Systèmes d'exploitation

INF3600

Exercices + Corrigés

Gestion des processus

Exercice 1 :

1) Quel est le rôle d'un système d'exploitation ? Les interpréteurs de commandes et les compilateurs font-ils parties du système d'exploitation ?

2) Qu'est ce qu'un système multiprogrammé ? Un système de traitement par lots ? Un système en temps partagé ?

3) Dans le système UNIX, les véritables appels système sont effectués à partir
   - d'un programme utilisateur
   - d'une commande shell
   - d'une procédure de la bibliothèque standard

   Sont-ils exécutés en mode superviseur ou en mode utilisateur ?

4) Comment sont organisés les fichiers dans le système UNIX ? Un utilisateur peut-il accéder à un fichier d'un autre utilisateur ? Si oui, comment ?

5) Dans le système UNIX, est-ce que tout processus a un père ? Que se passe-t-il lorsqu'un processus devient orphelin (mort de son père) ? Quand est-ce un processus passe à l'état Zambie ?

6) Pour lancer en parallèle plusieurs traitements d'une même application, vous avez le choix entre les appels système fork( ) et

pthread_create( ). Laquelle des deux possibilités choisir ? pourquoi ?

7) Citez quatre événements qui provoquent l'interruption de l'exécution d'un processus en cours, dans le système UNIX.

8) Quel est le rôle de l'ordonnanceur ? Décrire brièvement l'ordonnanceur du système UNIX ? Favorise-t-il les processus interactifs ?

9) Pourquoi le partage de données pose des problèmes dans un système multiprogrammé en temps partagé ? Le système UNIX permet-il de contrôler les accès aux données partagées ? Qu'est-ce qu'une section critique ?

Exercice 2 :

Que fait chacun des programmes suivants :

1)

```
int main( )
{
        int p=1 ;
        while(p>0) p=fork() ;
        execlp("prog", "prog", NULL) ;
        return 0 ;
}
```

2)

```
int     i=2 ;
int main ( )
{
    j=10;
    int p ;
   while(i-- && p = fork())
   if(p<0) exit(1) ;
```

```
        j += 2;
        if (p == 0)
        {       i *= 3;
                j *= 3;
        }
        else
        {
                i *= 2;
                j *= 2;
        }
        printf(« i=%d, j=%d », i,j) ;
        return 0 ;
}
```

3)

```
#include <stdio.h>
#include <unistd.h>
int main ( )
{       int fd[2], i=2;
        char ch[100];
        while ( i)
        {       pipe(fd);
                if( fork())
                {       close(fd[0]);
                        dup2(fd[1],1);
                        close(fd[1]);
                        break;
                } else
                {       close(fd[1]);
                        dup2(fd[0],0);
                        close(fd[0]);
                }
           i--;
        }
        scanf("%s", ch);
        printf("%s\n",ch);
        exit(0);
```

```
}
```

4)

```
int i=4,        j=10;
int main ( )
{
   int p ;
   p = fork();
   if(p<0) exit(1) ;
   j += 2;
   if (p == 0)
   {        i *= 3;
            j *= 3;
 }
 else
 {
            i *= 2;
            j *= 2;
 }
printf("i=%d, j=%d", i,j) ;
return 0 ;
}
```

5)

```
int main ( )
{       int p=1 ;
        for(int i=0 ; i<=4 ; i++)
        if (p>0) p=fork( ) ;
        if(p !=-1) execlp("prog", "prog", NULL) ;
 else        exit(1) ;
        while( wait(NULL) !=-1) ;
 return          0 ;
}
```

Exercice 3 : ordonnancement des processus

Considérons n processus P1, P2, …, Pn, arrivés en même temps et insérés dans cette ordre dans la file des processus prêts. Ces processus ne font pas d'E/S et leurs temps d'exécution sont respectivement c1, … et cn. Le temps de commutation est supposé nul.

1) Quel est le temps d'attente moyen des n processus dans chacun des cas suivants :

- D'un ordonnanceur circulaire avec un quantum qt.
- D'un ordonnanceur sans préemption fonctionnant selon la discipline premier arrivé, premier servi.

