L3

Examen final d'Architecture Système Avancé – 9 janvier 2014 – session 1

Michel SOTO

AUCUN DOCUMENT AUTORISÉ

Durée: 1 H 30

Le barème est indicatif - Nombre de pages: 3 sur 2 feuilles recto verso

La concision de vos réponses et la propreté de votre copie seront prises en compte.

PARTIE I : CONNAISSANCE ET COMPREHENSION DU COURS

Question 1 (3 points)

Répondez aux affirmations suivantes uniquement par "VRAI", ou "FAUX" ou "NE SAIS PAS".

Barème : **réponse exacte : +1 point, réponse fausse : -0,5 point sur la copie, "ne sais pas" : 0 point**

a) La pose, par un processus P, d'un verrou exclusif en mode *advisory* sur une zone de fichier interdit l'accès à cette zone à tout autre processus que P

b) La primitive `stat` retourne les statistiques d'utilisation d'un fichier

c) Sous Linux un **`fork`** effectué par un thread a pour effet de cloner tous les threads du processus.

Question 2 (2 points)

Ennoncez les diffrérences qui existent entre un fichier régulier et un tube ordinaire.

PARTIE II : APPLICATION DU COURS

Question 3 (8 points)

Soit 2 processus P_1 et P_2 créés par même processus P. P_1 doit exécuter les actions A et B. P_2 doit exécuter les actions C et D. Les actions A et C doivent être exécutées en premier. Elles peuvent être exécutées en parallèle ou dans n'importe quel ordre entre elles. L'action D ne doit être exécutée qu'après l'action A. Enfin l'action B ne doit être exécutée qu'après l'action D. Lorsque ses fils sont terminés, P affiche **`FIN`**.

a) En utilisant uniquement les tubes ordinaires, écrivez en C le programme réalisant les actions A, B, C et D dans l'ordre voulu. Les actions elles-mêmes se limiteront à un **`printf(Pi : action X\n)`** aux endroits concernés du programme

b) Pendant les phases de synchronisation, on souhaite maintenant que P_1 affiche :

- "`J'attends`" lorsque P_2 n'a pas terminé D alors que lui est prêt à exécuter B
- "`Je n'attends pas`" lorsque P_2 a terminé D alors que lui est prêt à exécuter B

et que P_2 affiche :

- "`J'attends`" lorsque P_1 n'a pas terminé A alors que lui est prêt à exécuter D
- "`Je n'attends pas`" lorsque P_1 a terminé A alors que lui est prêt à exécuter D

Proposez une fonctionnalité du système (Linux) permettant d'atteindre ce résultat sans utiliser aucun autre moyen de communication que les tubes ordinaires déjà utilisés en a).

c) Ecrivez l'algorithme d'une phase de synchronisation reposant sur la fonctionnalité proposée en b).

d) Complétez le code de la question a) en codant l'algorithme écrit en c).

Question 4 (8 points)

Ci-dessous, le code du minishell que nous avons vu en TD.

a) Dans quelle situation ce mini shell produit-il des processus à l'état zombi ? Justifiez votre réponse.

b) Proposez une solution pour faire disparaître systématiquement chaque zombi produit par ce mini shell.

c) Ecrivez le code de la solution que vous proposez en indiquant où il se situe dans le code ci-dessous.

