UNIVERSITÉ TOULOUSE III PAUL SABATIER

2A – SRI

Université Paul Sabatier
118 route de Narbonne
31062 Toulouse Cedex 09

Méthodes et langages pour le parallélisme
Travaux pratiques
Synchronisation par sémaphores

Remarque générale :

N'oubliez pas de créer des IPC sur lesquels vous avez les **droits d'accès**.

À la fin de l'exécution d'une application, les **IPC** utilisés doivent avoir été **détruits (par l'application ou à la main – voir fin du sujet)**. Ce sont des **ressources persistantes limitées** et vous pouvez donc **bloquer** les **autres** utilisateurs en ne les libérant pas.

TP1-Exercice 1 – Affichage alterné à 2 processus (parents)

Écrire une application dans laquelle le processus principal crée 2 processus fils et attend leur terminaison pour se terminer lui-même. Avant de se terminer, les deux fils affichent NB fois un message de NBL lignes à tour de rôle à l'écran (NB et NBL sont des paramètres de l'application).

Lors des exécutions, faire varier NB et NBL pour s'assurer que la synchronisation est valide.

Exemple d'exécution :

%./tp1_exo1 2 2 [Les 2 processus affichent 2 fois un message de 2 lignes]

Processus 0 (65) : Ligne 1/2 du message 1/2
Processus 0 (65) : Ligne 2/2 du message 1/2
Processus 1 (66) : Ligne 1/2 du message 1/2
Processus 1 (66) : Ligne 2/2 du message 1/2
Processus 0 (65) : Ligne 1/2 du message 2/2
Processus 0 (65) : Ligne 2/2 du message 2/2
Processus 0 (65) : Terminé
Processus 1 (66) : Ligne 1/2 du message 2/2
Processus 1 (66) : Ligne 2/2 du message 2/2
Processus 1 (66) : Terminé
Processus père (64) : Terminé

TP1-Exercice 2 – Affichage alterné à N processus sans lien de parenté

Généraliser la solution de l'exercice précédent à N processus s'exécutant en parallèle (N est aussi un paramètre de l'application).

Faire en sorte que les processus puissent être lancés dans des fenêtres différentes. Un paramètre supplémentaire (le rang de création dans « l'anneau » d'affichage ainsi créé) pourra être utilisé pour différencier les processus lancés à partir du même programme.

Exemple d'exécution :

Terminal 1	*Terminal 2*	*Terminal 3*	*Terminal 4*
%./tp1_exo2 0 4 2 2 [1er processus lancé sur 4, 2 messages de 2 lignes] . . . (Affichera en 1er) . . .	%./tp1_exo2 1 4 2 2 [2e processus lancé sur 4, 2 messages de 2 lignes] . . . (Affichera en 2e) . . .	%./tp1_exo2 2 4 2 2 [3e processus lancé sur 4, 2 messages de 2 lignes] . . . (Affichera en 3e) . . .	%./tp1_exo2 3 4 2 2 [4e processus lancé sur 4, 2 messages de 2 lignes] . . . (Affichera en 4e) . . .

TP2-Exercice 1 – Partage de données entre processus

Écrire une application dans laquelle N processus Unix modifient le contenu d'un tableau partagé contenant NBE entiers (N et NBE sont des paramètres de l'application).

Au cours de son exécution, un processus ajoute 1 à chacune des cases du tableau puis affiche le contenu du tableau partagé.

Remarque : Dans cet exercice, la synchronisation d'accès au tableau ne sera pas gérée (ceci est au programme des exercices ultérieurs), il se peut donc que les affichages ne reflètent pas la réalité.

On écrira deux versions de cette application :

- **Version 1** : Les N processus sont créés par le processus principal.
- **Version 2** : L'application est constituée de N processus sans lien de parenté. On n'écrira qu'un seul programme, paramétré par le « rang » du processus qu'on lance lors de l'appel à l'exécutable (cf. exécution ci-dessous).

