TP1 IPC

September 18, 2020

Contents

Quelques exercices sur la thématique des IPC (Inter Processus Communication) "en local" . . . sur les machines de l'UCA ?

➢ Enjoy !

1 Synchronisation d'échanges

Dans le cours, nous avons étudié un exemple d'utilisation des SHM.

➢ Un père et son fils souhaitaient échanger un message.

On en est resté à la gestion de l'exclusion mutuelle sur la mémoire partagée et on a vu que si cela resolvait le problème de l'intégrité des messages, **ce n'était pas suffisant pour que l'échange soit "correct"** (cf page 41 du cours).

On avait alors évoqué le problème de la synchronisation des échanges.

➢ C'est le sujet de cette première question !

Sur la base de l'échange sans sémaphore, on veut évoluer vers ce qui suit :

```
menez ~/EnseignementsCurrent/Cours_Sys_Prog/Tps/TP1_IPC/Src$ : gcc shm_sysv_withsem_synchr.c  -lpthread
menez ~/EnseignementsCurrent/Cours_Sys_Prog/Tps/TP1_IPC/Src$ : ./a.out
After forking, parent waiting for children completion/terminate ...

Child 1 executing...

Child 2 executing...
Message received by child 2 : Hello, I'm child number 1 at 2018-12-19 15:42:08
Message received by child 1 : Hello, I'm child number 2 at 2018-12-19 15:42:08
Message received by child 2 : Hello, I'm child number 1 at 2018-12-19 15:42:09
Message received by child 1 : Hello, I'm child number 2 at 2018-12-19 15:42:08
Message received by child 2 : Hello, I'm child number 1 at 2018-12-19 15:42:10
Message received by child 1 : Hello, I'm child number 2 at 2018-12-19 15:42:09
Message received by child 2 : Hello, I'm child number 1 at 2018-12-19 15:42:11
Message received by child 1 : Hello, I'm child number 2 at 2018-12-19 15:42:10
Message received by child 2 : Hello, I'm child number 1 at 2018-12-19 15:42:12
Message received by child 1 : Hello, I'm child number 2 at 2018-12-19 15:42:11
Message received by child 2 : Hello, I'm child number 1 at 2018-12-19 15:42:13
Message received by child 1 : Hello, I'm child number 2 at 2018-12-19 15:42:12
Message received by child 2 : Hello, I'm child number 1 at 2018-12-19 15:42:14
Message received by child 1 : Hello, I'm child number 2 at 2018-12-19 15:42:13
```

C'est à dire deux processus qui s'échangent, à l'alternat, des messages toutes les secondes.

➢ en utilisant une SHM !

Sous Linux vous pourrez voir cette SHM grâce à la commande `ipcs` .

On vous aide :

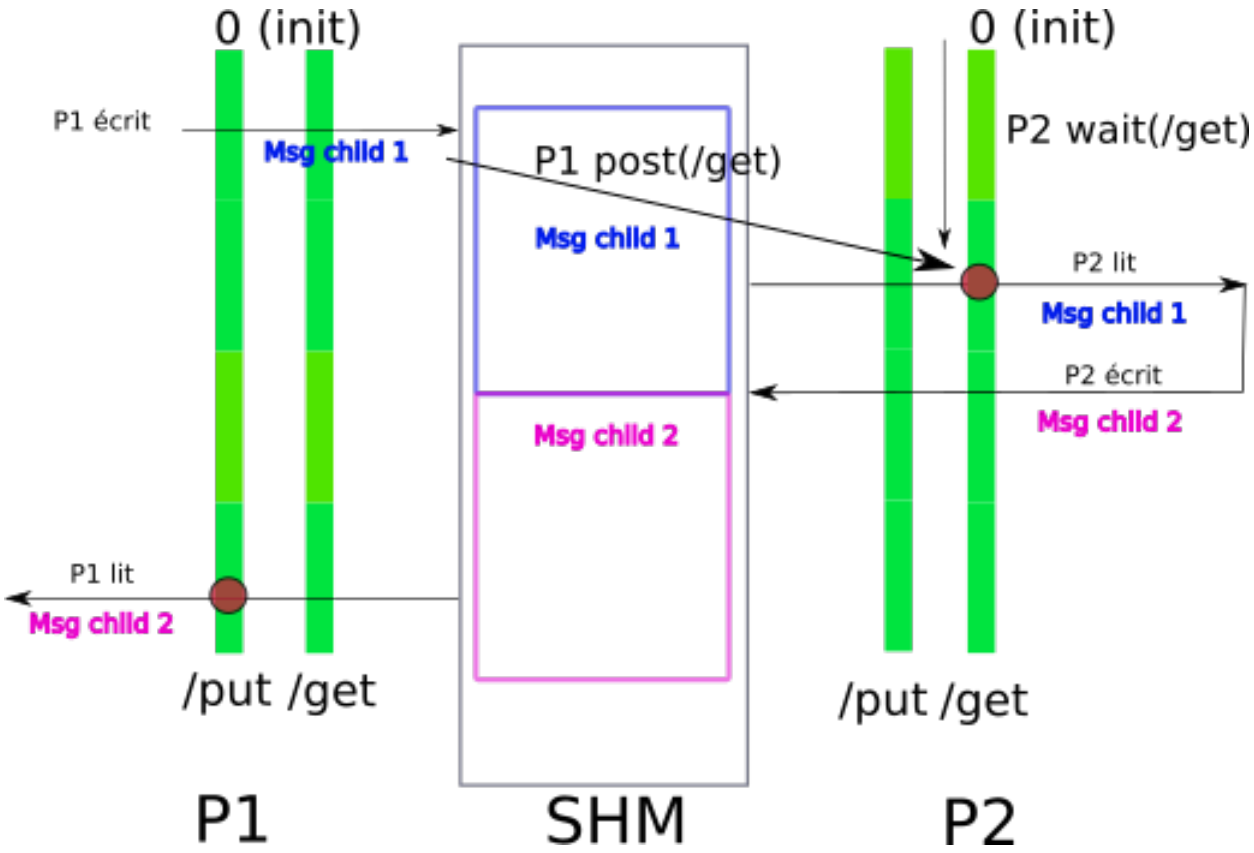
① On vous donne la fonction pour générer des messages contenant l'heure :

```c
#include <time.h>
void make_message(int num, char *message){
  /* Remplit le buffer "message" */
  char buftime[26];
  time_t timer;
  struct tm* tm_info;
  time(&timer);
  tm_info = localtime(&timer);
  strftime(buftime, 26, "%Y-%m-%d %H:%M:%S", tm_info);
  sprintf(message,"%s %d at %s\n", "Hello, I'm child number",  num, buftime);
}
```

