question 1.5 Classer les éléments suivant en fonction de leur rapidité d'accès à la mémoire. Disque dur/Registre/-Lecteur DVD-ROM/Cache/Mémoire centrale (RAM)/SSD. Donner un ordre de grandeur de capacité et de temps d'accès pour chaque élément.

> Lecteur DVD-ROM (8Go|150ms) < Disque Dur (1To|10ms) < SSD (256Go|0.1ms) < Mémoire centrale (RAM) (8Go|10ns) < Cache (4Mo|2ns) < Registre (<1ko|1ns).

Question 1.6 Quelle est la particularité d'une mémoire volatile ? Quel matériel physique utilise principalement de la mémoire volatile ?

Mémoire volatile ou non et mémoire vive. La mémoire volatile nécessite un courant électrique pour être conservée, une fois l'appareil éteint la mémoire est effacée, c'est le cas de la mémoire d'un processeur ou encore de la RAM. La mémoire est non volatile si elle est conservée lorsque l'appareil n'est pas alimenté, c'est le cas des disques DVD-ROM Blu-Ray ou encore des disques durs et des SSD. La mémoire vive est une mémoire stockée sur un matériel non mécanique, c'est le cas du SSD, de la RAM ou encore des clés USB. Les données sont stockées sur un circuit électronique, une mémoire vive peut-être volatile mais pas obligatoirement.

Question 1.7 Qu'est-ce que la multiprogrammation ? Dans un système d'exploitation multiprogrammé et à temps partagé, plusieurs utilisateurs sont susceptibles d'utiliser le même matériel au même moment. Cela conduit à des problèmes de sécurité.

— Lister quelques-uns de ces problèmes
— Proposer quelques mesures de protection.

La multiprogrammation est l'aptitude d'un système d'exploitation à exécuter plusieurs programmes (applications) simultanément (sur plusieurs processeurs ou plusieurs cœurs d'un processeur), ou à donner l'illusion d'un traitement simultané d'applications par le biais de l'ordonnancement préemptif des applications. La multiprogrammation (ou le multitâche) a été introduite afin d'augmenter les performances du système. En effet, les applications passent beaucoup de temps à attendre des signaux d'entrée sortie, et la multiprogrammation permet au système d'exploitation d'attribuer le ou les processeurs à d'autres applications pendant que l'application initiale attend des données issues du disque dur ou des entrées sorties (réseau, clavier).
Les problèmes de sécurité découlent du fait qu'un processus de l'utilisateur 1 peut écrire des données en mémoire centrale, causant des dommages sur les processus des autres utilisateurs. Un processus espion peut également lire et transmettre des données (fichiers stockés sur le disque) d'un utilisateur.
Les mesures de protections possibles sont par exemple la définition d'un espace d'adressage pour chaque processus (impossible de lire ou d'écrire au delà de cet espace), et le recours à des appels systèmes pour des opérations engageant la sécurité du système : changement de mot de passe, accès aux entrées sorties.

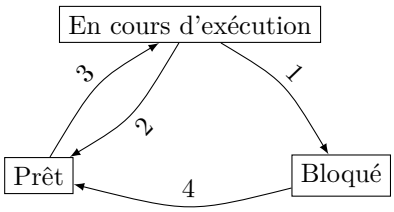
2 Processus

Question 2.1 Quelle est la différence essentielle entre un programme et un processus ?

Un programme est un code statique écrit dans un langage de programmation (C, Java, Pascal etc). Un processus est l'implémentation de ce programme, il est dynamique (il a une date de début et une date de fin). Le même programme peut donner lieu à plusieurs processus, simultanés ou non.

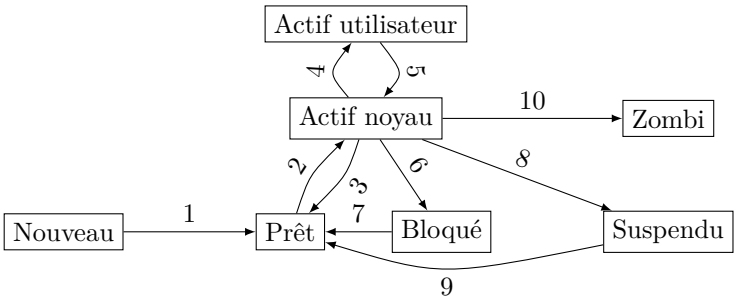
Question 2.2 Décrire, à l'aide d'un diagramme, les différents états d'un processus et la condition pour passer d'un état à un autre. On donnera le diagramme simplifié correspondant au cas général, puis le diagramme des états des processus d'UNIX.

Diagramme à trois états



- 1 : le processus en cours d'exécution est bloqué car une donnée (ou ressource) n'est pas disponible
- 2 : le processus en cours d'exécution est interrompu par l'ordonnanceur
- 3 : le processus prêt est choisi par l'ordonnanceur pour être exécuté
- 4 : la donnée (ou ressource) manquante est disponible, le processus n'est plus bloqué, il attend d'être sélectionné par l'ordonnanceur.

