Nom GRANET

Prénom Lucas

Note ..... /20 points 19/20

4 IRC - Année universitaire 2015/2016

4 Janvier 2016

Contrôle SYSTEMES D'EXPLOITATION && PROGRAMMATION CONCURRENTE

PARTIE 1 -TP

Durée: 1 heure

Documents Cours/TP **autorisés**

Calculatrices **non autorisées**

- On supposera dans chaque cas que les **#include** nécessaires à la bonne compilation et exécution des programmes décrits sont correctement installés et appelés.
- En cas d'ambigüité de compréhension d'une question, n'hésitez pas à ajouter toutes remarques supplémentaires permettant d'expliquer votre démarche.

LES TELEPHONES PORTABLES ET AUTRES APPAREILS DE STOCKAGE DE DONNEES NUMERIQUES NE SONT PAS AUTORISES.
Les téléphones portables doivent être éteints pendant toute la durée de l'épreuve et rangés dans les cartables..
Les cartables doivent être fermés et posés au sol.
Les oreilles des candidats doivent être dégagées.

Rappels importants sur la discipline lors des examens

La présence à tous les examens est strictement obligatoire ; tout élève présent à une épreuve doit rendre une copie, même blanche, portant son nom, son prénom et la nature de l'épreuve.
Une absence non justifiée à un examen invalide automatiquement le module concerné.
Toute suspicion sur la régularité et le caractère équitable d'une épreuve est signalée à la direction des études qui pourra décider l'annulation de l'épreuve; tous les élèves concernés par l'épreuve sont alors convoqués à une épreuve de remplacement à une date fixée par le responsable d'année.
Toute fraude ou tentative de fraude est portée à la connaissance de la direction des études qui pourra réunir le Conseil de Discipline.
Les sanctions prises peuvent aller jusqu'à l'exclusion définitive du (des) élève(s) mis en cause.

Exercice 3 [/3 points]

```
void F( ) {
   fork( ) ;
   fork( ) ;
   printf("*") ;
}
int main() {
   F( ) ;
   printf("*\n") ;
   fork() ;
   return 0 ;
}
```

Nombre d'étoile(s) affichées (s) : 8

Nombre total de processus : 8

```
int main() {
   if (fork( ) ) {
      fork( ) ;
   }
   else {
      fork( );
   }
   printf("*");
   return 0;
}
```

Nombre d'étoile(s) affichées (s) : 4

Nombre total de processus : 4

Exercice 4 [/2 points]

considérons les deux programmes suivants dont les noms des exécutables sont **p1** et **p2**:

```
int main( ) {              // executable p1
   printf ("1\n");
   return 0;
}
```

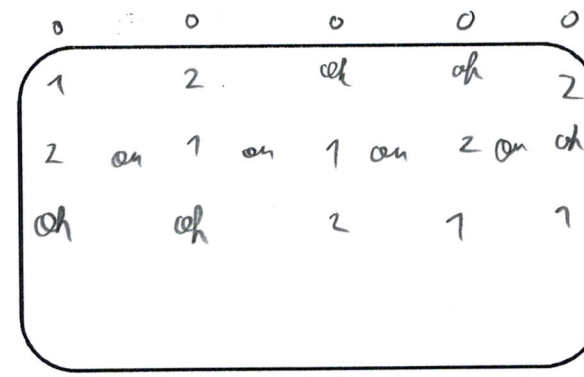
```
int main( ) {              // executable p2
   printf ("2\n");
   return 0;
}
```

On suppose que les 3 exécutables p0, p1 et p2 sont stockés dans le répertoire courant.

L'exécution du programme suivant :

```
int main( ) {              // executable p0
   printf("0\n") ;
   if (fork() == 0) {
      execlp("./p1", "p1", NULL) ;
      printf("Ok\n") ;
   }
   if (fork() == 0) {
      execlp("./p2", "p2", NULL) ;
      printf("Ok\n") ;
   }
   printf("Ok\n") ;
   return 0 ;
}
```

affiche :



Exercice 5 [/2,5 points]

```
void * f(void *arg) {
    int k ;
    int v = * ( (int *)arg );
    for (k = v ; k <v+10 ; k++) {
        printf("%d\n", k);
    }
    // unlock
    pthread_exit(NULL);
}
// Création des Pthread_Mutex
int main() {

   pthread_t th[5] ;

   int i , ind[5];

   for (i=0 ; i<5 ; i++) {

       ind[i] = i*10;
       // lock
       pthread_create(&th[i] , NULL , f , &ind[i]) ;

   }

   for (i=0 ; i<5 ; i++) {

       pthread_join(th[i] , NULL) ;

   }

   return 0;

}
```

L'exécution de ce programme permet de créer 5 threads, le premier thread affiche les entiers de **0** à **9**, le deuxième de **10** à **19**, ... etc. Que faut-il ajouter à ce programme pour que l'affichage des nombres se produise dans l'ordre de **0** à **49**.