② Ensuite, l'idée est d'utiliser deux sémaphores (Dijkstra) pour séquencer les opérations de lecture et d'écriture des deux processus (père et fils).

Derrière mon clavier, je sens que ce n'est toujours pas ça :-)

La figure qui suit illustre le séquencement "théorique/souhaité" des accès sur la shm :



A gauche le processus P1 et à droite le processus P2.

➢ Les sémaphores (représentés par les verticales vertes) sont dessinés des deux cotés (P1, P2) car on rappelle qu'ils sont crées dans le père et donc récupérables par tous les processus !

➢ Le premier sémaphore s'appelle `/put` , il contrôle l'accés en écriture de P2.

Pour avoir le droit d'écrire dans la SHM, il faut que ce sémaphore mette à disposition une clé.

➢ Le second sémaphore s'appelle `/get` , il contrôle l'accés en lecture de P1.

Pour avoir le droit de lire dans la SHM, il faut que ce sémaphore mette à disposition une clé.

On comprend sur la figure le séquencement théorique des opérations, mais on sait que dès le départ, sans synchronisation, il est possible que P2 lise **avant** que P1 n'ait écrit !

Pour éviter cela, je vous donne le début de l'utilisation des sémaphores par P1 et P2 :

① P2 doit attendre pour faire un get (lecture dans la SHM), donc il fait un sem_wait(sur le sémaphore get).

Ainsi il ne peut pas lire dans la SHM tant que le sémaphore n'a pas recu de clé. Il est bloqué !

② Cette clé lui sera donnée par P1 (un sem_post(sur le sémaphore get)) lorsque ce dernier aura fini d'écrire son message dans la SHM.

Voilà, comme cela on est certain que P2 ne lira pas avant que P1 ait écrit !

TODO : Identifier l'autre situation à risque indiquée par le deuxième point rose.

➢ A quel cas, elle correspond ?

➢ A vous de placer sur la figure les opérations sem_wait() et sem_post().

Lorsque vous avez fini de remplir la figure, le codage des deux processus est immédiat (ou du moins plus rapide).

2 Echanges avec tubes

Même exercice que précédemment, à savoir échange de messages, mais **avec des tubes**.

Que devient la synchronisation ? comment est-elle présente ?