Diagramme des processus d'UNIX



1. Allocation de ressources (mémoire)
2. Élu par l'ordonnanceur
3. Interrompu par l'ordonnanceur (round robin)
4. Passage en mode utilisateur, après un appel système ou une interruption
5. Passage en mode noyau (mode système) suite à une interruption ou pour faire un appel système
6. Le processus se met en attente d'un événement (E/S, `sleep`, `wait`)
7. L'événement attendu est arrivé
8. Processus suspendu par le signal `SIGSTOP`
9. Processus réveillé par le signal `SIGCONT`
10. Fin d'exécution

Question 2.3 Énumérer les différentes causes pouvant provoquer l'interruption du processus exécuté par le processeur, et décrire dans chaque cas comment il s'effectue.

Il y a deux causes possibles :

— L'ordonnanceur estime que le temps alloué au processus en cours d'exécution est désormais suffisant (dans le cadre du temps partagé), ou bien suite à la réception d'un signal (d'une interruption) non masquée, un autre processus, correspondant à l'interruption, est lancé à la place du processus courant.

— Le processus en cours se bloque suite à un appel système bloquant émis par le processus en cours d'exécution. Cela se produit lorsqu'une donnée est manquante (absente du cache), et qu'un accès disque est nécessaire. Pour éviter d'attendre que cette donnée soit disponible, le processus courant est bloqué.

Dans tous les cas, le contexte du processus est sauvegardé (descripteurs de fichiers, état de la mémoire et des registre) afin de pouvoir être restauré plus tard, quand le processus sera à nouveau exécuté.

Question 2.4 Énumérer et discuter les différentes solutions possibles concernant la survie des processus fils, lorsque le processus père disparaît.

On peut envisager deux réactions :
- Le système détruit immédiatement tous les processus fils. Avantage : pas de processus encombrant le système si le père, à l'origine de la demande, disparaît. Inconvénient : perte possible de données (fichiers ouverts par le père et les fils.
- Le système laisse les processus fils se dérouler jusqu'à leur terme (ils deviennent alors des zombies, et seul le système peut les tuer). C'est ce que fait UNIX. Avantage : les processus fils vont à leur terme, et les fichiers ouverts peuvent ainsi être fermés par les fils. Pas de perte de données. Inconvénient : les processus zombies restent actifs jusqu'au redémarrage du système, ou doivent être tués par l'administrateur système.

Question 2.5 Définir la différence entre l'ordonnancement avec réquisition et sans réquisition. Expliquer pourquoi l'ordonnancement sans réquisition a peu d'utilité dans un système réel.

L'ordonnancement avec préemption (ou avec réquisition) permet d'interrompre un processus au motif qu'il a utilisé le processeur pendant une durée jugée suffisante par l'ordonnanceur. Un processus n'effectuant pas d'opération d'E/S (seulement du calcul) monopoliserait le processeur dans un ordonnancement sans préemption. Il est clair qu'une telle situation interdit le temps partagé et l'obtention de garanties temps réel, c'est la raison pour laquelle l'ordonnancement préemptif est très souvent utilisé. En particulier pour mettre en place le tourniquet.

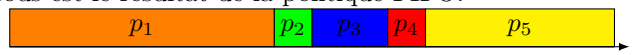
Question 2.6 Soient les processus suivants à exécuter (voir table 1) :

Processus	Temps CPU	Priorité
1	7	3
2	1	1
3	2	3
4	1	4
5	5	2

TABLE 1 – Données sur les processus à exécuter (1 = faible priorité, 4 = haute priorité)

On suppose que les processus sont arrivées dans l'ordre 1, 2, 3, 4, 5.
- Donner un diagramme de Gantt qui illustre l'exécution de ces tâches en utilisant le « premier arrivé premier servi », le tourniquet (quantum=1), le plus court d'abord, et un algorithme avec priorité (sans réquisition).
- Quelle est la date de fin de chaque processus avec chacun des algorithmes ci-dessus ?
- Quel est le temps d'attente de chaque processus avec les algorithmes ci-dessus ?
- Lequel de ces ordonnancements produit le plus petit temps d'attente moyen sur l'ensemble des processus ?

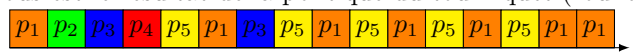
— L'ordonnancement ci-dessous est le résultat de la politique FIFO.



Processus	P_1	P_2	P_3	P_4	P_5
Date de fin	7	8	10	11	16
Attente	0	7	8	10	11

Le temps d'attente moyen des processus est de 7.2 unités de temps avec la politique FIFO.

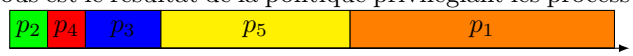
— L'ordonnancement ci-dessous est le résultat de la politique du tourniquet (round robin).



Processus	P_1	P_2	P_3	P_4	P_5
Date de fin	16	2	7	4	14
Attente	9	1	5	3	9

Le temps d'attente moyen des processus est de 5.4 unités de temps avec la politique du tourniquet.

— L'ordonnancement ci-dessous est le résultat de la politique privilégiant les processus les plus courts d'abord.



Processus	P_1	P_2	P_3	P_4	P_5
Date de fin	16	1	4	2	9
Attente	9	0	2	1	4

Le temps d'attente moyen des processus est de 3.2 unités de temps avec la politique du processus le plus court d'abord (solution optimale car $< C >$ et $< W >$ sont équivalents.

— L'ordonnancement ci-dessous est le résultat de la politique privilégiant les processus les plus prioritaires.