Dans quel cas, obtient-on un meilleur temps d'attente moyen ?

2) Supposons que le nombre de processus est 5 et que leurs temps d'exécution sont égaux à :
2*qt + r avec r<qt.

- Montrez comment les processus vont utiliser le processeur dans le cas d'un ordonnanceur circulaire avec un quantum qt. Calculer le temps moyen de séjour des processus.
- Quel serait le temps moyen de séjour des 5 processus dans le cas d'un ordonnanceur sans préemption fonctionnant selon la discipline premier arrivé, premier servi.

Dans quel cas, obtient-on un meilleur temps de séjour moyen ?

Exercice 4 : Ordonnancement des processus

On considère 4 processus, A, B, C, D. On suppose que l'exécution des processus nécessite :

- Pour A : 7 unités de temps CPU, 3 unités de temps d'E/S et 5 unités de temps CPU.
- Pour B : 6 unités de temps CPU, 4 unités de temps d'E/S, 4 unités de temps CPU.
- Pour C : 5 unités de temps CPU.
- Pour D : 1 unité de temps CPU, 4 unités de temps d'E/S et 2 unités de temps CPU.

On suppose que

- A se présente en premier, à l’instant 0,
- B se présente à l’instant 1,
- C se présente à l’instant 9,
- D se présente à l’instant 12.

Montrez comment les 4 processus vont utiliser le processeur dans chacun des cas suivants :

1) Chaque processus a son propre périphérique d’E/S et l’ordonnanceur fonctionne selon Premier Arrivée Premier Servi PAPS (sans préemption).

2) Chaque processus a son propre périphérique d’E/S et l’ordonnanceur utilise l'algorithme du tourniquet, avec un quantum de 5. Le temps de commutation est égal à 0. Donnez, dans ce cas, les temps de séjour des processus A, B, C et D.

3) Les trois processus utilisent le même périphérique d'E/S dont la file d'attente est gérée premier arrivée premier servi. L’ordonnanceur du processeur utilise l'algorithme du tourniquet, avec un quantum de 5. Le temps de commutation est supposé égal à 0.

Exercice 5 :

1) Soient trois processus concurrents P1, P2 et P3 qui partagent les variables n et out. Pour contrôler les accès aux variables partagées, un programmeur propose les codes suivants :

```
Semaphore mutex1 = 1 ;
Semaphore mutex2 = 1 ;

Code du processus p1 :
        P(mutex1) ;
        P(mutex2) ;
        out=out+1 ;
        n=n-1 ;
        V(mutex2) ;
```

V(mutex1) ;

Code du processus p2 :

P(mutex2) ;
out=out-1 ;
V(mutex2) ;

Code du processus p3 :

P(mutex1) ;
n=n+1 ;
V(mutex1) ;

Cette proposition est-elle correcte ? Sinon, indiquer parmi les 4 conditions requises pour réaliser une exclusion mutuelle correcte, celles qui ne sont pas satisfaites ? Proposer une solution correcte.

1) On veut effectuer en parallèle le produit de deux matrices A et B d'ordre n (nxn). Pour se faire, on crée m (m<n) processus légers (threads). Chaque processus léger se charge de calculer quelques lignes de la matrice résultat R :

   Pour j = 0 à n-1 $\quad R[i,j] = \sum_{k=0,n-1} A[i,k]*B[k,j]$;

Donner sous forme de commentaires (en utilisant les sémaphores et les opérations P et V), le code des processus légers : CalculLignes ( ). Préciser les sémaphores utilisés et les variables partagées.

Exercice 6 : Synchronisation des processus

Deux villes A et B sont reliés par une seule voie de chemin de fer. Les trains peuvent circuler dans le même sens de A vers B ou de B vers A. Mais, ils ne peuvent pas circuler dans les sens opposés. On considère deux classes de processus : les trains allant de A vers B (Train AversB) et les trains allant de B vers A (Train BversA). Ces processus se décrivent comme suit :

Train AversB :

Demande d'accès à la voie par A ;
Circulation sur la voie de A vers B;

Sortie de la voie par B;

Train BversA :

Demande d'accès à la voie par B ;
Circulation sur la voie de B vers A;
Sortie de la voie par A;

1) Parmi les modèles étudiés en classe (producteur/consommateur, lecteur/rédacteur, les philosophes), ce problème correspond à quel modèle ?

