Kyle ESPARON
Alexis JAMAL

Compte-rendu global

1.1 Initiation à Linux

1.1.1 Traitement de flux

Un tube redirige la sortie standard de la commande à gauche de celui-ci vers l'entrée standard de la commande à sa droite.

La redirection permet d'affecter un flux de données vers un fichier particulier. C'est ainsi qu'il sera possible de récupérer le résultat d'une commande dans un fichier, ou encore d'envoyer un fichier vers une commande. (1)

Ces redirections sont liées aux descripteurs de fichiers car les trois différents fichiers ouverts par défaut, stdin (le clavier), stdout (l'écran) et stderr (la sortie des messages d'erreur vers l'écran) possèdent un descripteur de fichier utilisé pour la redirection. En effet, on redirige la sortie standard [descripteur 1 : la sortie standard (stdout en C)] d'une commande vers l'entrée standard [descripteur 0 : l'entrée standard (stdin en C)] d'une autre commande ou d'un fichier. (2)

1.1.2 Finger

Première commande : (3)

```
awk -F: '($3 >= 1000) {printf "%s:%s\n",$1,$3}' /etc/passwd
```

Deuxième commande :

```
cut -d: -f1-3 /etc/passwd | egrep [0-9]{4}
```

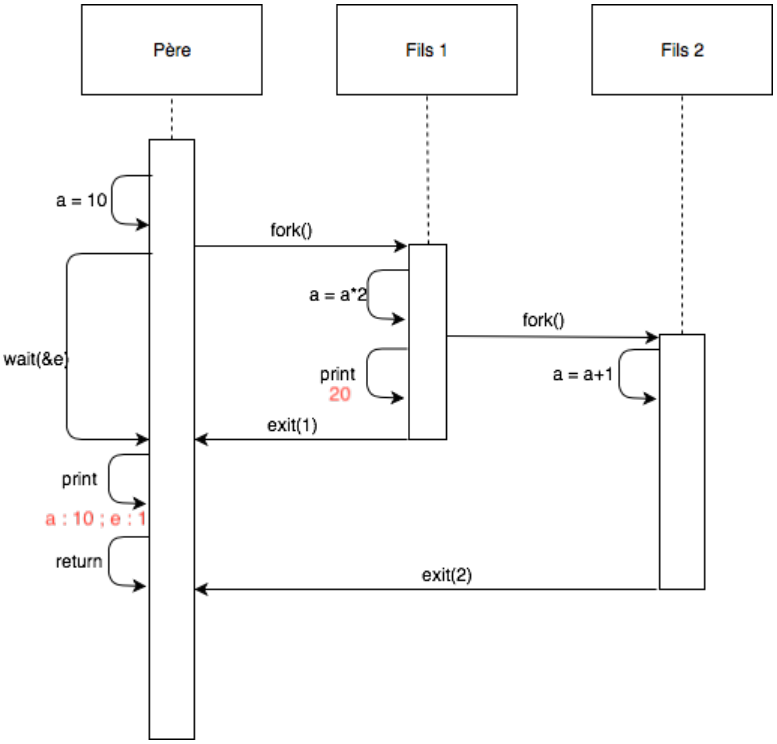
1.2 Processus

1.2.1 Fork

Détails du programme :

La fonction fork() crée un fils au processus courant, la fonction wait() attend que l'un de ses fils se termine et exit() tue le processus courant (en passant en paramètre le statut de sortie).