Processus	P_1	P_2	P_3	P_4	P_5
Date de fin	8	16	10	1	15
Attente	1	15	8	0	10

Le temps d'attente moyen des processus est de 6,8 unités de temps avec la politique du processus le plus court d'abord.

Question 2.7 Les priorités des processus prêts sont généralement recalculées au cours du temps, expliquer pourquoi.

— On l'a vu à l'exercice précédant, la durée d'utilisation du CPU peut varier au cours du temps, en fonction des instructions constituant le code d'un processus. L'estimation de ce temps peut varier au cours du temps et avoir un impact sur la priorité du processus.

— Sous UNIX, la priorité d'un processus baisse quand il a bénéficié plusieurs fois du processeur, afin de privilégier les processus courts (interactifs).

Question 2.8 Donner une équation reliant le temps de traitement, le temps d'attente et le temps de calcul d'un processus.

temps de traitement = temps d'attente + temps de calcul

3 Gestion des E/S et ordonnancement du disque

Question 3.1 Qu'est-ce qu'un contrôleur de périphérique ? Qu'est-ce qu'un pilote de périphérique (device driver) ?

Un contrôleur de périphérique est un circuit électronique chargé de commander un périphérique (une imprimante, un clavier).
Un pilote de périphérique est un programme qui permet au système d'exploitation de communiquer avec le contrôleur du périphérique en question, en particulier pour lire et écrire des données.

Question 3.2 Un disque comporte 200 pistes numérotées de 0 à 199. La dernière requête traitée concernait la piste 125 et une requête pour la piste 143 est en cours. La liste des nouvelles requêtes dans l'ordre d'arrivée est la suivante : 86, 147, 91, 177, 94, 150, 102, 175, 130.

Calculer le déplacement total de la tête pour chacun des algorithmes suivants : FCFS (First Come First Served), SSF (Shortest Seek First), SCAN (algorithme de l'ascenseur : le bras se déplace dans un sens et n'en change que lorsque plus aucune piste ne peut être trouvée en déplaçant le bras dans la direction courante) et C-SCAN (Circular-SCAN : identique à SCAN, mais les numéros de piste sont « circulaires » : 199 + 1 = 0).

- **FCFS** Les déplacements sont de $|143-86|+|86-147|+|147-91|+|91-177|+|177-94|+|94-150|+|150-102|+|102-175|+|175-130| = 565$
- **SSF** On commence par servir les requêtes les plus proches : $|143-147|+|147-150|+|150-130|+|130-102|+|102-94|+|94-91|+|91-86|+|86-175|+|175-177| = 162$
- **SCAN** Le sens du bras est montant, puisqu'on est passé de la piste 125 à la piste 143. On servira donc les requêtes dans l'ordre $(147, 150, 175, 177, (down), 130, 102, 94, 91, 86)$. Ce qui conduit à un déplacement total égal à $(177-143)+(177-86) = 125$ pistes.
- **C-SCAN** Le sens du bras est montant, puisqu'on est passé de la piste 125 à la piste 143. On servira donc les requêtes dans l'ordre $(147, 150, 175, 177, (down), (up), 86, 91, 94, 102, 130)$. Ce qui conduit à un déplacement total égal à $(177-143)+(177-86)+(130-86) = 169$ pistes.

Question 3.3 Aucun algorithme d'ordonnancement du disque à l'exception de FCFS n'est vraiment équitable, expliquer pourquoi. Pourquoi SCAN est-il plus équitable que SSF ? Pourquoi C-SCAN est-il plus équitable que SCAN ? Pourquoi l'équité (fairness) constitue-t-elle un objectif important dans un système à temps partagé ?

- On dira que le système est équitable si la requête 1 arrivée à $t_1 < t_2$ est toujours servie avant la requête 2 arrivée à la date t_2. Si l'on veut respecter cette contrainte, on n'a pas d'autre choix que de les traiter dans leur ordre d'arrivée.
- SCAN est plus équitable que SSF car s'il y a beaucoup de requêtes qui arrivent en temps réel le bras aura tendance à rester au milieu du disque avec SSF, les requêtes relatives aux pistes situées aux extrémités attendrons beaucoup plus longtemps. Avec SCAN, ce problème disparaît, car la tête continuera de s'éloigner du centre tant qu'il y a des requêtes situées plus loin.
- C-SCAN est plus équitable que SCAN parce qu'avec SCAN, les requêtes situées sur des pistes intermédiaires (au milieu du disque) ont plus de chance d'être traîtées rapidement, parce que le centre du disque est visité à l'aller et au retour pour une seule visite de l'extrémité du disque.
- L'équité est un critère important en temps partagé parce que les requêtes proviennent d'utilisateurs différents, qui doivent bénéficier de la même qualité de service.

Question 3.4 Quelles sont les stratégies possibles concernant l'arrivée des nouvelles requêtes au niveau du disque ? Pourquoi est-il dangereux d'autoriser le traitement immédiat de nouvelles requêtes ?

On peut stocker ces requêtes dans l'ordre où elles sont arrivées, et donner la priorité aux plus anciennes ou au plus récemment arrivées. On peut aussi tenir compte de priorités données pour chaque requête.
Le traitement immédiat des nouvelles requêtes est susceptible de provoquer la famine : les anciennes requêtes ne sont jamais satisfaites tant que de nouvelles requêtes surviennent.