d) Ce mini shell ne permet pas à son utilisateur de rediriger l'entrée ou la sortie de ses commandes. Complétez le code afin que cela soit désormais possible. Indiquez où intervient votre code dans le code existant.

```
#include <stdio.h>
#include <string.h>
#include <signal.h>
#include <unistd.h>

#define TAILLE_LIGNE 250
#define TAILLE_ARG    40
#define EVER (;;)

char commande [TAILLE_LIGNE];

/*================================================*/
  main(){
/*================================================*/
char *arg[TAILLE_ARG];
char *bg, char *mot;
int i, pid, CodeRetour;

/*1*/for EVER {
/*2*/    printf ("mishell> ");

     // Lecture de la ligne de commande
     // entrée par l'utilsateur
/*3*/   fgets(commande, TAILLE_LIGNE, stdin);

     // Suppression du retour chariot
/*4*/   commande[strlen(commande)-1]='\0';

     // Recherche et suppression du & eventuel
/*5*/   if (bg=strrchr(commande, '&')) *bg='\0';

    // Recherche des mots separés par un blanc
/*6*/  for(i=0,mot=strtok(commande, " ");
           mot!=NULL;
              mot=strtok(NULL," "),i++){

     // Preparation des parametres du execvp
/*7*/   arg[i]=(char *)malloc(strlen(mot)+1);
/*8*/   strcpy(arg[i],mot);

/*9*/  }//for
/*10*/ arg[i]=NULL;

    // La ligne de commande est-elle vide ?
/*11*/ if (i>0)
/*12*/     if (fork()==0) {// CODE DU FILS
/*13*/         printf("\n");
/*14*/         execvp (arg[0], arg);
/*15*/         perror ("execvp");
/*16*/      }

/*17*/      else {// CODE DU PERE
/*18*/            if (bg!=NULL)
/*19*/               // Pas d'attente
/*20*/            else {// Attente
/*21*/                  pid=wait(&CodeRetour);
/*22*/            }// if (bg!=NULL)
/*23*/      }// if (fork()==0)

/*24*/    }// for EVER
/*25*/ }// main()
```

ANNEXE 1

SYNOPSYS
int get_in (char *cde, char *file)
int get_out (char *cde, char *file)

DESCRIPTION
La fonction get_in copie dans la chaîne pointée par file le 1er mot suivant le caractère '<' si ce dernier est présent dans la chaîne pointée par cde. Dans ce cas, le caractère '<' et le 1er mot qui le suit sont supprimés de la chaîne pointée par cde.
La fonction get_out effectue le même travail mais pour le caractère '>'.

VALEUR RENVOYEE
Les fonctions get_in et get_out renvoient une valeur négative en cas d'erreur, une valeur supérieure à zéro si le caractère '<' ou, respectivement, le caractère '>' est présent dans cde, zéro sinon.

ANNEXE 2

SIGHUP	terminaison du processus leader de session
SIGINT	frappe du caractère intr sur le clavier du terminal de contrôle
SIGQUIT	frappe du caractère quit sur le clavier du terminal de contrôle
SIGILL	détection d'une instruction illégale
SIGABRT	terminaison anormale provoquée par l'exécution de la fonction abort
SIGFPE	erreur arithmétique (division par zéro, …)
SIGKILL	signal de terminaison
SIGSEGV	violation mémoire
SIGPIPE	écriture dans un tube sans lecteur
SIGALRM	fin de temporisation (fonction alarm)
SIGTERM	signal de terminaison
SIGUSR1	signal émis par un processus utilisateur
SIGUSR2	signal émis par un processus utilisateur
SIGCHLD	terminaison d'un fils
SIGSTOP	signal de suspension
SIGTSTP	frappe du caractère susp sur le clavier du terminal de contrôle
SIGCONT	signal de continuation d'un processus stoppé

ANNEXE 3

SYNOPSIS
```
#include <unistd.h>
#include <fcntl.h>

int fcntl(int fd, int cmd, ... /* arg */ );
```

DESCRIPTION
fcntl() permet de se livrer à diverses opérations sur le descripteur de fichier fd. L'opération en question est déterminée par la valeur de l'argument cmd.

fcntl() prend un troisième paramètre optionnel. La nécessité de fournir ce paramètre dépend de cmd.

Attribut d'état du fichier
Un descripteur de fichier dispose de certains attributs, initialisés par open(2) et éventuellement modifiés par fcntl().

F_GETFL
Lire les attributs d'état du fichier ; arg est ignoré.

F_SETFL
Positionner les nouveaux attributs pour le descripteur de fichier à la valeur indiquée par arg. Sous Linux, cette commande ne peut changer que O_APPEND, O_ASYNC, O_DIRECT, O_NOATIME et O_NONBLOCK.

VALEUR RENVOYEE
La valeur renvoyée par fcntl() varie suivant le type d'opération :

F_GETFL La valeur des attributs.
Toutes les autres commandes : Zero.