L'exclusion mutuelle

Implémentation des sémaphores par fichiers

Exercice 1. Expliquer ce qu'il se passe dans les fichiers `semaphores.h` et `semaphores.c` :

```
 1  #ifndef SEMAPHORES_H
 2  #define SEMAPHORES_H

 4  #include <unistd.h>
 5  #include <fcntl.h>
 6  #include <stdio.h>
 7  #include <string.h>
 8  #include <errno.h>

10  #define LOCKDIR "/tmp/"
11  #define MAXTRIES  3
12  #define NAPTIME 1
13  #define TAILLE_CHEMIN 1

15  extern int lock(char *name);
16  extern void unlock(char *name);

18  #endif /* SEMAPHORES_H */
```

```
 1  #include "semaphores.h"

 3  static char *lockpath(char *);

 5  int lock(char *name)
 6  {
 7    char *path=lockpath(name);
 8    int fd,essays=0;
 9    extern int errno;

11    while( (fd = creat(path,0)) == -1 && errno == EACCES )
12      {
13        if( ++essays >= MAXTRIES)
14          return 0;
15        sleep(NAPTIME);
16      }
17    if( fd == -1 || close(fd) == -1 )
18      perror("lock");
19    return(1);
20  }

22  void unlock(char *name)
23  {
24    if(unlink(lockpath(name)) ==  -1)
25      perror("unlock");
26  }

28  static char *lockpath(char *name)
29  {
30    static char path[TAILLE_CHEMIN];

32    strcpy(path,LOCKDIR);
33    return(strcat(path,name));
34  }
```

Le producteur/consommateur

Le code suivant (incomplet) présente la situation où un producteur et un consommateur partagent des donnés dans un fichier nommés `compteur` :

```
1  int compteur=0;

3  void producteur(int cons_pid,char *nomfic)
4  {
5    int pid=getpid();
6    srand(pid*time(NULL));
7    while(1){
8      sleep(rand()%MAXDELAI);
9      compteur= mon_read(nomfic);
10     if(compteur == NB_CASES)
11       mon_sleep();
12     else
13       {
14         compteur++;
15         mon_write(nomfic,&compteur);
16       }

18     if(compteur == 1)
19       mon_wakeup(cons_pid);
20     printf("[%d] Nombre d'objets produits : %d.\n",
21            pid,compteur);
22   }
23 }

25 void consommateur(int prod_pid,char *nomfic)
26 {
27   int pid=getpid();
28   srand(pid*time(NULL));
29   while(1){
30     sleep(rand()%MAXDELAI);
31     compteur = mon_read(nomfic);

33     if(compteur == 0)
34       mon_sleep();
35     else
36       {
37         compteur--;
38         mon_write(nomfic,&compteur);
39       }

41     if(compteur == NB_CASES -1)
42       mon_wakeup(prod_pid);
43     printf("[%d] Nombre d'objets à consommer : %d.\n",
44            pid,compteur);
45   }
46 }

48 int main(void)
49 {
50   pid_t pid;

52   mon_write("compteur",&compteur);
53   switch(pid = fork())
54     {
55     case -1:
56       exit(EXIT_FAILURE);
57     case 0:
58       producteur(getppid(),"compteur");
59     default:
60       consommateur(pid,"compteur");
61     }
62   exit(EXIT_FAILURE);
63 }
```

Exercice 2. À l'aide des primitive que vous connaissez, implémenter les fonctions `mon_read`, `mon_write`. Améliorez cette implémentation de façon qu'il y ait une exclusion mutuelle entre un appel à `mon_read`, `mon_write`.

Exercice 3. À l'aide des primitive que vous connaissez, donnez une implémentation des primitives `mon_sleep`, et `mon_wakeup`. Améliorez cette implémentation en utilisant le bit d'attente. (Suggestion : bloquer un signal choisi et utiliser `sigsuspend` à la place de `wait` ou `waitpid`.)