Question 3.5 Soit un disque contenant 32 blocs numérotés de 0 à 31. Le bloc 20 est défectueux. Le disque contient les 4 fichiers suivants avec l'allocation contiguë.

Fichier	Bloc de début	Taille
`file1`	2	4
`monfichier`	9	7
`data`	23	2
`log`	27	4

Donner la représentation de l'espace libre à l'aide de la technique du vecteur binaire, de la liste chaînée, et de la liste de blocs contigus.

On peut représenter le disque et son utilisation (cases grisées) comme suit :

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

Le vecteur binaire pour l'espace libre vaut :

`00111100011111110000100110011110`

La liste chaînée conduit à stocker l'adresse du premier bloc libre `0` et à ajouter les pointeurs suivants :

0 1	1 6	2	3	4	5	6 7	7 8
8 16	9	10	11	12	13	14	15
16 17	17 18	18 19	19 21	20	21 22	22 25	23
24	25 26	26 31	27	28	29	30	31 NULL

La liste des blocs libres contigus est :

Zone libre	Début	Taille
1	0	2
2	6	3
3	16	4
4	21	2
5	25	2
6	31	1

4 Gestion de la mémoire

Question 4.1 On considère un système à swapping dont la mémoire contient des zones libres (ordonnées en fonction des adresses mémoire) de 10K, 4K, 20K, 18K, 7K, 9K, 12K, et 15K. Quelles zones l'algorithme *first-fit* sélectionne-t-il pour les demandes successives d'allocation de segments suivantes : 12K, 10K, puis 9K ? Même question pour l'algorithme du meilleur ajustement (*best-fit*), et l'algorithme de la zone libre suivante (*next-fit*).

L'algorithme « premier ajustement » (*first fit*) sélectionne le premier bloc libre de la liste pouvant satisfaire la demande d'allocation courante.

- La troisième zone libre (20K) est sélectionnée pour répondre à la première demande de 12K. La nouvelle liste des zones libres est alors 10K, 4K, 8K, 18K, 7K, 9K, 12K, et 15K.
- La première zone libre (10K) est sélectionnée pour répondre à la seconde demande de 10K. La nouvelle liste des zones libres est alors 4K, 8K, 18K, 7K, 9K, 12K, et 15K.
- La troisième zone libre (18K) est sélectionnée pour répondre à la troisième demande de 9K. La nouvelle liste des zones libres est alors 4K, 8K, 9K, 7K, 9K, 12K, et 15K.

L'algorithme « meilleur ajustement » (*best fit*) sélectionne le plus petit bloc libre permettant de répondre à la demande d'allocation.

- La septième zone libre (12K) est sélectionnée pour répondre à la première demande de 12K. La nouvelle liste des zones libres est alors 10K, 4K, 8K, 18K, 7K, 9K, et 15K.
- La première zone libre (10K) est sélectionnée pour répondre à la seconde demande de 10K. La nouvelle liste des zones libres est alors 4K, 8K, 18K, 7K, 9K, et 15K.
- La cinquième zone libre (9K) est sélectionnée pour répondre à la troisième demande de 9K. La nouvelle liste des zones libres est alors 4K, 8K, 18K, 7K, et 15K.

L'algorithme « ajustement suivant » (*next fit*) est une variante de *first fit*. Au lieu de prendre la liste des zones libres depuis le début à toute nouvelle demande d'allocation, on reprend cette liste à partir de la position atteinte lors de la dernière allocation. L'idée est d'éviter d'avoir à tester de nombreux espaces très petits au début de la liste, situation qui a tendance à se produire avec *first fit*.

- La troisième zone libre (20K) est sélectionnée pour répondre à la première demande de 12K. La nouvelle liste des zones libres est alors 10K, 4K, 8K, 18K, 7K, 9K, 12K, et 15K.
- La quatrième zone libre (18K) est sélectionnée pour répondre à la seconde demande de 10K. La nouvelle liste des zones libres est alors 10K, 4K, 8K, 8K, 7K, 9K, 12K, et 15K.
- La sixième zone libre (9K) est sélectionnée pour répondre à la troisième demande de 9K. La nouvelle liste des zones libres est alors 10K, 4K, 8K, 8K, 7K, 12K, et 15K.

Question 4.2 On considère la table de segments suivante.

Segment	Base	Longueur
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

Dans la notation (x, y) d'une adresse logique, x désigne le numéro du segment et y le décalage par rapport à la base du segment. Quelles sont les adresses physiques correspondant aux adresses logiques suivantes ?

$(0, 430)$	$(1, 10)$	$(1, 11)$	$(2, 500)$	$(3, 400)$	$(5, 16)$	$(4, 112)$

- $(0, 430)$ désigne l'adresse physique 219+430 = 649.
- $(1, 10)$ désigne l'adresse physique 2300+10 = 2310.
- $(1, 11)$ désigne l'adresse physique 2300+11 = 2311.
- $(2, 500)$ provoque une erreur d'adressage, car le décalage (500) est supérieur ou égal à la longueur (100).
- $(3, 400)$ désigne l'adresse physique 1327+400 = 1727.
- $(5, 16)$ provoque une erreur d'adressage, car le segment 5 n'existe pas dans la table.
- $(4, 112)$ provoque une erreur d'adressage, car le décalage (112) est supérieur ou égal à la longueur (96).

