I42: partie « Système d'exploitation » — Exercices

1 Les Sémaphores

Exercice 1:

Soient trois processus concurrents P1, P2 et P3 qui exécutent chacun le programme suivant :

```
P_i() // i = 1,2,3 {
   int n=0;
   while(true)
      printf("cycle %d de %d", n++, i);
}
```

Synchronisez les cycles des processus à l'aide de sémaphores de manière à ce que :
- Chaque cycle de P1 s'exécute en concurrence avec un cycle de P2
- Le processus P3 exécute un cycle après que P1 et P2 terminent tous deux l'exécution de leur cycle
- Lorsque P3 termine un cycle, les processus P1 et P2 entament chacun un nouveau cycle et ainsi de suite ...

Exercice 2:

Plaçons nous dans une usine automatisée d'assemblage de stylo à bille. Chaque stylo est formé d'un corps, d'une cartouche, d'un bouchon arrière et d'un capuchon. Les opérations à effectuer sont les suivantes :
- Remplissage de la cartouche avec l'encre (opération RC),
- Assemblage du bouchon arrière et du corps (opération BO),
- Assemblage de la cartouche avec le corps et le capuchon (opération AS),
- Emballage (opération EM).

Chaque opération est effectuée par une machine spécialisée (mRC, mBO, mAS, mEM). Les stocks de pièces détachées et d'encre sont supposés disponibles quand la machine est disponible. Les opérations RC et BO se font en parallèle. L'opération AS doit être effectuée après ces deux opérations, en prélevant directement les éléments sur les machines mRC et mBO. Le produit assemblé est déposé dans un stock en attente de l'opération EM. L'opération EM se fait donc après AS, à partir du stock. Le stock est supposé de taille N et de discipline FIFO.

Synchronisez au moyen de sémaphores l'enchaînement des opérations de fabrication de stylos à bille et donner le pseudo-code de chaque machine.

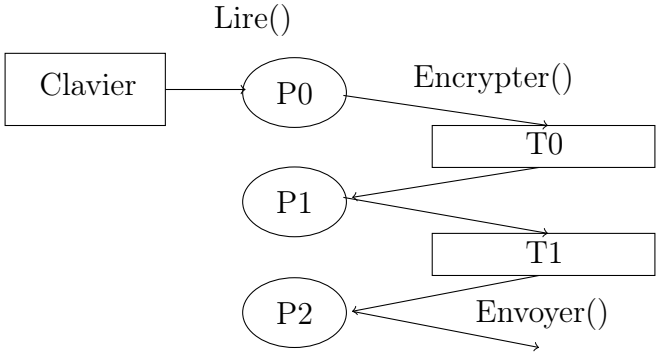
Exercice 3:

Considérez trois processus concurrents P0, P1 et P2 appartenant chacun à une couche logique différente et qui communiquent au moyen de deux tampons T0 et T1 de même taille N :
- P0 et P1 partagent le tampon T0
- P1 et P2 partagent le tampon T1

Chaque couche se charge d'un traitement particulier :
- Le processus P0 se charge de lire du clavier des messages qu'il traite avant de les déposer dans le tampon T0. Ce traitement consiste au cryptage du message avant de la mettre dans le tampon T0.
- Le processus P1 se charge de transférer directement les messages du tampon T0 vers le tampon T1

— Le processus P2 récupère les messages du tampon T1 pour les envoyer à un destinataire.

Trois fonctions sont prédéfinies :
— `Message Encrypter (Message);` qui permet d'encrypter le message
— `Message Lire ();` qui permet de lire un message du clavier.
— `Envoyer (Message);` qui envoie un message à un destinataire.

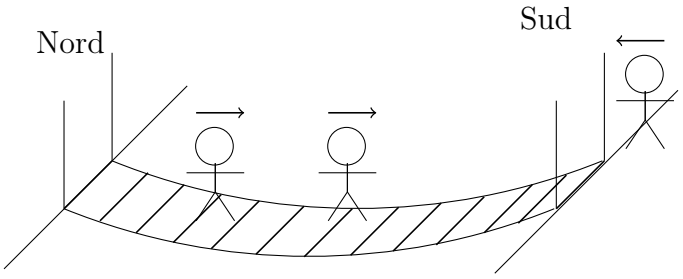


1. Donnez les pseudo-codes des trois processus dans le cas où T0 et T1 sont des tampons de N messages.
2. Donnez les pseudo-codes des trois processus dans le cas où T0 et T1 sont des tampons de N caractères.