2) Ecrire sous forme de commentaires en utilisant les sémaphores, les opérations P et V, les codes de demandes d'accès et de sorties, de façon à ce que les processus respectent les règles de circulation sur la voie unique.

Exercice 7 :

Considérons le problème producteur/consommateur, vu en classe. Adaptez la solution suivante :

1- Au cas de n producteurs, n consommateurs et un seul tampon de taille (Max, il peut contenir au plus Max messages). Les producteurs produisent des messages et les déposent dans le tampon. Chaque message déposé dans le tampon est récupéré (consommé) par un seul consommateur.

2- Au cas d'un seul producteur, n consommateurs et n tampons de même taille (Max). Chaque message produit par le producteur est déposé dans tous les tampons en commençant par le premier. Le consommateur i récupère (consomme) les messages déposés dans le tampon i.

Semaphore Mutex =1, Vide=Max, Plein=0 ;
Message tampon [Max] ;
Producteur ( )

```
{       int ip =0 ;
        Message m ;
 Repeter
        {       m = creermessage() ;
                P(Vide) ;
 P(Mutex)                ;
 Tampon[ip]=m;
 V(Mutex)                ;
 ip++                    ;
 V(Plein)                ;
        }tant que vrai ;
}


Consommateur( )
{       int ic =0 ;
        Message m ;
 Repeter
 {
                P(Plein) ;
 P(Mutex)                ;
 m               = Tampon[ic];
 V(Mutex)                ;
 ic++                    ;
 V(Vide)                 ;
        }tant que vrai ;
}
```

Solutions

Exercice 1 :

1) Il gère et contrôle le matériel et offre aux utilisateurs une machine virtuelle plus simple d'emploi que la machine réelle (appels systèmes). Non, les interpréteurs et les compilateurs ne font pas parties du système d'exploitation.

2) Un système multiprogrammé gère le partage des ressources (mémoire, processeur, périphériques…) de l'ordinateur entre plusieurs programmes chargés en mémoire. Dans un système de traitement par lots, les processus sont exécutés l'un à la suite de l'autre selon l'ordre d'arrivée. Dans un système en temps partagé, le processeur est alloué à chaque processus pendant au plus un quantum de temps. Au bout de ce quantum, le processeur est alloué à un autre processus.

3) A partir de la bibliothèque standard des appels système (instruction TRAP). Ils sont exécutés en mode superviseur (Leurs codes constituent le système d'exploitation).

4) Les fichiers sont organisés dans des répertoires. Chaque répertoire peut contenir des fichiers ou des répertoires (une structure arborescente). Pour contrôler les accès aux fichiers, chaque fichier a son propre code d'accès sur 9 bits. Un utilisateur peut accéder à un fichier d'un autre utilisateur si le code d'accès du fichier le permet. Le chemin d'accès est absolu.

5) Oui à l'exception du processus INIT. Le processus INIT devient son père. Un processus devient Zombie lorsqu'il effectue l'appel système exit et envoie donc un signal à son père puis se met en attente que le père ait reçu le signal.

6) tht_create ( ) car le fork( ) consomme beaucoup d'espace (duplication de processus). Mais il faut faire attention au conflit d'accès aux objets partagés.