Question 4.3 Qu'est-ce que la pagination à la demande ? Quand se produisent les défauts de pages ? Est-ce que le défaut doit être signalé à l'utilisateur ?

`http://deptinfo.cnam.fr/Enseignement/CycleA/AMSI/cours_systemes/14_gestion_memoire/gestio.htm`
La **pagination à la demande** consiste à ne charger en mémoire centrale qu'une partie de l'espace d'adressage d'un processus. Cela permet d'exécuter des programmes dont le code ou les données ne pourraient être entièrement contenus en mémoire centrale, mais également de faire cohabiter plusieurs processus dont les besoins cumulés en matière de mémoire dépassent la capacité de la mémoire centrale.

Un **défaut de page** se produit lorsqu'un processus actif a besoin d'accéder à du code ou à des données qui ne sont pas chargés en mémoire centrale. Un défaut de page ne doit pas être signalé à l'utilisateur : le système d'exploitation bloque le processus et effectue une demande d'E/S en direction de la mémoire auxiliaire (le disque dur) afin que la page manquante soit copiée en mémoire centrale. Une fois cette page copiée, le processus passe à l'état prêt.

Question 4.4 On considère la liste suivante de références de pages :

```
(1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6)
```

On suppose que la mémoire centrale possède 3 cadres. Combien de défauts de pages observe-t-on pour les algorithmes de remplacement suivants, sachant que tous les cadres sont initialement vides ?

— LRU (Least Recently Used)
— Seconde chance
— Optimal

— LRU (Least Recently Used)
On construit la matrice 3×3, où chaque rangée représente une page réelle (il y a 3 cadres). Les trois premières références `1, 2, 3` conduisent à affecter la page virtuelle i à la page réelle i :

0	0	0	1
1	0	0	2
1	1	0	3

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3																	

La référence à `4` conduit à choisir la page réelle 1 comme victime

0	1	1	4
0	0	0	2
0	1	0	3

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1																

La référence à `2` conduit mettre à jour la matrice comme suit :

0	0	1	4
1	0	1	2
0	0	0	3

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2															

La référence à `1` conduit à choisir la page réelle 3 comme victime

0	0	0	4
1	0	0	2
1	1	0	1

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3														

La référence à `5` conduit à choisir la page réelle 1 comme victime

0	1	1	5
0	0	0	2
0	1	0	1

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1													

La référence à `6` conduit à choisir la page réelle 2 comme victime

0	0	1	5
1	0	1	6
0	0	0	1

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2												

La référence à `2` conduit à choisir la page réelle 3 comme victime

0	0	0	5
1	0	0	6
1	1	0	2

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2	3											

La référence à `1` conduit à choisir la page réelle 1 comme victime

0	1	1	1
0	0	0	6
0	1	0	2

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2	3	1										

La référence à `2` conduit à la mise à jour de la matrice

0	1	0	1
0	0	0	6
1	1	0	2

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2	3	1	3									

La référence à 3 conduit à choisir la page réelle 2 comme victime

0	0	0	1
1	0	1	3
1	0	0	2

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2	3	1	3	2								

La référence à 7 conduit à choisir la page réelle 1 comme victime

0	1	1	7
0	0	1	3
0	0	0	2

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2	3	1	3	2	1							

La référence à 6 conduit à choisir la page réelle 3 comme victime

0	1	0	7
0	0	0	3
1	1	0	6

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2	3	1	3	2	1	3						

La référence à 3 conduit à la mise à jour de la matrice

0	0	0	7
1	0	1	3
1	0	0	6

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2	3	1	3	2	1	3	2					

La référence à 2 conduit à choisir la page réelle 1 comme victime

0	1	1	2
0	0	1	3
0	0	0	6

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2	3	1	3	2	1	3	2	1				

La référence à 1 conduit à choisir la page réelle 3 comme victime

0	1	0	2
0	0	0	3
1	1	0	1

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2	3	1	3	2	1	3	2	1	3			

La référence à 2 conduit à la mise à jour de la matrice

0	1	1	2
0	0	0	3
0	1	0	1

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2	3	1	3	2	1	3	2	1	3	1		

La référence à 3 conduit à la mise à jour de la matrice

0	0	1	2
1	0	1	3
0	0	0	1

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2	3	1	3	2	1	3	2	1	3	1	2	

La référence à 6 conduit à choisit la page réelle 3 comme victime

0	0	0	2
1	0	0	3
1	1	0	6

1	2	3	4	2	1	5	6	2	1	2	3	7	6	3	2	1	2	3	6
1	2	3	1	2	3	1	2	3	1	3	2	1	3	2	1	3	1	2	3

12 défauts de page détectés.

