Licence 3 Informatique

Université Paul Sabatier
118 route de Narbonne
31062 Toulouse Cedex 09

Programmation Système
Travaux pratiques n°4 & 5

Exercice 1 – Tubes : anneau de communication

Préparation [TP4] : Schéma de communication entre les processus.
Rendu Moodle : Code associé au programme demandé.
Date limite : Fin de la 4e séance de TP.

On veut communiquer une information de manière **circulaire** entre N processus : le « premier » processus envoie l'information au « second » qui la transmet au « troisième » qui la transmet au « quatrième » et ainsi de suite jusqu'à ce que l'information revienne au premier émetteur. Tout processus qui reçoit une information affiche son contenu, incrémente sa partie variable avant de la réémettre vers le « suivant ».

L'information transmise entre les processus sera composée de trois parties : l'identité du processus qui en est émetteur, une valeur entière immuable et une valeur entière qui sera incrémentée à chaque émission, ceci permettra de vérifier que l'information circule dans le bon sens et de la bonne manière.

Le nombre N de processus pourra varier d'une exécution à l'autre et constituera le **paramètre** de l'application.

- ❖ Écrire un programme dans lequel le processus principal crée N processus fils qui communiquent selon le protocole décrit au-dessus. Lorsque l'information sera revenue au premier processus, l'application devra se terminer (le père attendra la fin de ses fils).
- Avant de coder, établir un **schéma** de communication entre les différents processus (fils) en précisant les structures de données utiles et le nombre de tubes employés.
- Le fichier rendu aura pour nom : VotreNomPrenom_tp45_exo1.c

Exemples d'exécution :

%./tp45_exo1 2 *[Cas où 2 processus communiquent]*

Processus de pid 4325, n°1 dans l'anneau : j'envoie au n°2 l'info [4325 – 234 – 1]
Processus de pid 4326, n°2 dans l'anneau : j'ai reçu de 4325, j'envoie au n°1 l'info [4326 – 234 – 2]
Processus de pid 4325 : l'information m'est revenue de 4326, je peux me terminer
Processus de pid 4326 : je me termine (aussi)
%

%./tp45_exo1 5 *[Cas où 5 processus communiquent]*

Processus de pid 5433, n°1 dans l'anneau : j'envoie au n°2 l'info [5433 – 765 – 1]
Processus de pid 5434, n°2 dans l'anneau : j'ai reçu de 5433, j'envoie au n°3 l'info [5434 – 765 – 2]
Processus de pid 5435, n°3 dans l'anneau : j'ai reçu de 5434, j'envoie au n°4 l'info [5435 – 765 – 3]
Processus de pid 5434 : je me termine (aussi)
Processus de pid 5436, n°4 dans l'anneau : j'ai reçu de 5435, j'envoie au n°5 l'info [5436 – 765 – 4]
Processus de pid 5437, n°5 dans l'anneau : j'ai reçu de 5436, j'envoie au n°1 l'info [5437 – 765 – 5]
Processus de pid 5436 : je me termine (aussi)
Processus de pid 5433 : l'information m'est revenue de 5437, je peux me terminer
Processus de pid 5435 : je me termine (aussi)
Processus de pid 5437 : je me termine (aussi)

%

Exercice 2 – Signaux : conservation ou non du traitement associé

Préparation [TP4] : Structure (ou algorithme) de chaque version du programme demandé.
Rendu Moodle : Code associé à chaque version du programme demandé.
Date limite : 01/12/2019.
Code disponible sous Moodle : afficher.h, afficher.c, tp45_exo2-v2_base.c et tp45_exo2-v2_base.c

On veut mettre en évidence la manière dont les traitements associés à un signal par un processus sont hérités par ses processus fils ou oubliés lors d'un recouvrement (ou commutation d'image).

Code disponible sous Moodle :

- afficher.h et afficher.c : qui fournissent la fonction :
  void afficher(void)
  qu'utilisera un processus pour afficher indéfiniment un message l'identifiant.
- tp45_exo2-v2_base.c et tp45_exo2-v2_base.c qui fournissent un squelette de code pour chaque version.

❖ **Version 1** : Mise en évidence de l'héritage du traitement par un processus fils.