3 Sous traitance avec tubes

Dans les TPs du premier semestre, vous avez appris à sous-traiter une tâche à une processus fils :

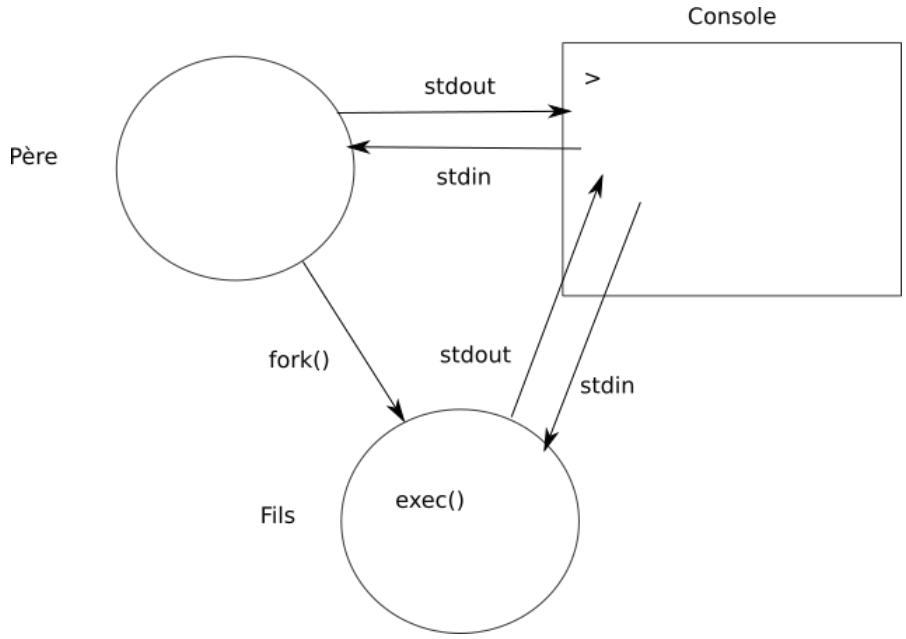
```c
/* Fichier : SANS tube_ls.c
   Le pere cree un processus fils pour sous traiter la commande
   MAIS c'est le fils qui affiche dans la console. */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <string.h>
int main(int argc, char *argv[]) {
  int status;
  char cmd[] = "ls";
  char options[] = "-l";

  /* Le  pere cree un processus fils pour sous traiter la commande */
  if (fork()==0){  /* On est dans le fils */
    execlp(cmd, cmd, options, NULL, NULL);
    exit(2);
  }

  wait(&status);
  return EXIT_SUCCESS;
}
```

Ainsi, dans cet exemple,

- Le processus père fait exécuter une commande au processus fils.
- **Le fils produit le résultat texte dans la même console que son père** : normal il en a hérité !

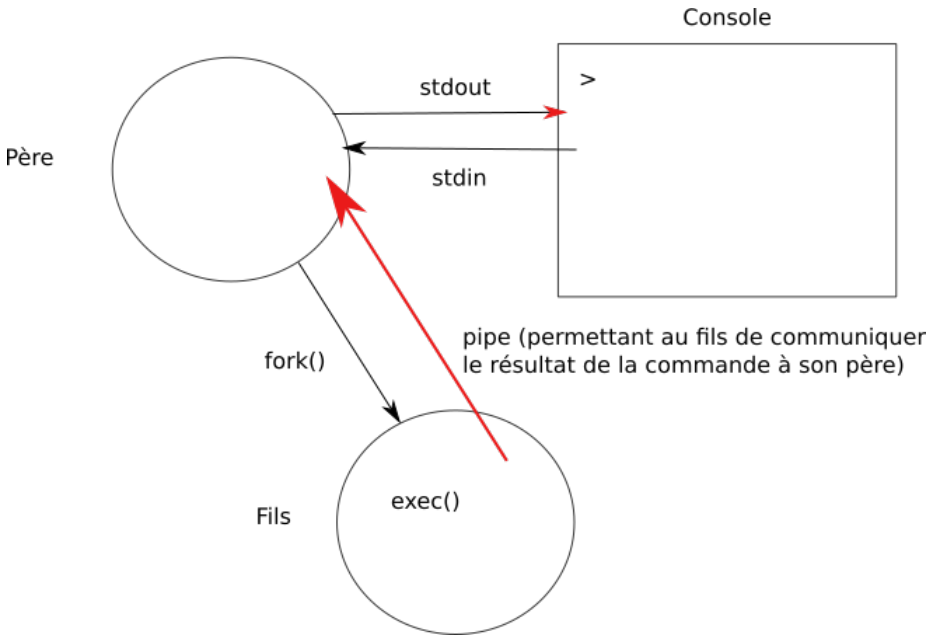


Pour l'exercice qui suit, on se prépare à faire des applications "réseau" ... et l'architecture est modifiée.

Le père reçoit (pour l'instant par le clavier) une commande à traiter et il la sous traite à un fils, **MAIS ce n'est pas le fils qui écrit sur la console**, c'est le père !

➢ **Il faut donc que le fils rende son résultat (du texte sur stdout) via un pipe au père !**

Et c'est le père qui devra écrire sur la console !



Pourquoi cette architecture ?

Ainsi, c'est le père qui pourra gérer la synchronisation des échanges avec les processus distants sur le réseau.

① Ce père, c'est un "serveur" qui gère la liaison réseau

① et qui sous-traite l'exécution d'une commande à un processus fils.

TODO : Je crois bien que le cours répond à cette question ... faire marcher !