2 Moniteur

Exercice 4:

Compétition d'accès pour deux classes de processus



Soit un pont d'une largeur ne permettant pas la traversée dans les deux sens en même temps sur celui-ci.

Soient deux classes de processus *NS* et *SN* représentant respectivement les gens qui veulent emprunter le pont dans le sens *Nord-Sud* et les gens qui veulent l'emprunter dans le sens *Sud-Nord*.

La circulation sur le pont doit respecter les deux contraintes suivantes :
— le pont ne doit jamais être emprunté simultanément par des personnes allant dans des sens différents
— le pont peut être emprunté par une ou plusieurs personnes allant dans le même sens

Dans ce problème on ne tiendra pas compte des problèmes de famine.

1. Proposer le pseudo-code de ces deux classes de processus tout en respectant les contraintes précédentes en utilisant un moniteur (4 fonctions à écrire).
2. Ecrire le pseudo-code mais cette fois à l'aide de sémaphores.

3 Interblocage

Exercice 5:

Soit un système composé de 4 processus P_0, P_1, P_2 et P_3, et de 3 types de ressources R_0, R_1 et R_2. Le vecteur des ressources existentes est $E = [3, 2, 2]$. Les conditions d'utilisation de ressources sont les suivantes :

— P_0 possède 1 unité de R_0 et demande 1 unité de R_1,
— P_1 possède 2 unités de R_1 et demande 1 unité de R_0 et de R_2,
— P_2 possède 1 unité de R_0 et demande 1 unité de R_1,
— P_3 possède 2 unités de R_2 et demande 1 unité de R_0.

Donnez le graphe d'allocation des ressources pour le système décrit ci-avant. Réduisez le graphe et indiquez s'il y a interblocage.

Exercice 6:

Soient trois processus (*P1, P2, P3*) concurrents qui utilisent en exclusion mutuelle chacun trois ressources parmi six (*A, B, C, D, E, F*). Ces trois processus exécutent en boucle respectivement les pseudo-codes suivants :

```
P1                  P2                  P3
Prendre(D)          Prendre(C)          Prendre(A)
Prendre(E)          Prendre(B)          Prendre(B)
Prendre(C)          Prendre(F)          Prendre(E)
// Utilisation      // Utilisation      // Utilisation
// des              // des              // des
// ressources       // ressources       // ressources
Libérer(D)          Libérer(F)          Libérer(E)
Libérer(E)          Libérer(B)          Libérer(B)
Libérer(C)          Libérer(C)          Libérer(A)
```

Ces processus concurrents peuvent-ils entrer en interblocage ? Si oui décrivez le scénario menant à cet interblocage et est-il possible de l'éviter (sans modifier les codes des processus), sinon justifiez pourquoi.

Exercice 7:

Soit un ensemble de processus composé de plusieurs producteurs et d'un seul consommateur, avec un tampon de taille N, dont voici les pseudo-codes :

```
Sémaphore plein=0, vide=N, mutex=1;
char T[N];
int ip=0;
```

```
Producteur()
char ch[N];
int i,M;
répéter {
   M=Lire(ch)
   M = min(M,N);
   pour i de 1 à M faire
      P(vide);
   P(mutex);
   Déposer(ch, M, ip);
   ip = (ip+M)%N;
   V(mutex);
   pour i de 1 à M faire
      V(plein);
} tant que vrai
```

```
Consommateur()
int ic = 0;
char c;
répéter {
   P(plein);
   P(mutex);
   c = T[ic];
   V(mutex);
   V(vide);
   Traiter(c);
   ic = (ic+1)%N;
} tant que vrai
```

1. Peut-on avoir interblocage
   — Dans le cas où il n'y qu'un seul producteur ?
   — Dans le cas où il y a plusieurs producteurs ?
   Expliquez pourquoi ou donner un exemple d'interblocage.
2. Compléter le code précédent afin d'éviter les situations d'interblocage.

4 Les Signaux

Exercice 8:

Le signal `SIGCHLD` est envoyé automatiquement par le processus fils à son processus père lorsque le processus fils se termine (par exemple par l'instruction `exit()`).

Compléter le code suivant afin que le processus père n'attende pas le processus fils de façon bloquante et que le processus fils ne devienne pas zombie.

```
int main(int argc, char *argv[]) {
   if (fork() == 0)  {
      for (int i = 0; i < 10; i++);  /* une phase de calcul */
      exit(1);
   }
   while(1);  /* une phase de calcul */
}
```

Exercice 9:

Lorsqu'un processus tente d'écrire dans un tube rompu, le signal `SIGPIPE` est envoyé au processus. Le traitement par défaut de ce signal est la terminaison du processus.