- Écrire un programme dans lequel un processus (père) associe le traitement consistant à afficher le message « >> Ctrl-C/SIGINT reçu par le processus de n° <pid> » lorsque le signal SIGINT lui parvient, puis crée un processus fils qui affiche indéfiniment un message l'identifiant (utiliser la fonction *afficher*() fournie).
  Exécuter cette application pour voir ce qu'il se passe en envoyant de temps en temps le signal SIGINT par appui des touches Ctrl & C du clavier.
- Le fichier rendu aura pour nom : VotreNomPrenom_tp45_exo2-v2.c

- **Exemple d'exécution :**

%./tp45_exo2-v1
Processus (père) de pid 5123 : protégé contre SIGINT [Ctrl-C au clavier]
>> Ctrl-C/SIGINT reçu par 5123
Processus (père) de pid 5123 : j'ai créé un fils de pid 5124
Processus (fils) de pid 5124 : je boucle
Processus (fils) de pid 5124 : je boucle [Ctrl-C au clavier]
. . . . La suite à tester . . . .

❖ **Version 2** : Mise en évidence de l'oubli du traitement associé à un signal par un processus appelant une primitive de recouvrement.

- Écrire un programme qui affiche indéfiniment un message l'identifiant (utiliser la fonction *afficher*() fournie). À partir de ce programme, créer un exécutable nommé *boucle*.
- Écrire un programme dans lequel le processus principal associe le traitement consistant à afficher le message « >> Ctrl-C/SIGINT reçu par <pid> » lorsque le signal SIGINT lui parvient ; puis remplace son code (recouvrement ou commutation d'image) par celui de l'exécutable *boucle*. Exécuter cette application pour voir ce qu'il se passe.
- Le fichier rendu aura pour nom : VotreNomPrenom_tp45_exo2-v2.c

Exemple d'exécution :

%./tp45_exo2-v2

Processus de pid 5126 : protégé contre SIGINT [Ctrl-C au clavier]
>> Ctrl-C/SIGINT reçu par 5126
Processus de pid 5126 : je vais exécuter « boucle »
Processus de pid 5126 : je boucle
Processus de pid 5126 : je boucle
Processus de pid 5126 : je boucle [Ctrl-C au clavier]
. . . . La suite à tester . . . .

Exercice 3 – Signaux et tubes : envoi périodique

Préparation [TP5] : Schéma de communication entre les processus.
Rendu Moodle : Code associé au programme demandé.
Date limite : Fin de la 5e séance de TP.
Code disponible sous Moodle : tp45_exo3_base.c

Code disponible sous Moodle :

- tp45_exo3_base.c : Squelette de programme.

❖ Écrire un programme dans lequel un processus père envoie à son fils un message à intervalles de temps réguliers un certain nombre de fois (**Attention** : ne pas utiliser la primitive *sleep()*). Le délai (en secondes) entre deux envois, et le nombre d'envois à réaliser constituent les paramètres de cette application. Le père peut continuer son traitement de fond avant d'envoyer son message (gestion asynchrone) et doit se terminer en dernier.

- Avant de coder, établir un **schéma** de communication entre les processus en précisant les tubes et signaux utilisés.
- Le fichier rendu aura pour nom : VotreNomPrenom_tp45_exo3.c

Exemple d'exécution :

%./tp45_exo3 2 7 [Envoi toutes les 2 secondes à concurrence de 7 messages]

Fils - Message de mon pere : Message numero 0 envoye a Thu Nov 22 14:45:14 2018
Fils - Message de mon pere : Message numero 1 envoye a Thu Nov 22 14:45:16 2018
Fils - Message de mon pere : Message numero 2 envoye a Thu Nov 22 14:45:18 2018
Fils - Message de mon pere : Message numero 3 envoye a Thu Nov 22 14:45:20 2018
Fils - Message de mon pere : Message numero 4 envoye a Thu Nov 22 14:45:22 2018
Fils - Message de mon pere : Message numero 5 envoye a Thu Nov 22 14:45:24 2018
Fils - Message de mon pere : Message numero 6 envoye a Thu Nov 22 14:45:26 2018
Fils : Je me termine a Thu Nov 22 14:45:26 2018
Pere : Je me termine en dernier a Thu Nov 22 14:45:26 2018

Exercice 4 – Signaux et tubes : ouvriers

Préparation [TP5] : Schéma de communication entre les processus.
Rendu Moodle : Code associé au programme demandé.
Date limite : 11/12/2019 – Minuit.

