CSI 3531
Devoir 1
Date Limite : 26 May 2019

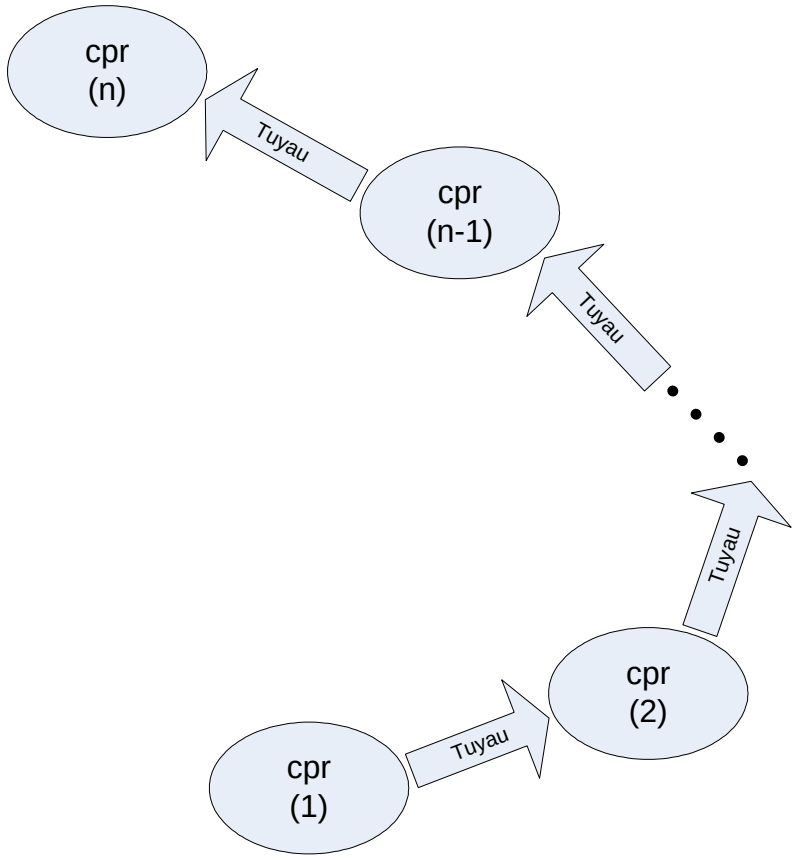
Description

Vous aurez à compléter un programme *cpr* (créer processus) qui devra créer un processus enfant. Le processus enfant à son tour créera un enfant et ainsi de suite pour créer *n* processus. Le fichier *cpr.c* vous est fourni et vous devez le compléter.

La commande pour créer les processus démontrés ci-dessous est « **cpr num** » où *num* est le nombre *n* de processus totaux à créer (c'est-à-dire *n-1* enfants). Le premier processus créé (étiqueté *n*) est créé en exécutant la commande. Ce premier processus crée un enfant en exécutant la commande « **cpr num-1** », c'est-à-dire, décrémente son argument de 1 et crée sont enfant avec la nouvelle valeur. Donc l'enfant créera un autre enfant en décrémentant son argument. Lorsqu'un processus reçoit un argument avec la valeur 1, il ne créera pas d'enfant.

Quand un processus crée un enfant, il doit d'abords créer un tuyau et attacher le bout écrivant du tuyau à la sortie standard de l'enfant avant d'exécuter la commande « **cpr num-1** ». Donc tous les processus enfants (i.e. processus 1 à n-1) écrivent à des tuyaux en écrivant à leur sortie standard.

Tous les processus qui créent des enfants (i.e. processus 2 à n) lisent du bout de lecture du tuyau et écrivent toutes données lues à leur sortie standard. Donc, toute sortie écrite aux tuyaux éventuellement apparaisse à l'écran (via le processus n) après avoir traverser un ou plusieurs tuyaux. Notez que vous n'attachez pas les bouts de lecture des tuyaux aux entrées standards (quoique ceci soit possible).



Les actions prises par les processus comprennent les suivant:

- Processus 1 : écrit tout simplement à sa sortie standard « Processus 1 commence », attend 5 secondes (en utilisant l'appel *sleep(5)*), et ensuite écrit à sa sortie standard « Processus 1 termine ».
- Processus 2 à n : Crée un enfant tel que décrit ci-dessus, écrit aussi à sa sortie standard les mêmes messages que le Processus 1 (en substituant le numéro 1 avec la valeur de l'argument reçu par le processus, 2..n) ainsi que lire du bout de lecture du tuyau pour écrire les données lues à sa sortie standard. Vous devez écrire les messages et données lues dans l'ordre nécessaire pour faire paraître les messages de tous les processus dans l'ordre suivant (dans cet exemple 5 processus ont été crées, i.e. *cpr 5* est exécuté)

```
Processus 5 commence
Processus 4 commence
Processus 3 commence
Processus 2 commence
Processus 1 commence
Processus 1 termine
Processus 2 termine
Processus 3 termine
Processus 4 termine
Processus 5 termine
```

Un délai de 5 secondes devrait avoir lieu entre les messages « Processus 1 commence » et « Processus 1 termine ». Notez qu'un parent n'exécute pas l'appel *wait* car il sait que sont enfant a terminé quand le bout d'écriture du tuyau est fermé (il ne peut plus lire du bout de lecture du tuyau).