Donnez les étapes ou les opérations à réaliser afin que le processus affiche un message d'erreur spécifiant le descripteur pour lequel le tube est rompu puis se termine.

Exercice 10:

Soit le programme suivant :

```
#define N 5
int main(int argc, char *argv[]) {
   pid_t pid[N];
   int i;
   for (i = 0; i < N; i++) {
      if ((pid[i] = fork()) == 0) {
         while(1)
            printf(''fils numero %d\n'', i);
      }
   }
}
```

On veut que le processus père utilise les signaux `SIGSTOP` et `SIGCONT` pour bloquer et débloquer l'exécution de ses processus fils. Au départ tous les processus fils créés doivent se mettre en pause.

Le processus père répète indéfiniment les étapes suivantes en commençant par le premier processus fils :

— Envoyer `SIGCONT` au fils puis s'endormir pendant une seconde
— Au réveil envoyer `SIGSTOP` au même fils
— Passage au fils suivant (le suivant du dernier est le premier)

De plus, lorsqu'un processus fils reçoit le signal `SIGCONT`, il affiche un message indiquant qu'il a reçu le signal `SIGCONT` avant de poursuivre son exécution. Complétez le code précédent afin de répondre aux contraintes de l'énoncé.

Exercice 11:

Soit le programme suivant :

```
#include <unistd.h>
#include <signal.h>
#include <stdio.h>

void sigintP() { }
void sigintF() { }
void sigalrm() { }
void sigchld() {
   int status;
   wait(&status);
   exit();
}

int main(void) {
   signal(SIGCHLD, sigchld);
   if (fork() == 0) {
      signal(SIGINT, singintF);
      while(1) {
         printf("ici le fils\n");
         sleep(1);
      }
   }
   while(1) {
      signal(SIGINT, sigintP);
      printf("ici le père\n");
      sleep(1);
  }
  return 0
}
```

Complétez le code précédent de manière à réaliser les traitements suivants :

— A chaque fois que l'utilisateur presse les touches Ctrl-C, le processus père affiche son `pid` sans se terminer ;

— A la première combinaison des touches Crtl-C lorsque le programme s'exécute, les processus père et fils ne se terminent pas immédiatement mais après un délai de 5 secondes.

Indication : faire que Ctrl-C doive déclencher un appel système `alarm(5)`, qui envoie automatiquement le signal `SIGALRM` après 5 secondes.

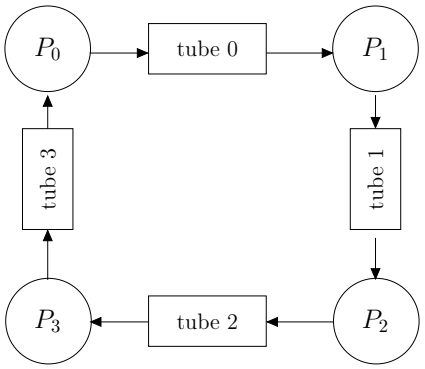
5 Les Tubes

Exercice 12:

Soient N processus qui communiquent entre eux au moyen de tubes de communication non nommés (unnamed pipe). Chaque processus partage deux tubes : un avec le processus le précédent et un avec le processus lui succédant. Dans le cas où $N = 4$, les processus communiquent selon le schéma suivant :

Compléter le code suivant de manière à implémenter cette architecture de communication des N processus créés : l'entrée standard et la sortie standard de chaque processus P_i sont redirigées vers les tubes appropriés. Par exemple, pour le processus P0, l'entrée et la sortie standards deviennent respectivement les tubes tube3 et tube0.

```
#define N 4
void proc(int param) ;
```




```
int main() {
   int i ;
   for( i = 0 ; i < N; i++)
      if ( fork() == 0) {
         proc(i) ;
         exit(0) ;
      }
   exit(0) ;
}
```

Remarque : On ne demande pas d'écrire le code de la fonction `proc()`.

Exercice 13:

Ecrivez le code principal (le *main*) d'un processus qui permet de simuler un ('|') utilisé habituellement en commande shell. Ce code doit avoir le même comportement que cette commande.