On se propose de simuler une chaîne de fabrication d'un certain type de produit. Cette chaîne est composée de NBO processus ouvrier qui travaillent en parallèle à traiter les requêtes d'un certain nombre NBC de processus clients.

Le premier ouvrier de la chaîne reçoit une requête de la part d'un client. Il prépare le produit demandé et le transmet au deuxième ouvrier. Chaque ouvrier de la chaîne réalise une tâche (par exemple, en modifier la valeur) sur le produit reçu avant de le transmettre à l'ouvrier suivant.

Quand le produit arrive au dernier ouvrier, celui-ci prévient le client ayant passé commande en lui envoyant le signal SIGUSR1.

Quand il n'y a plus de clients susceptibles de faire des commandes, les ouvriers se terminent et le programme principal rend la main au processus shell.

Après avoir passé sa commande, un client attend jusqu'à ce qu'on l'avertisse de la disponibilité de son produit.

- Avant de coder, commencer par bien réfléchir au **schéma de communication** entre les différents processus en précisant les structures de données utiles et en optimisant le nombre de tubes utiles. Vous pouvez largement vous inspirer des exercices précédents.
- Écrire un programme simulant le comportement décrit au-dessus. Le faire de manière incrémentale :
  - Écrire la fonction *void ouvrier (int num) ;* qui définit le **comportement** d'un processus ouvrier de numéro num.
  - Écrire la fonction *void client (int num) ;* qui définit le **comportement** d'un processus client de numéro num.
  - Écrire le programme principal de l'application lançant NBO processus ouvriers et NBC processus clients en parallèle (NBO et NBC étant des **paramètres** de l'application).

- Le fichier rendu aura pour nom : VotreNomPrenom_tp45_exo4.c

Exemple d'exécution :

%./tp45_exo4 3 2 *[3 ouvriers, 2 clients]*

```
Ouvrier 0 (27) : Réception de la commande du client 30          // Ouvrier n° 0 de pid 27
        Client 0 (30) : Commande envoyée à l'ouvrier 0… je vais l'attendre
<< 30 Infos[0] = 30  Infos[1] = 30  Infos[2] = 30 >>            // Affiché par Ouvrier 0
Ouvrier 1 (28) : Réception de la commande du client 30
<< 30 Infos[0] = 0  Infos[1] = 30  Infos[2] = 30 >>             // Affiché par Ouvrier 1
Ouvrier 2 (29) : Réception de la commande du client 30
<< 30 Infos[0] = 0  Infos[1] = 1  Infos[2] = 30 >>              // Affiché par Ouvrier 2
Ouvrier 0 (27) : Réception de la commande du client 31
                Client 1 (31) : Commande envoyée à l'ouvrier 0… je vais l'attendre
<< 31 Infos[0] = 31 Infos[1] = 31  Infos[2] = 31 >>             // Affiché par Ouvrier 0
        Client 0 (30) : Ma commande est arrivée !
Ouvrier 1 (28) : Réception de la commande du client 31
```

<< 31 Infos[0] = 0 Infos[1] = 31 Infos[2] = 31 >> // *Affiché par Ouvrier 1*

Ouvrier 2 (29) : Réception de la commande du client 31

<< 31 Infos[0] = 0 Infos[1] = 1 Infos[2] = 31 >> // *Affiché par Ouvrier 2*

Client 1 (31) : Ma commande est arrivée !

Fin de l'application

%

À propos de l'affichage effectué par un ouvrier :

La structure de la commande comprend le pid du client émetteur et autant de valeurs que d'ouvriers dans la chaîne (le tableau Infos, de 3 cases ici) initialisées à la valeur de pid du client émetteur.

Quand il traite la commande, un ouvrier affiche la commande puis modifie la valeur qui lui correspond par son numéro dans la chaîne avant de transmettre la commande à l'ouvrier suivant.

Pour aller plus loin…

Modifier et/ou améliorer l'exercice 4 :

- En permettant à un client de passer plusieurs commandes les unes à la suite des autres (le nombre de commandes sera un paramètre de l'application).
- En faisant que certains clients choisissent de ne pas se bloquer en attente de la préparation de leur commande et de continuer une tâche différente (se promener dans le magasin, par exemple, au lieu de rester à attendre devant le comptoir). On pourra faire coexister les deux types de clients dans la même application.

Vous pouvez aussi piocher dans les annales des exercices et les implanter.