7) 1) fin d'un quantum, 2) demande d'une E/S, 3) arrivée d'un signal, 4) mise en attente par l'opération sem_wait d'un sémaphore…

8) L'ordonnanceur gère l'allocation du processeur aux différents processus. L'ordonnanceur d'UNIX est un ordonnanceur à deux niveaux, à priorité qui ordonnance les processus de même priorité

selon l'algorithme du tourniquet. L'ordonnanceur de bas niveau se charge de sélectionner un processus parmi ceux qui sont prêts et résidents en mémoire. Cette restriction permet d'éviter lors de commutation de contextes qu'il y ait un chargement à partir du disque d'un processus en mémoire (réduction du temps de commutation). L'ordonnanceur de haut niveau se charge de ramener des processus prêts en mémoire en transférant éventuellement des processus sur disque (va-et-vient).
Oui, il favorise les processus interactifs car ces derniers font beaucoup d'E/S et à chaque fin d'E/S, ils se voient attribuer une priorité négative.

9) Un autre processus peut accéder aux données partagées avant qu'un processus n'est fini de les utiliser (modifier). Oui, par exemple les sémaphores. Une suite d'instructions qui accèdent à des objets partagés avec d'autres processus.

Exercice 2

1) Le père crée des processus fils tant qu'il n'y a pas d'échec. Le père et les processus créés se transforment en prog.

2) Le processus père tente de créer un fils et rentre dans la boucle. Si la création a échoue, le processus père se termine (p<0). Sinon il sort de la boucle car l'expression (i—) devient égale à 0. Il exécute ensuite j+=2 ; i*=2 ; j*=2 ; et enfin, il imprime les valeurs 0 et 24.
Le fils ne rentre dans la boucle car i=1 mais p=0. Il exécute ensuite j+=2 ; i*=3 ; j*=3 ; et enfin, il affiche les valeurs 3 et 36.

3) pour i=2, le père crée un pipe puis un fils. Il dirige sa sortie standard vers le pipe puis il se met en attente de données du clavier pour les déposer sur le pipe. Ensuite, il se termine.
Le fils dirige son entrée standard vers le pipe puis crée un autre pipe et son propre fils (i=1). Il dirige sa sortie standard vers le deuxième pipe créé puis se met en attente de lecture de données du

premier pipe. Les données lues sont déposées sur le deuxième pipe. Ensuite, il se termine.
Le petit fils dirige son entrée standard vers le pipe. Il se met en attente de lecture de données du second pipe. Il sort de la boucle car i devient nul. Les données lues sont affichées à l'écran. Enfin, il se termine.

4) Le père tente de créer un fils. S'il ne parvient pas il se termine. Sinon, il affiche i =8, j=24.
Le fils affiche i=12, j=36

5) Le père tente de créer 5 fils. S'il parvient, il se transforme en prog. Sinon il se termine. Les fils créés se transforment en prog.

Exercice 3 :

1) Posons $A_i = \min(qt, c_i)$ pour i=1,n

$TAM_1 = [0+ (n-1)*A_1 + (n-2)*A_2 +\ldots.+ A_{n-1}] / n$

$TAM_2 = [0+ (n-1)*c_1 + (n-2)*c_2 +\ldots.+ c_{n-1}] / n$
Comme $A_i \leq c_i$ pour i=1,n , $TAM_1 \leq TAM_2$.

2) (P1,qt) (P2,qt) (P3,qt) (P4,qt) (P5,qt) (P1,qt) (P2,qt) (P3,qt) (P4,qt) (P5,qt) (P1,r) (P2,r) (P3,r) (P4,r) (P5,r)

TSM1 = [(10 qt +r) + (10 qt +2r) + (10 qt +3r) + (10 qt + 4r) +(10qt +5r)] /5
= 10 qt + 3 r

3) (P1, 2qt+r)(P2, 2qt+r) (P3, 2qt+r) (P4, 2qt+r) (P5, 2qt+r)
TSM2 = [(2 qt +r) + 2(2qt+r) + 3(2 qt +r) + 4(2 qt +r) + 5(2 qt +r)] /5
= 6 qt + 2 r

TSM2 $\leq$ TSM1

Exercice 4 :

1) (A,7) (B,6) (C,5) (A,5)(D,1)(B,4)(D,2)
2) (A,5) (B,5) (A,2)(C,5)(B,1)(D,1)(A,5)(B,4)(D,2)
pour A : 24
pour B : 27
pour C : 8
pour D : 18
3) (A,5) (B,5) (A,2)(C,5)(B,1)(D,1)(A,5)(B,4)(D,2)

Exercice 5 : Synchronisation des processus

1) Non, car si P2 est en section critique et P1 a exécuté P(mutex1) alors P1 est bloqué et empêche P3 d'entrer en section critique.
Conditions non vérifiées :
Un processus en dehors de sa section critique bloque un autre processus.