1. Proposez une version utilisant des tubes non nommés.
2. Proposez une version avec des tubes nommés.

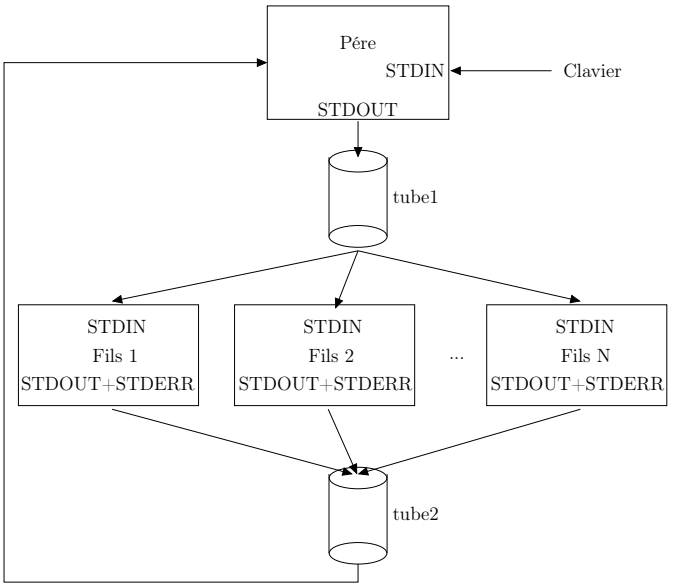
Exercice 14:

Soit le programme suivant :

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#define N 5

void gestion() { /* traitement */};

int main(void) {
   pid_t pid[N];
   int i;
   for (i = 0; i < N; i++) {
      if ((pid[i] = fork()) == 0) {
         execlp("traitement", "traitement", NULL);
         exit(1);
      }
   }
   gestion();
   return 0;
}
```



On veut faire communiquer le processus père avec ses processus fils au moyen de deux tubes anonymes selon le schéma suivant :

La sortie standard du processus père est redirigée vers le tube `tube1` qui devient l'entrée standard des fils. Les sorties standards et d'erreurs standards des fils sont redirigées vers le tube `tube2`.

Attention, le processus père lira les données directement dans le tube `tube2` et n'aura donc pas de redirection de la sortie du tube vers son entrée standard.

Complétez le code de manière à établir ces canaux de communication.

Exercice 15:

Soit une fonction $F()$ d'une librairie externe qui écrit une certaine quantité de données sur la sortie standard (descripteur 1). On aimerait récupérer en utilisant un tube anonyme ces données afin de les traiter.

```
#define Taille 1024
int main(void) {
   char data[Taille];
   F();
   utiliser_resultat(data);
}
```

1. Insérez du code avant et après l'appel à $F()$ afin que tous les caractères émis par $F()$ sur la sortie standard soient récupérés dans `data`. Vous pouvez utiliser des variables et des appels supplémentaires mais vous ne pouvez pas utiliser de fichiers ni de processus supplémentaires.
2. Que se passe-t-il si la taille des données émises par $F()$ dépasse celle de `data` ?
3. Que se passe-t-il si la taille des données émises par $F()$ dépasse celle du tube ?
4. Proposez une deuxième solution corrigeant ces problèmes. Pour cela, vous pouvez utiliser un processus supplémentaire.

6 Les objets IPC

Exercice 16:

Soient deux processus indépendants p1 et p2. Ecrire les primitives P() et V() afin d'effectuer un rendez-vous entre les deux processus.

Exercice 17:

Soient deux processus indépendants p1 et p2. Ces deux processus partagent une zone de mémoire partagée pour s'échanger des données. Ecrire le code permettant aux deux processus d'échanger ainsi une chaîne de caractères. Attention, seul le premier processus connaît la clé de la zone mémoire. Il passera l'identificateur de cet objet à l'autre processus au moyen d'un tube nommé.

Exercice 18:

Soient deux processus indépendants p1 et p2.

Ces deux processus vont s'échanger un message à l'aide d'une file de messages. Le premier processus va envoyer un message constitué d'une chaîne de caractères, de longueur maximale de 256 caractères, et d'un entier dans la file de message. Ce message sera appelé de type 1. Ensuite, ce processus p1 attend le message de retour contenant la chaîne de caractères transformée par le processus p2. Le type du message retourné sera dit de type 2.

Le processus p2 quant à lui attend des messages de type 1 dans la file, les traite et renvoie dans la file des messages de type 2 contenant la chaîne de caractères transformée. La transformation d'un message de type 1 contenant un entier et une chaîne de caractères consistera à additionner à chaque caractère de la chaîne la valeur de l'entier (attention au marqueur de fin de chaîne de caractères).