Raphaël CASTANIER

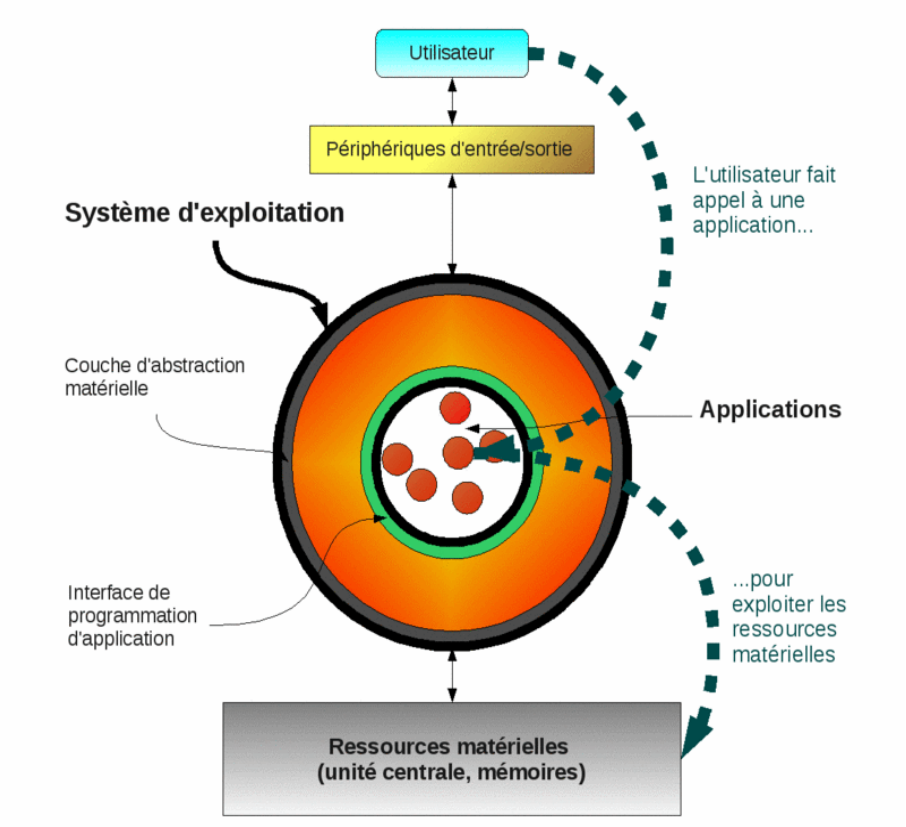
Joffrey RIUS BERNAL

Groupe TP2/ OS308

Grenoble INP ESISAR

Compte Rendu TP6

Manipulation des signaux -



Les relations entre utilisateur, applications, système d'exploitation et matériel

Année Académique 2017/2018

Sommaire

Exercice1 : Synchronisation père/fils par des signaux

Enoncé :

Ecrire un programme qui crée deux processus, père et fils. Le père affiche les nombres entiers impairs compris entre 1 et 100, alors que le fils affiche les entiers pairs compris dans le même intervalle. Synchroniser les processus à l'aide des signaux pour que le résultat d'affichage soit : 1 2 3 ... 100.

Commentaires :

Afin de réaliser cet exercice, nous avons réalisé une fonction « `doubleIncr` » qui `incrémente` de deux une variable globale. Cette fonction est appelée successivement par le fils puis par leur père qui incrémente leurs valeurs « var » chacun de leur côté.

Cet appel en « ping pong » est géré par l'appel de la fonction « kill » qui permet l'envoi du SIGUSR1, ce qui déclenche l'appel de la fonction « doubleIncr ». Afin que l'appel soit fait l'un après l'autre nous avons géré l'arrêt puis le redémarrage du processus fils ainsi que celui du père.

L'objectif étant que le fils doit afficher les entiers pairs et que le père affiche les entiers impairs, nous avons fait démarrer la variable « var » à :

- 0 pour le fils afin qu'il incrémente la variable de deux en deux et alors par ce fait affiche uniquement les entiers pairs.
- -1 pour le père qui affichera alors à contrario du fils uniquement les entiers impairs.

Une fois que la variable de comptage a atteint 100, les processus père et fils s'arrêtent.

Exercice 2 : Horloge

Enoncé :

Ecrire un programme C qui utilise 3 processus H, M, S qui incrémentent les 3 « aiguilles » d'une horloge. S reçoit un signal SIGALRM chaque seconde (émis par la fonction alarm(1)) et émet un signal à M quand son compteur passe de 59 à 0. Quand M reçoit un signal, il incrémente son compteur. Quand son compteur passe de 59 à 0, M envoie un signal à H. Les paramètres correspondent aux valeurs d'initialisation des compteurs.

Commentaires :

Cet exercice possède 3 processus ce qui justifie alors les deux appels à la fonction « fork() ». Une fois les processus fils et pères créés, nous avons défini l'utilité de chacun. Ainsi, dans le dernier processus créé qui est alors le processus S, l'appel de la fonction alarme(1) permet d'envoyer le signal SIGALRM à lui-même. Ainsi, le signal SIGALRM appelle la fonction « sec_handler ».

Une fois que cette fonction handler est appelée, elle incrémente les secondes et appelle alarme(1) de nouveau. Etant à nouveau dans cette fonction, la fonction « sec_handler » s'appelle en boucle. Ainsi, cela cadence ce processus à 1Hz.

Une fois que ce cadencement est réalisé et conforme, arrivé à 59 secondes après le premier le coup, nous réalisons un « `kill(getppid(),SIGALRM);` » qui permets le déclenchement de l'incrémentation des minutes. Comme on peut observer au-dessus le « kill » est fait sur le père car c'est lui qui permet la gestion des minutes.