Exemple d'exécution pour la version 1 :

%./tp2_exo1_v1 4 10 [On crée 4 processus qui partagent un tableau de 10 entiers]

Processus 0 (414) : 1 1 1 1 1 1 1 1 1 1

Processus 1 (415) : 2 2 2 2 2 2 2 2 2 2

Processus 2 (416) : 3 3 3 3 3 3 3 3 3 3

Processus 3 (417) : 4 4 4 4 4 4 4 4 4 4

Processus pere (413) : 4 4 4 4 4 4 4 4 4 4

[Remarque : Ici, les différents processus terminent leur modification du tableau avant d'afficher, mais il se peut – selon l'entrelacement de leurs exécutions – que ce ne soit pas toujours le cas, voir exemple dans l'exercice suivant]

Exemple d'exécution pour la version 2 :

On considère 4 processus indépendants, lancés chacun – à partir du même programme – dans une fenêtre (terminal) différente.

Terminal 1	***Terminal 2***	***Terminal 3***	***Terminal 4***
%./tp2_exo1_v2 0 4 5 [1er processus lancé sur 4, tableau de 5 cases] Tableau : 1 1 1 1 1	%./tp2_exo1_v2 1 5 [2e processus lancé sur 4, tableau de 5 cases] Tableau : 2 2 2 2 2	%./tp2_exo1_v2 2 5 [3e processus lancé sur 4, tableau de 5 cases] Tableau : 3 3 3 3 3	%./tp2_exo1_v2 3 5 [4e processus lancé sur 4, tableau de 5 cases] Tableau : 4 4 4 4 4

TP2-Exercice 2 – Synchronisation des accès au tableau

Si vous n'avez pas constaté d'affichages « incohérents » lors de l'exécution de l'application précédente, temporisez légèrement vos processus lors de leur boucle de modification du tableau (grâce à la primitive *usleep()* – Voir son man) afin qu'ils relâchent l'UC au cours de leur exécution.

Exemple d'affichage « incohérent » :

Processus 0 (609) : tableau : 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 1

Processus 1 (610) : tableau : 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2

Processus 2 (611) : tableau : 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3

Processus pere (608) : tableau : 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3

En repartant de la version 1, utilisez des sémaphores pour que l'affichage reste cohérent quelle que soit la façon dont les processus obtiennent l'UC.

TP3-Exercice 1 – Producteurs-Consommateurs – Version de base

Écrire une application dans laquelle NBP processus « Producteurs » et NBC processus « Consommateurs » synchronisent leurs accès à un buffer (tableau) partagé géré de manière circulaire (voir C-TD). Les dépôts se feront dans l'ordre croissant des indices du tableau et les retraits se feront dans l'ordre des dépôts.

Pour simplifier, on se limitera à des processus ayant un lien de parenté, donc descendant du processus principal de l'application.

Les messages seront du type *Message* défini ainsi *typedef struct { int info ; int type ; } Message ;* (dans cet exercice, le champ type ne sera pas utilisé).

TP3-Exercice 2 – Producteurs-Consommateurs – Dépôts alternés

Modifier la solution de l'exercice 2 du TP précédent pour que les producteurs soient de deux types et déposent leurs messages dans le buffer de manière alternée (voir C-TD).

Pour continuer…

Pour aller plus loin, améliorer l'application pour que les producteurs et les consommateurs puissent s'exécuter dans des fenêtres séparées.

Commandes UNIX de gestion des IPC

* ***Fournir des informations sur l'usage des ressources IPC***

ipcs [-smq]

-s pour obtenir les informations sur les ensembles de sémaphores

-m pour obtenir les informations sur les segments de mémoire partagée

-q pour obtenir les informations sur les files de messages

* ***Supprimer une file de messages, un ensemble de sémaphores ou un segment de mémoire partagée***

ipcrm [-M key | -m id | -Q key | -q id | -S key | -s id] ...

Option en majuscule: Suppression grâce à la clé obtenue par ipcs

Option en minuscule : Suppression grâce à l'identificateur obtenu par ipcs

Il est aussi plus simple d'utiliser un script pour nettoyer régulièrement les IPC.

Exemple de script (bash) :

```
#! /bin/sh
if [ $# -ne 1 ]
then
  echo "Usage : $0 [m | s | q]" >&2
  exit 1
fi
clesIpc=`ipcs -"$1" | grep 0x | tr -s ' ' | cut -f2 -d' '`
for cle in $clesIpc
do
  ipcrm -"$1" $cle
done
exit 0
```