Noyau d'un système d'exploitation
INF2610

QCM 2 – Threads & Communication interprocessus

Département de génie informatique et génie logiciel

POLYTECHNIQUE MONTRÉAL
AFFILIÉE À L'UNIVERSITÉ DE MONTRÉAL

Automne 2017

QCM 2 – Threads et communication interprocessus

Répondez aux questions à choix multiples en sélectionnant une ou plusieurs réponses. Les questions font référence aux systèmes d'exploitation de la famille UNIX et considèrent les options par défaut des appels système.

QCM 2 - Threads

1) Lorsqu'un thread d'un processus se termine en invoquant « exit », :
a) le système force la terminaison de tous les threads du processus.
b) le système termine uniquement le thread invoquant « exit ».
c) Les processus fils sont adoptés par le processus init.
d) le système termine uniquement le thread invoquant « exit » et ses threads descendants.
e) aucune de ces réponses.

2) Quels sont les éléments partagés par l'ensemble des threads d'un même processus ? Ils partagent :
a) l'espace d'adressage.
b) la table des descripteurs de fichiers.
c) le compteur ordinal.
d) la pile d'exécution.
e) les gestionnaires de signaux.
f) le masque des signaux du processus.
g) les signaux en attente du processus.
h) aucun des éléments ci-dessus.

QCM 2 - Threads

3) Considérez le code suivant (sans les directives d'inclusion) :

```
int a=0;
void *increment(void *) { a++; return NULL;}
int main ()
 { pthread_t th[3];
   int i;
   for(i=0; i <3; i++)
          pthread_create(&th[i], NULL, increment, NULL);
   for(i=0; i <3; i++)
          pthread_join(th[i], NULL);
   printf(" a = %d\n", a);
   return 0;
}
```

Le programme est-il déterministe ?

a) Oui, car il affichera toujours la valeur 3.

b) Non, car il affichera 1, 2 ou 3.

c) Non, car il affichera 0, 1, 2 ou 3.

d) Oui, car il affichera toujours 0.

e) aucune de ces réponses.

QCM 2 - Threads

4) Considérez le code suivant (sans les directives d'inclusion) :

```
int a=0;
void *increment(void *) { a++; return NULL;}
int main ()
 { pthread_t th[3]; int i;
   for(i=0; i <3; i++)
          pthread_create(&th[i], NULL, increment, NULL);
   printf(" a = %d\n", a);
   return 0;
}
```

Le programme est-il déterministe ?

a) Oui, car il affichera toujours la valeur 3.

b) Non, car il affichera 1, 2 ou 3.

c) Non, car il affichera 0, 1, 2 ou 3.

d) Oui, car il affichera toujours 0.

e) aucune de ces réponses.

QCM 2 - Threads

5) Un processus P crée deux threads utilisateur puis réalise ce qui suit :

«int i=100; while(i--); sleep(3); while(i++<100); exit(0);».

Chaque thread créé incrémente une variable globale v puis se termine.
Chaque thread créé peut s'exécuter :
a) pendant que P est en pause (*sleep(3)*) en concurrence avec l'autre thread.
b) pendant que P est en pause mais pas en concurrence avec l'autre thread.
c) avant/après la pause de P et avant « exit(0) », mais jamais en concurrence avec P ou l'autre thread.
d) après « exit(0) ».
e) aucune de ces réponses.

QCM 2 – Communication interprocessus

6) Considérez la commande « *ls* | *sort* > *data* ». La sortie erreur STDERR du processus exécutant « ls » est :
a) le fichier data.
b) le pipe.
c) l'écran.
d) le clavier.
e) aucune de ces réponses.

7) Supposez qu'un processus père crée un processus fils en utilisant « fork ». Si le processus fils redirige STDOUT vers un fichier FICH (ce fichier est ouvert par le processus père avant l'appel à « fork »),
a) la sortie standard du père est aussi redirigée vers FICH.
b) la sortie standard du père reste inchangée.
c) la sortie standard du père sera fermée.
d) le père va perdre l'accès au fichier FICH.
e) aucune de ces réponses.

QCM 2 – Communication interprocessus

8) On veut faire communiquer deux threads *utilisateur* d'un même processus via un tube anonyme (*pipe*). À quel niveau doit-on créer le tube anonyme ?
Le tube anonyme doit être créé :
a) avant la création du premier thread.
b) après la création du premier thread et avant la création du second thread.
c) après la création du second thread.
d) dans chacun des deux threads.
e) aucune de ces réponses car les threads ne pourront pas communiquer en utilisant les appels système « *read* » et « *write* ».

QCM 2 – Communication interprocessus

9) On veut faire communiquer deux threads utilisateur d'un même processus via un tube nommé. À quel niveau doit-on ouvrir le tube nommé ? Le tube nommé doit être ouvert :

a) avant la création du premier thread.
b) dans le premier thread.
c) dans le second thread.
d) dans chacun des deux threads (une en lecture et une autre en écriture).
e) aucune de ces réponses car les threads ne pourront pas ouvrir le tube.

10) Si un thread d'un processus exécute cette instruction « dup2(fd=open("*fich1*",O_WRONLY),1); » alors

a) la sortie standard du processus devient le fichier *fich1.*
b) l'écriture dans *fich1* par un thread du processus est redirigée vers l'écran.
c) l'écriture dans *fich1* par un thread du processus peut être réalisé en utilisant le descripteur fd ou 1.
d) Tous les threads du processus n'ont plus accès à l'écran via le descripteur 1.
e) aucune de ces réponses.

Exercice 1

Considérez le programme suivant qui a en entrée trois paramètres : deux fichiers exécutables et un nom de fichier. Ce programme crée deux processus pour exécuter les deux fichiers exécutables.

Complétez le code de manière à exécuter, l'un après l'autre, les deux fichiers exécutables et à rediriger les sorties standards des deux exécutables vers le fichier spécifié comme troisième paramètre. On récupérera ainsi dans ce fichier les résultats du premier exécutable suivis de ceux du deuxième.

```
int main(int argc, char* argv[])
{      /*0*/ int fd = open(argv[3], O_WRONLY);
       if (fork()==0)
       {     /*1*/  dup2(fd,1); close(fd); argv[2]=NULL;
             execvp(argv[1], &argv[1]);
             /*2*/ exit(1);
       }
       /*3*/ wait(NULL);
       if (fork()==0)
       {     /*4*/ dup2(fd,1); close(fd); argv[3]=NULL;
             execvp(argv[2], &argv[2]);
             /*5*/ exit(1);
       }
       /*6*/  close(fd); wait(NULL);
      exit(0);
}
```

Exercice 2

Le signal SIGCHLD est un signal qui est automatiquement envoyé par le fils à son père lorsque le fils se termine (par un exit, un return, ou autre). **Ajoutez une fonction** et **le code nécessaire** pour que le père n'attende pas son fils de façon bloquante et que le fils ne devienne pas zombie (pour longtemps).

```
/*0*/ void sig_action(int sig) { wait(NULL): }
int main(int argc, char *argv[])
{           /*1*/  signal(SIGCHLD,sig_action);
            if (fork()==0)
            {           /*2*/  signal(SIGCHLD,SIG_DFL); // non nécessaire
                        for (int i = 0 ; i <10 ; i++) ; //simule un petit calcul
                        /*3*/
                        exit(0) ;
                        /*4*/
            }
            /*5*/
            while(1) ; //Simule un calcul infini
            /*6*/
   exit(0);
}
```