La méthode de déclenchement des heures est similaire à celle des minutes car arrivée à 59 min, la minute suivant provoque incrémentation d'une heure et remet les minutes ainsi que les secondes à 0. Cette gestion des secondes/minutes/heures est indépendante au vu de la gestion de chaque entité par un processus distinct.

Exercice 3 : Roulette russe

Enoncé :

Un père fait jouer six fils à la roulette russe. Le père crée six fils, numérotés de 1 à 6. Une fois ses fils crées, il envoie un signal au premier pour lui indiquer de commencer à jouer, à la réception de ce signal le fils lit un fichier « barillet » alimenté manuellement ou par le père avec un entier de 1 à 6. Si l'entier lu par le fils est égal à son numéro, le fils se tue (il s'arrête). Sinon, il envoie un signal au père pour l'avertir qu'il est encore vivant. Si le fils a survécu, le père passe au fils suivant. Si le fils est mort, le père annonce à l'aide d'un signal aux autres fils qu'ils sont libres. Ceux-ci s'en réjouissent et s'arrêtent de jouer.

Commentaires :

Nous avons réalisé ce code en différentes parties (fonctions callback et main) dans le but de pouvoir regrouper les actions similaires réalisées pour chaque processus.

Fonction « jouer » :

Grâce au code donné en annexe, nous allons lire une variable dans le fichier « barillet » afin que celui-ci nous retourne un entier. Par la suite chaque processus entrant dans cette fonction va comparer son « numéro de fils » avec la valeur lue en amont dans le fichier. Si la comparaison est dite 'vraie' (variable identique) le fils envoie un signal à son père et se tue en « EXIT_FAILURE ». Sinon il indique à son père qu'il est toujours vivant.

Fonction « libérer » :

Cette fonction est appelée via « SIGCHLD » ce qui indique qu'il y a eu un processus fils de terminé. C'est un fils qui est mort et donc la fin de la partie. Afin que les fils restants soient libérés, le père leur envoie un signal « SIGUSR2 » pour qu'ils passent chacun leur tour par la fonction « finir ».

Fonction « finir » :

Une fois qu'un fils est dans cette fonction c'est que le jeu est fini. Il s'en réjouit et s'arrête de jouer.

Fonction « main» :

Afin que le fichier « barillet » soit initialisé, la méthode « ramdom » est appelée. Elle renvoie un entier, qui est modifié pour être un numéro de fils :

```
srand(time(NULL));                // initialisation de rand

int valeur = rand()%NB_FILS+1;    // valeur aléatoire comprise entre 1 et NB_FILS
```

Elle permet d'envoyer au fichier un entier au hasard permettant de sélectionner en aval le fils qui devra mourir.

Ensuite les signaux : SIGUSR1 et SIGUSR2 sont paramétrés afin que chaque signal dirige les processus fils vers la fonction appropriée.

```
if(signal(SIGUSR1, jouer)==SIG_ERR) assert(0); // fonction appelée lors d'un appel du père (avec assertion)
if(signal(SIGUSR2, finir)==SIG_ERR) assert(0); // fonction appelée lors d'un appel du père (avec assertion)
```

Afin de créer plusieurs fils tout en gardant un unique père, nous avons réalisé un tableau de numéro de fils qui s'incrémente et conserve le PID fils à chaque passage du père dans le fork() .Cette boucle « for » est réalisée de telle façon qu'une fois le fils crée il ne peut pas créer à son tour un autre fils.

Une fois que le père a fini la création de ses fils, il appelle successivement chaque fils à aller jouer à la roulette russe (fonction jouer). L'ensemble des fils sont endormis en attendant l'appel du père.

Fonctions « annexe» (lecture/écriture fichier):

Ces fonctions proviennent du sujet (données en annexe) mais il nous a semblé utile d'en détailler le fonctionnement pour montrer que nous avons compris leur fonctionnement.

La fonction « lire_valeur » possède un paramètre qui est le chemin du fichier à lire. Une fois le chemin du dossier défini, si le fichier n'est pas null, nous lisons le fichier afin de ranger l'entier présent dans le fichier dans une variable. A la fin de la fonction la variable est retournée afin que la fonction « jouer » qui appelle « lire_valeur » possède la variable.

Afin d'écrire dans un fichier, nous avons utilisé la fonction « ecrire_valeur ». Cette dernière est également paramétrée par un 'path' (chemin) mais aussi par une variable entière qui sera écrite dans le fichier. Afin d'écrire dans ce fichier, il est ouvert puis si 'fopen' ne renvoie pas null, la variable en paramètre est écrite dans le fichier via un 'printf'. Puis le fichier est fermé.

Conclusion

Lors de ce TP, nous avons pu mettre en pratique un grand nombre de compétences vues en cours (CM) dont l'utilisation du fork(), des signaux, des appels fonctions et des callback. Lors de ces 3 exercices indépendants, nous avons pu organiser et programmer la gestion de différents processus afin que ceux-ci effectuent des actions en communication les uns avec les autres ou, que leur fonctionnement soit distinct comme pour chaque fils de l'exercice « Roulette Russe ». L'utilisation de la redirection des processus via l'appel 'Signal' a été très fréquemment utilisé lors de la programmation de ce TP et nous a donc permis de pouvoir se familiariser avec son fonctionnement. Cette pratique nous aura aussi permis de comprendre la méthode 'callback' puis comprendre l'utilisation de 'Alarme(1)' qui nous était inconnue.

De plus, étant donné que l'ensemble des exercices est programmé en C cela nous a permis de continuer notre apprentissage de ce langage.