Processus P1
P(mutex1) ;
n=n-1 ;
V(mutex1) ;
P(mutex2) ;
Out = out +1 ;
V(mutex2) ;

2) On va utiliser un vecteur T de n booléens. T[i] est 0 si le calcul de la ligne i n'est pas encore entamée. T[i] est égal à 1 sinon. Ici, on n'a pas besoin de sémaphores pour les accès aux matrices (en lecture uniquement). Par contre, on a besoin d'un sémaphore binaire mutex pour contrôler les accès au vecteur T. Initialement, tous les éléments de T sont nuls.

```
fonction CalculLignes ( )
{   pour i = 0 à n-1 pas 1
    faire    P(mutex)
```

```
                si ( T[i]==0)
                {       T[i] = 1 ;
                        V(mutex) ;
                        Pour j = 1 à n pas 1
                                faire   Pour k=1 à n pas 1
                                        faire   R[i,j] += A[i,k] * B[k,j]
                                fait
                        fait
                } else V(mutex) ;
fait
   }
```

Exercice 6 :

1) Modèle des lecteurs et des rédacteurs (la voie joue le rôle de la base de données).

2)
Sémaphore mutex =1, autorisation =1 ;

Demande d'accès par un train AversB
P(mutex)
Si NbAB =0 alors P(autorisation)
NbAB=NbAB+1 ;
V(mutex) ;

Sortie de la voie par B
P(mutex)
Si NbAB =1 alors V(autorisation)
NbAB=NbAB-1 ;
V(mutex) ;

Demande d'accès par un train BversA
P(mutex)
Si NbBA =0 alors P(autorisation)
NbBA=NbBA+1 ;
V(mutex) ;

Sortie de la voie par A
P(mutex)
Si NbBA =1 alors V(autorisation)
NbBA=NbBA-1 ;
V(mutex) ;

Exercice 7 :
a)

Semaphore Mutex =1, Vide=Max, Plein=0 ;
Message tampon [Max] ;
int ip =0 ; // ip devient global
int ic =0 ; // ic devient global

```
Producteur ( int i)
{
        Message m ;
 Repeter
        {       m = creermessage( ) ;
                P(Vide) ;
  P(Mutex)              ;
  Tampon[ip]=m;
  ip++ ;                        // ip++ est dans la section critique
                V(Mutex) ;
  V(Plein)              ;
        }tant que vrai ;
}


Consommateur( int i )
{
        Message m ;
 Repeter
 {
                P(Plein) ;
  P(Mutex)              ;
  m             = Tampon[ic];
```

```
                ic++ ;  // ic est partagé entre tous les consommateurs
 V(Mutex)                ;

 V(Vide)                 ;
        }tant que vrai ;
}
```

b)

```
Semaphore
Mutex [n]  = {1, 1, ….,1},
Vide [n] = {Max, Max, …, Max},
Plein [n] = {0, 0, ., 0} ;
Message tampon [n][Max] ;
Producteur ( )
{       int ip =0 ; // un seul producteur
        Message m ;
 Repeter
        {       m = creermessage() ;
                pour i=0, n- 1, pas 1
                faire
                        P(Vide[i]) ;
  P(Mutex[i])                   ;
  Tampon[i][ip]=m;
  V(Mutex[i])                   ;
  ip++                  ;
  V(Plein[i])                   ;
        }tant que vrai ;
}


Consommateur( int i)
{       int ic =0 ;
        Message m ;
 Repeter
 {
                P(Plein[i]) ;
  P(Mutex[i])           ;
```

```
  m                    = Tampon[i][ic];
  V(Mutex[i])                    ;
  ic++                   ;
  V(Vide[i])                   ;
          }tant que vrai ;
}
```