— Déroulement de l'algorithme de la seconde chance, on construit d'abord la liste des pages (dernière page arrivée par la droite) :

0	0	0
1	2	3

Demande de la page 4 (1 est le premier arrivé donc le premier sorti, décalage de 2 et 3 sur la gauche) :

0	0	0
2	3	4

Demande de la page 2 (2 est déjà présente bit de référence passe à 1 :

1	0	0
2	3	4

Demande de la page 1 (2 a le droit à une seconde chance, son bit de référence et réinitialisé à 0 et 2 est placé à la fin de la liste, 3 est le premier arrivé donc le premier sorti, décalage de 4 et 2 sur la gauche et ajout de 1)

0	0	0
4	2	1

L'algorithme continue de la même manière pour les demandes suivantes dans l'ordre : (5, 6, 2, 1, 2, 3, 7)

0	0	0
2	1	5

0	0	0
1	5	6

0	0	0
5	6	2

0	0	0
6	2	1

0	1	0
6	2	1

1	0	0
2	1	3

0	0	0
3	2	7

L'algorithme continue de la même manière pour les demandes suivantes dans l'ordre : (6, 3, 2, 1, 2, 3, 6)

0	0	0
2	7	6

0	0	0
7	6	3

0	0	0
6	3	2

0	0	0
3	2	1

0	1	0
3	2	1

1	1	0
3	2	1

0	0	0
3	2	6

— Déroulement de l'algorithme optimal, on construit d'abord la liste des pages :

1	2	3

On cherche à remplacer la page dont le premier appel apparaît le plus tard dans la liste :

(2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6)

Pour la demande de la page 4 c'est la page 3 qui apparaît le plus tardivement.

1	2	4

On continue de la même manière pour les demandes suivantes dans l'ordre : (2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6)

1	2	4

$X2$

1	2	5

$X1$

1	2	6

$X4$

3	2	6

$X1$

3	7	6

$X3$

3	2	6

$X1$

3	2	1

$X3$

6	2	1

$X1$

5 Interblocages

Question 5.1 A quoi servent les conditions de Bernstein ? Donner un exemple de leur utilisation.

Les conditions de Bernstein servent à vérifier si l'on peut diviser une fonction séquentielle en plusieurs parties indépendantes afin d'exécuter ces parties en parallèle. Soit la fonction écrite en C++ effectuant les opérations suivantes :

```
void f(int a, int b, int &c, int &d)
{
c = a + b;
d = 2*a;
}
```

On peut la décomposer en deux parties, `f1` et `f2` pouvant être exécutées en parallèle :

```
void f1(int a, int b, int &c)
{
c = a + b;
}
```

et

```
void f2(int a, int b, int &d)
{
d = 2*a;
}
```

En effet, les trois conditions de Bernstein sont satisfaites : la première stipule que `f1` n'a pas d'entrées parmi les sorties de `f2`, la seconde stipule que `f2` n'a pas d'entrées parmi les sorties de `f1`, la dernière dispose que `f1` et `f2` n'ont pas de sorties en commun.

Question 5.2 Donner quelques exemples d'interblocage qui ne sont pas liés à l'informatique.

La réservation de ressources croisées dans les ateliers de production : deux boulangers ont besoin d'un pain et d'un couteau pour produire du pain tranché. Le premier prend le couteau, le second prend le pain, mais aucun des deux ne peut produire de pain tranché.

Quatre véhicules venant respectivement de l'est, du nord, de l'ouest et du sud veulent traverser un carrefour découpé en quatre zones carrées. La zone N est occupée par le véhicule venant du nord, la zone Z par le véhicule venant du sud, la zone E par le véhicule venant de l'est et la zone O par le véhicule venant de l'ouest.

- le véhicule venant de l'est (en zone E) attend que le véhicule venant du nord traverse le carrefour pour pouvoir s'y engager (il demande la zone N)
- le véhicule venant du nord (en zone N) attend que le véhicule venant de l'ouest traverse le carrefour pour pouvoir s'y engager (il réclame la zone O)
- le véhicule venant de l'ouest (en zone O) attend que le véhicule venant du sud traverse le carrefour pour pouvoir s'y engager (il réclame la zone S)
- le véhicule venant du sud (en zone S) attend que le véhicule venant de l'est traverse le carrefour pour pouvoir s'y engager (il réclame la zone E)

Question 5.3 Les quatre conditions nécessaires d'interblocage sont-elles indépendantes ?

Non, la condition 2) « Détention et attente » implique la condition 4) « Attente circulaire ».

Question 5.4 Est-il possible d'observer un interblocage n'impliquant qu'un seul processus ?

Non, la condition d'attente circulaire, pour qu'un interblocage se déclare, ne peut pas être satisfaite avec un seul processus. Un processus seul peut entrer dans une boucle infinie, mais il ne s'agit pas d'un interblocage.

Question 5.5 Dans un système où les processus sont exécutés séquentiellement, l'un après l'autre, peut-il y avoir un interblocage ?

Non, car à tout instant, un seul processus est en cours d'exécution, les autres n'ayant pas démarré. Toutes les ressources du système sont donc disponible pour ce processus, qui ne peut pas subir d'interblocage puisqu'aucun autre processus n'a pu réserver de ressources.

Question 5.6 Un système peut-il se trouver dans un état qui ne soit ni sûr, ni interbloqué ? Si oui, donner un exemple d'un tel état, si non, démontrer que tous les états sont soit des états sûrs, soit des interblocages.