Diagramme de séquence :



1.2.2 Killbill

On crée un zombie ainsi (son PID est 2653) :

```
esparon@esparon-virtual-machine:~/Téléchargements/SE2016.ESPARON_JAMAL/bin$ ./zo
mbie
2653 p GRRRRRRRRRRRR! prroooceeesseurrr …      C  P  U  mmmééémmmmoiiiire
```

On récupère le PID du zombie passé en paramètre de la commande (donc 2653) qu'on convertit en int avec la fonction atoi. Ensuite nous récupérons le PID du père du zombie grâce à la fonction parentpid, que l'on passe en paramètre de la fonction kill, avec le signal SIGBUS.

```
int pid = atoi(argv[1]);
kill(parentpid(pid), SIGBUS);
```

Le zombie a bien été tué :

```
R   A   M       pprrooooceeeesseurrr …    C  P  U
    _____________________
   |,----.,----.,----.,--.\
   ||    ||    ||    ||   \\
 = |`----'`----'|----||----\`.
   [            |   -||- __|(|
 _ [  ,--.      |____||.--.  |
   =-(( `))-----------(( `))==
      `--'             `--'
Et c'est ainsi que le zombie se fit tué par un bus.
```

1.2.3 Comprendre les zombies

Un processus zombie (on utilise plutôt l'orthographe anglaise) est un terme désignant un processus qui s'est achevé, mais qui dispose toujours d'un identifiant de processus (PID) et reste donc encore visible dans la table des processus. (4)

Pour éliminer un processus zombie il faut tuer son père.

Le programme zombie affiche "GRRRRRRRR !" quand il reçoit un signal autre que SIGBUS. Lorsque nous lançons le programme, le système d'exploitation tente d'arrêter le processus zombie, ce qui explique le grognement.

Bonus

Programme chucknorris

Voici la ligne de commande. Nous récupérons tous les PID des zombies avec "pidof zombie". Nous redirigeons la sortie standard de cette commande vers l'entrée standard de "awk '{ print \"kill -7 \" $2 }' qui permet d'afficher sur la sortie standard kill -7 [PID du zombie] pour tuer chaque zombie. La commande est exécutée avec "sh".

```
system("pidof zombie | awk '{ print \"kill -7 \" $2 }' | sh");
```

2.1 Puzzle de caractères

2.1.1 Sémaphores

Disposition des appels des sémaphores:

```
if(!fork()) {
    struct sembuf up = {(i+1)%6, 1, 0}; //number, operation, flag
    struct sembuf down = {(i)%6, -1, 0}; //number, operation, flag
    while (getline(&line, &len, fp) != -1) { // Tant qu'on peu lire

        semop(semid, &down, 1);
        printf("%s", line); // on affiche !
        fflush(stdout);
        semop(semid, &up, 1);
    }
    if (line){
        free(line); // et de libérer la mémoire
        exit(EXIT_SUCCESS);

    }
```

2.1.2 Preuve

Notre programme n'utilise qu'un seul jeton qui est passé de sémaphore en sémaphore et l'affichage n'est réalisé uniquement quand un sémaphore possède le jeton, comme on le voit dans le code ci-dessus, l'affichage est réalisé entre le down et le up.

4.1 À ne pas oublier

4.1.1 C’est bien de partager

Les processus ont besoin de mémoire partagée pour pouvoir interagir entre eux, partager des variables… Pour que la mémoire de chaque processus soit protégée il faut qu’il utilise la zone de mémoire qui lui est allouée, c’est le système d’exploitation qui s’en charge.

4.1.2 Se mettre à la page

Lorsqu’un processus tente d’accéder à une adresse qu’il ne lui est pas allouée, on obtient un core dumped. C’est le système d’exploitation qui va retourner cette erreur pour protéger le reste de système.

4.2 Morpion

4.2.1 Rien ne va plus

On peut gérer les attentes et la synchronisation avant l’accès mémoire grâce à des sémaphores. L’arbitre ne peut luter contre la fraude s’il n’est pas lui même dans la section critique car les processus ont accès à la mémoire partagée uniquement dans la section critique.

Références

1 - https://askubuntu.com/questions/172982/what-is-the-difference-between-redirection-and-pipe

2 - https://shell.figarola.fr/x54.html

3 - https://askubuntu.com/questions/645236/command-to-list-all-users-with-their-uid

4 - https://fr.wikipedia.org/wiki/Processus_zombie

5 - http://www.courstechinfo.be/OS/GestMem.html