L'exclusion mutuelle

Implémentation des sémaphores par fichiers

Exercice 1. Expliquer ce qu'il fait le code suivant :

semaphores.h

```
 1 : #ifndef SEMAPHORES_H
 2 : #define SEMAPHORES_H
 3 :
 4 : #include <unistd.h>
 5 : #include <fcntl.h>
 6 : #include <stdio.h>
 7 : #include <string.h>
 8 : #include <errno.h>
 9 :
10 : #define LOCKDIR "/tmp/"
11 : #define MAXTRIES  3
12 : #define NAPTIME 1
13 : #define TAILLE_CHEMIN 1
14 :
15 : extern int lock(char *name);
16 : extern void unlock(char *name);
17 :
18 : #endif /* SEMAPHORES_H */
```

semaphores.c

```
 1 : #include "semaphores.h"
 2 :
 3 : static char *lockpath(char *);
 4 :
 5 : int lock(char *name)
 6 : {
 7 :     char *path = lockpath(name);
 8 :     int fd, essays = 0;
 9 :     extern int errno;
10 :
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 : }
21 :
22 : void unlock(char *name)
23 : {
24 :     if (unlink(lockpath(name)) == -1)
25 :         perror("unlock");
26 : }
27 :
28 : static char *lockpath(char *name)
29 : {
30 :     static char path[TAILLE_CHEMIN];
31 :
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` :

prodcons.c

```
 1 : void producteur(int cons_pid, char *nomfic)
 2 : {
 3 :     int pid = getpid();
 4 :     srand(pid * time(NULL));
 5 :     while (1)
 6 :     {
 7 :         sleep(rand() % MAXDELAI);
 8 :         compteur = mon_read(nomfic);
 9 :         if (compteur == NB_CASES)
10 :             mon_sleep();
11 :         else
12 :         {
13 :             compteur++;
14 :             mon_write(nomfic, &compteur);
15 :         }
16 :
17 :         if (compteur == 1)
18 :             mon_wakeup(cons_pid);
19 :         printf("[%d] Nombre d'objets produits : %d.\n", pid, compteur);
20 :     }
21 : }
22 :
23 : void consommateur(int prod_pid, char *nomfic)
24 : {
25 :     int pid = getpid();
26 :     srand(pid * time(NULL));
27 :     while (1)
28 :     {
29 :         sleep(rand() % MAXDELAI);
30 :         compteur = mon_read(nomfic);
31 :
32 :         if (compteur == 0)
33 :             mon_sleep();
34 :         else
35 :         {
36 :             compteur--;
37 :             mon_write(nomfic, &compteur);
38 :         }
39 :
40 :         if (compteur == NB_CASES - 1)
41 :             mon_wakeup(prod_pid);
42 :         printf("[%d] Nombre d'objets à consommer : %d.\n", pid, compteur);
43 :     }
44 : }
45 :
46 : int main(void)
47 : {
48 :     pid_t pid;
49 :
50 :     mon_write("compteur", &compteur);
51 :     switch (pid = fork())
52 :     {
53 :     case -1:
54 :         exit(EXIT_FAILURE);
55 :     case 0:
56 :         producteur(getppid(), "compteur");
57 :     default:
58 :         consommateur(pid, "compteur");
59 :     }
60 :     exit(EXIT_FAILURE);
61 : }
```

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`.)