Un système est soit dans un état sûr, soit dans un état non sûr. Un état non sûr n'est pas nécessairement un interblocage, mais il peut y conduire sans que l'on puisse l'empêcher. Un état non sûr n'induit pas nécessairement un interblocage, car des processus peuvent délibérément libérer des ressources, qui peuvent alors permettre d'éviter un interblocage.
Exemple : on a deux boulangers, un pain et un couteau. Le premier boulanger réserve le couteau, puis tire à pile ou face. S'il obtient face, il aiguise le couteau (le pain n'est alors pas requis), avec pile, il réserve le pain pour le couper. Le second boulanger réserve le pain, puis tire à pile ou face. S'il obtient face, il met le pain au four afin de le cuire davantage (auquel cas, le couteau n'est pas requis), avec pile, il requiert le couteau pour couper le pain. Supposons que le premier boulanger réserve le couteau, et le second, le pain. Si l'un des boulanger (ou les deux) tire pile, il n'y a pas d'interblocage. En revanche, si les deux boulanger obtiennent face, le système est interbloqué.
Analogie : une équipe de football dispute trois matches de pool dans un groupe contenant 4 équipes. Un match gagné rapporte 3 points, un match nul 1 point, une match perdu 0 point. Une équipe est qualifiée pour la phase finale si elle est classée première ou deuxième (au nombre total de points) à l'issue des trois matches. Une équipe qui a remporté ses deux premiers matches est qualifiée, quoiqu'il arrive au troisième match (état sûr). On considère l'équipe A, et les résultats suivants pour les deux premiers matches.

Journée	Equipe	Points	Equipe	Points
1	A	0	B	3
1	C	1	D	1
2	A	1	C	1
2	B	3	D	0

A l'issue de ces deux premières journées, le total des points est [1 6 2 1]. L'équipe B est automatiquement qualifiée et assurée de terminer en tête du classement. On suppose que lors de la troisième et dernière journée, l'équipe A bat l'équipe D. le total des points de l'équipe A s'élève alors à 4 points. Le sort de cette équipe (élimination ou qualification pour la phase finale) ne dépend plus d'elle, mais du résultat du match de la troisième journée entre les équipes B et C. En effet, si B bat C (ou que B et C font un match nul), alors C a 2 points (ou 3), et A est qualifiée. A l'inverse, si C bat B, alors C a 5 points, et A est éliminée. L'état de l'équipe A à l'issue de la seconde journée est non sûr, car quoi que cette équipe fasse (même si elle gagne son dernier match), c'est le résultat des autres équipes qui peut décider de son sort.

Question 5.7 On considère un système d'exploitation gérant 4 types de ressources disponibles dans les quantités suivantes : $(3, 14, 12, 12)$. Cinq processus sont actuellement dans l'état suivant :

Processus	Allocation	Maximum
p_0	0 0 1 2	0 0 1 2
p_1	1 0 0 0	1 7 5 0
p_2	1 3 5 4	2 3 5 6
p_3	0 6 3 2	0 6 5 2
p_4	0 0 1 4	0 6 5 6

— Quel est l'état du vecteur *Disponible* et de la matrice *Besoin* de l'algorithme du banquier ?
— Le système est-il dans un état sûr ?
— Si le processus p_1 émet maintenant la demande $(0, 4, 2, 0)$, peut-elle être satisfaite immédiatement ?

— Le vecteur *Disponible* s'obtient par $(3, 14, 12, 12) - \sum_{i=0}^{4} Allocation_i = (1, 5, 2, 0)$. La matrice *Besoin* est égale à $Max - Allocation$, soit

$$Besoin = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 7 & 5 & 0 \\ 1 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 6 & 4 & 2 \end{bmatrix}$$

— Pour savoir si le système est dans un état sûr, on peut soit trouver une séquence de processus sûre, soit appeler `TestSur`. C'est ce que nous faisons ci-après.

`Termine=[0 0 0 0 0]` et `DispoTest=[1 5 2 0]`
`Recherche d'un processus` i `non terminé tel que` $\texttt{Besoin}_i \leq$ `DispoTest`
  $\texttt{Besoin}_0 \leq$`[1 5 2 0]`
  `DispoTest = [1 5 2 0]+[0 0 1 2] = [1 5 3 2]`
  `Termine(0) = vrai`
`Recherche d'un processus` i `non terminé tel que` $\texttt{Besoin}_i \leq$ `DispoTest`
  $\texttt{Besoin}_3 \leq$`[1 5 3 2]`
  `DispoTest = [1 5 3 2]+[0 6 3 2] = [1 11 6 4]`
  `Termine(3) = vrai`
`Recherche d'un processus` i `non terminé tel que` $\texttt{Besoin}_i \leq$ `DispoTest`
  $\texttt{Besoin}_1 \leq$`[1 11 6 4]`
  `DispoTest = [1 11 6 4]+[1 0 0 0] = [2 11 6 4]`
  `Termine(1) = vrai`
`Recherche d'un processus` i `non terminé tel que` $\texttt{Besoin}_i \leq$ `DispoTest`
  $\texttt{Besoin}_2 \leq$`[2 11 6 4]`
  `DispoTest = [2 11 6 4]+[1 3 5 4] = [3 14 11 8]`
  `Termine(2) = vrai`
`Recherche d'un processus` i `non terminé tel que` $\texttt{Besoin}_i \leq$ `DispoTest`
  $\texttt{Besoin}_4 \leq$`[3 14 11 8]`
  `DispoTest = [3 14 11 8]+[0 0 1 4] = [3 14 12 12]`
  `Termine(4) = vrai`
`TestSur retourne vrai`

L'état initial est sûr. On peut également déduire une séquence de processus du déroulement de `TestSur` : $(p_0, p_3, p_1, p_2, p_4)$.