Pour compléter le devoir:

1. Commencez avec le fichier fourni *cpr.c*. Complétez la documentation pour indiquer votre nom et numéro d'étudiant. Prenez le temps de bien documenter votre code. Prenez aussi le temps de bien concevoir la solution au problème avant de coder.
2. Complétez la fonction *creerEnfantEtLire(int prcNum)*.
3. Soumettez votre devoir en téléchargeant le fichier *cpr.c*. N'oubliez pas de cliquez sur le bouton soumettre après (et seulement après) avoir télécharger le fichier.
4. Un mot d'avertissement, si vous désirez écrire des messages au terminal durant la création de processus, écrivez à la sortie d'erreur standard (standard error) avec l'appel suivant : *fprintf(stderr,"message\n").*
5. Insérer un délai de 10 secondes (avec *sleep(10)*) avant la terminaison des processus pour introduire un délai après avoir écrit le message « Processus termine ». Durant le délai pour chaque processus, observez l'état des processus avec la commande « ps –u test1 » (remplacez test1 avec votre nom d'utilisateur si vous utilisez une des machines Linux de SITE pour faire votre devoir). Vous noterez qu'un processus est un *zombie*. Expliquez cette observation. Ajoutez votre explication comme commentaire au début de votre code source à l'endroit indiquée.

Information utile:

1. Un descripteur de fichier (file descriptor) est un nombre entier, qui sert de poignée (handle) pour identifier un fichier ouvert. Ce descripteur est utilisé avec d'autres fonctions ou systèmes d'appel tel que *read()* et *write()* pour faire des opérations E/S avec le fichier correspondant.

2. Chez UNIX/Linux, chaque processus à sa création comprend 3 descripteurs de fichier ouvert:
   a. L'entrée standard (standard input), identifier avec le descripteur 0, Donc *read(0,buf,4)* lira 4 octets de l'entrée standard et les copiera dans le tampon *buf*. Normalement, l'entrée standard d'un programme est le clavier de terminal.
   b. La sortie standard (standard output) est identifiée avec le descripteur 1, et normalement correspond à l'écran du terminal.
   c. La sortie d'erreur standard (standard error) est identifiée avec le descripteur 2, et normalement correspond à l'écran du terminal.
   d. En fait, lorsqu'un processus est démarré d'une commande tapée dans le shell, l'entrée standard, la sortie standard, et la sortie d'erreur standard sont trois descripteurs qui réfèrent au terminal (fichier tty) du shell. Donc lire du fichier tty correspond à la lecture du clavier tandis qu'écrire au fichier tty correspond à l'écriture sur l'écran du terminal.
   e. Notez que plusieurs fonctions de la librairie standard C utilisent ces descripteurs par défaut. Par exemple, *printf("chaine")* écrit « chaine » sur la sortie standard (normalement sur l'écran).
   f. Il est possible de demander au shell UNIX de connecter la sortie standard d'un processus à l'entrée standard de l'autre processus avec un tuyau. Il s'agit d'intercaler le caractère « | » entre deux commandes : par exemple « who | wc ».

Dans cet exemple, le shell créera un tuyau UNIX qui connectera la sortie standard du processus who à l'entrée standard du processus wc. Donc, les données écrites par le processus who utilisant le descripteur 1 (sortie standard) sont envoyées dans le tuyau. Ces données seront lues par le processus wc qui lie ses données utilisant le descripteur 0 (entrée standard), c'est-à-dire, du tuyau.