On ajoute un mutex_lock avant la création du thread.

On ajoute également un mutex_unlock avant le pthread_exit() de la fonction des Thread.

On utilisera les mutex de la bibliothèque Pthread.

Exercice 6 [/3 points]

```
void * f( void *arg ) {
    int id = *( (int*)arg );
    printf("%d : Ok\n", id);
    pthread_exit (NULL);
}
int main ( ) {
    pthread_t th[10];
    int k , id[10] ;
    for (k=0 ; k<10 ; k++) {
        id[k] = k;
        printf("Thread %d \n", id[k]);
        pthread_create (&th[k] ,  NULL , f , (void *)&id[k]);
    }
    return 0;
}
```

À l'exécution de ce programme, on obtient les affichages suivants :

Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
Thread 8
Thread 9
0 : Ok
5 : Ok
6 : Ok
4 : Ok
3 : Ok
7 : Ok
8 : Ok
2 : Ok
9 : Ok
1 : Ok

On remplace **&id[k]** par **&k** dans l'appel à **pthread_create()**, est-ce qu'on obtient les mêmes affichages. Justifier votre réponse.

Non, nous obtenons pas le même affichage même si l'changement de variable n'a aucun effet.
En effet l'ordre dépend de l'ordonnancement du système, par exemple "Thread 0" peut avoir fini avant que "Thread 1" ne soit créé.

À l'exécution de ce programme, peut-on obtenir les affichages suivants ?

Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
Thread 8
Thread 9

☒ oui ☐ non

Si **oui**, proposez une solution permettant d'éviter ce cas.

Oui, dans le seul et unique cas où la création des threads ont échoué*. On ne peut pas corriger ce problème sauf en effectuant un reboot du système.

* En effet, il manque la trace des threads dans cet affichage c'est que les threads n'ont pu être créés.

Exercice 7 [3 /3 points]

```
void *f(void * arg) {
  char *f = (char *)arg;
  int *r = (int *)malloc(sizeof(int));
  FILE *df;
  char buffer[1024];
  int k , n;
  *r = 0;
  df=fopen(f , "r");
  while ( (n=fread(buffer , 1 , 1024 , df)) !=0 ) {
    for(k =0 ; k <n ; k++ ) {
      if (buffer[k] == '\n') *r = *r + 1;
    }
  }
  fclose(df);
  printf("%s - %d\n",f,*r);
  pthread_exit( (void *)r );
}
```

```
int main(int argc,char **argv) {
  pthread_t th[20];
  int k , S=0;
  int *N;
  for(k=1 ; k<argc ; k++ ) {
    pthread_create(&(th[k]) , NULL , f , argv[k]);
  }
  for (k=1 ; k<argc ; k++ ) {
    pthread_join(th[k] , &N);
    S = S+ *N;
  }
  printf("S = %d\n", S);
  return 0 ;
}
```

Décrire la fonctionnalité réalisée par ce programme

Ce programme permet de compter le nombre de retour à la ligne d'une liste de fichiers passés par argument au programme (max. 20 fichiers)

Il affiche également le nombre de retour à la ligne pour chaque fichier sous la forme "Nom-fichier - < nb-retour >"

Exercice 8 [3 /3 points]

```
sem_t s1 , s2;
void *fonc_th1(void *arg) {
        sem_wait(&s1) ;
        printf("th1\n") ;
        sem_post(&s2) ;
        pthread_exit(NULL);
}
void *fonc_th2(void *arg) {
        sem_wait(&s2) ;
        printf("th2\n");
        sem_post(&s1) ;
        pthread_exit(NULL);
}
int main() {
        pthread_t th1,th2;
        sem_init(&s1 , 0 , 0);
        sem_init(&s2 , 0 , 0);
        pthread_create(&th1 , NULL , fonc_th1 , NULL);
        pthread_create(&th2 , NULL , fonc_th2 , NULL);
        sleep(2);
        sem_post(&s2);
        printf("main()\n");
        pthread_join(th1 , NULL);
        pthread_join(th2 , NULL);
        return 0;
}
```

Ce programme affiche :

- ☐ main() th1 th2
- ☐ th1 th2
- ☒ main() th2 th1
- ☐ th2 th1
- ☐ main() th2
- ☐ main() th1
- ☐ th1
- ☐ th2
- ☐ Rien