— p_1 émet la $Demande = (0\ 4\ 2\ 0)$. On applique l'algorithme du banquier. L'étape 1 vérifie que la demande n'excède pas la demande maximale (condition satisfaite), et que les ressources disponibles sont suffisantes, ce qui est également le cas. On met à jour $Disponible$, $Allocation_1$ et $Besoin_1$:

```
Disponible = [1 1 0 0]
Allocation1 = [1 4 2 0]
Besoin1 = [0 3 3 0]
```

Enfin, on appelle `TestSur` pour vérifier que la satisfaction de la demande émise par p_1 conduit à un état sûr.

```
Termine=[0 0 0 0 0] et DispoTest=[1 1 0 0]
Recherche d'un processus i non terminé tel que Besoini ≤ DispoTest
      Besoin0 ≤[1 1 0 0]
      DispoTest = [1 1 0 0]+[0 0 1 2] = [1 1 1 2]
      Termine(0) = vrai
Recherche d'un processus i non terminé tel que Besoini ≤ DispoTest
      Besoin2 ≤[1 1 1 2]
      DispoTest = [1 1 1 2]+[1 3 5 4] = [2 4 6 6]
      Termine(2) = vrai
Recherche d'un processus i non terminé tel que Besoini ≤ DispoTest
      Besoin1 ≤[2 4 6 6]
      DispoTest = [2 4 6 6]+[1 4 2 0] = [3 8 8 6]
      Termine(1) = vrai
Recherche d'un processus i non terminé tel que Besoini ≤ DispoTest
      Besoin3 ≤[3 8 8 6]
      DispoTest = [3 8 8 6]+[0 6 3 2] = [3 14 11 8]
      Termine(3) = vrai
Recherche d'un processus i non terminé tel que Besoini ≤ DispoTest
      Besoin4 ≤[3 14 11 8]
      DispoTest = [3 14 11 8]+[0 0 1 4] = [3 14 12 12]
      Termine(4) = vrai
TestSur retourne vrai, l'état est sûr
```

Le processus p_1 peut donc voir sa demande satisfaite.

Question 5.8 Quelles sont les différences essentielles entre interblocage et famine ?

Un interblocage implique nécessairement plusieurs processus (condition 4), alors que la famine peut ne concerner qu'un seul processus. Un interblocage est un état dont on ne peut pas sortir, or une famine peut cesser (lorsque les processus plus prioritaires sont moins nombreux).

6 Processus concurrents

Question 6.1 Montrer qu'un comportement indésirable peut se manifester si les opérations P et V d'un sémaphore sont interruptibles.

On rappelle que l'opération $P(sem)$ a pour code :

```
sem--;
if(sem < 0)
{
mettre en sommeil le processus;
}
```

On suppose qu'un sémaphore appelé `sem` a une capacité initiale de 1. Si un premier processus implémentant P est interrompu à la fin de l'instruction `sem--;` (on a donc `sem` égal à 0), et qu'un second processus implémentant P sur le même sémaphore est activé juste après (`sem` est égal à -1), alors le second processus est mis en sommeil. Puis, lorsque le premier processus sera restauré et reprendra, il sera lui aussi mis en sommeil et sera bloqué. Même chose pour tout processus qui appellera P sur ce sémaphore.

Question 6.2 Indiquer succinctement comment on pourrait définir les sémaphores à partir d'un système d'exploitation qui permet de masquer les interruptions. Expliquer pourquoi on réserve ce type de solutions aux primitives systèmes.

On pourrait implémenter les primitives P et V en s'assurant que toutes les interruptions sont masquées en commençant l'exécution de P et de V, puis qu'elles sont à nouveau prises en compte dès la fin de P et de V.
On réserve ce type de solutions aux primitives systèmes, parce que les processus utilisateurs ne sont pas autorisés à masquer les interruptions. S'ils pouvaient le faire, alors il suffirait qu'un processus utilisateur masque définitivement les interruptions pour que le processeur puisse être accaparé par un processus utilisateur. Une primitive système a donc une durée très courte, afin que le masquage des interruptions soit sans dommage pour le système.

Question 6.3 On considère un système constitué de trois ressources identiques. Chaque processus a besoin de deux ressources pour s'exécuter. Un interblocage est-il possible si deux processus peuvent s'exécuter simultanément ? Même question si trois processus peuvent s'exécuter simultanément.

Avec 2 processus, l'interblocage est impossible. En effet, chaque processus peut toujours obtenir au moins une ressource. Mais dans ce cas, l'autre processus peut obtenir deux ressources, et se terminer.
Avec 3 processus, l'interblocage est possible : si chaque processus réserve une seule ressource, aucun d'eux ne peut se terminer.

Question 6.4 Plus généralement, si le système gère r ressources identiques, que tout processus a besoin de m ressources et qu'il peut y avoir jusqu'à p processus en exécution, donner une condition suffisante de non interblocage.

Le pire cas est le suivant : chaque processus a obtenu $m-1$ ressources, et il n'en reste plus aucune. Ainsi, le système est interbloqué. On évite tout interblocage si $r > p(m-1)$