3. Vous utiliserez les fonctions C suivantes:
   a. fork() – étudier en classe (pour plus d'information détaillée, utilisez la commande man fork)
   b. pipe()
      - Introduit aussi en classe.
      - Notez qu'il est possible d'attacher plusieurs processus à chaque bout d'un tuyau. Le SE maintien le tuyau, tant que des processus sont attaché aux bouts du tuyau.
   c. execvp(const char * program, const char *args[]) (d'autres formes de cet appel existe tel que execlp(const char *program, const char *program, ….))
      - Remplace l'image du processus avec le programme spécifié dans le premier argument de la fonction.
      - Le deuxième argument est un tableau de chaînes de caractères qui seront donné au nouveau programme comme arguments (le tableau est terminé avec NULL).
      - Un convention dicte que args[0] est le nom du fichier du programme à être exécuté.

   d. dup2(int oldfd, int newfd) – clone (duplicate) le descripteur oldfd sur le descripteur de fichier newfd. Si newfd correspond à un fichier ouvert, ce fichier sera d'abords fermé. Donc le même fichier peut être accéder soit par oldfd, soit par newfd (essentiellement deux connections au fichier). Voyez http://mkssoftware.com/docs/man3/dup2.3.asp ou "man dup2" pour plus d'information. Par exemple, le programme suivant :

```
int main(int argc, char *argv[]) {
    int fd;
    printf("Bonjour le monde!")
    fd = open("outFile.dat", "w");
    if (fd != -1) dup2(fd, 1);
          printf("Bonjour le monde!");
     close(fd);
}
```

   rédigera la sortie standard du programme au fichier "outFile.dat". Le premier « Bonjour le monde! » sera imprimer sur la console, tandis que le deuxième sera stocké dans le fichier « outFile.dat ».
   e. *read(int fd, char *buff, int bufSize)* – lire du fichier (ou tuyau) identifier par le descripteur *fd* un nombre de caractères *bufSize* et copier les caractères lu dans le tampon *buff*. La fonction retourne le nombre de caractères lus (peut être moindre que *bufSize*), ou -1 lors d'une erreur, ou 0 lorsque la fin du fichier est atteint (dans

le cas du tuyau, aucun processus est attaché au bout écrivant et aucune donnée existe dans le tuyau).

f. *write(int fd, char *buff, int buffSize)* – Écrit dans le fichier (ou tuyau) *fd* le nombre de caractères *buffSize* retrouver dans le tampon *buff*.
g. close(int fd) – ferme un fichier ouvert. Le descripteur *fd* pourra être réutilisé.
h. Vous voudrez probablement utiliser la fonction *printf()* pour formater votre sortie. Cette fonction écrit à la sortie standard (df 1). Mais attention car cette fonction met en tampon les données de sortie et n'écrit pas immédiatement à la sortie standard (fait un *write* à df 1). Pour forcer une écriture immédiate, utilisez *fflush(stdout)*. Une autre alternative serait d'utilisez *sprintf()* pour formater la sortie dans un tampon de mémoire et ensuite utilisez *write()* pour écrire les données dans le tampon à la sortie standard.

4. Voici un indice comment observer les références des processus aux tuyaux :
   a. Ajoutez un long délai (disons 300 secondes) à différents points dans votre code. Lorsque vous exécutez votre programme, divers processus seront créés que vous pouvez examiner à travers le répertoire */proc*. Obtenez les PIDs (identificateur de processus) des processus créés avec la commande « ps –u test1 » (remplacez test1 avec votre nom d'utilisateur si vous utilisez une des machines Linux de SITE pour faire votre devoir). Vous pouvez trouvez ce que les différents descripteurs de fichier réfèrent en examinant le répertoire *fd* du processus comme suit :

```
[test1@sitedevga W2007]$ ps -u test1
  PID TTY          TIME CMD
 1114 ?        00:00:08 sshd
 1115 pts/0    00:00:00 bash
 1210 pts/1    00:00:00 bash
 1362 tty1     00:00:00 bash
 1987 pts/0    00:00:00 cpr
 1988 pts/0    00:00:00 cpr
 1989 pts/0    00:00:00 cpr
 1990 pts/1    00:00:00 ps
[test1@sitedevga W2007]$ ls -lR /proc/1988/fd
/proc/1988/fd:
total 0
lrwx------  1 test1  test1  64 Jan 18 11:50 0 -> /dev/pts/0
l-wx------  1 test1  test1  64 Jan 18 11:50 1 -> pipe:[11950]
lrwx------  1 test1  test1  64 Jan 18 11:50 2 -> /dev/pts/0
lr-x------  1 test1  test1  64 Jan 18 11:50 3 -> pipe:[11951]
```

Permission d'écriture activée indique que ceci est le bout d'écriture du tuyau.

Notez que pour le processus ayant le PID 1988, la sortie standard (fd 1) réfère à un tuyau. En plus il est attaché au bout d'écriture du tuyau car la permission d'écriture est activée du tuyau. Notez aussi que df 3 est attaché au bout de lecture d'un tuyau différent (car sont identificateur 11951 est différent).

Linux/UNIX offre des pages de documentation (man pages). La commande *man « nom de fonction »* imprimera à l'écran de l'information détaillé au sujet de la fonction donnée.