

Le diner des cinq philosophes

Janvier 2009

Le programme ci-dessous est une implantation du problème du *diner des cinq philosophes*. Il utilise la règle proposée par E. Dijkstra dans son article “Hierarchical ordering of sequential processes”, Acta Informatica, vol. 1 (1971). La solution de Dijkstra est fondée sur une règle simple : quand un philosophe termine son repas, il libère les ressources à ses voisins. L’objectif de l’exercice est de répondre aux questions en s’appuyant sur le code source et la sortie d’une exécution.

1 Questions

1. La mémoire virtuelle d’un processus comprend plusieurs zones. Lesquelles ?
2. Pour chacune des zones, vous donnerez un exemple de symbole (variable, fonctions. . .) du programme qui s’y rapporte.
3. Le programme proposé lance plusieurs processus fils : combien ?
4. Quel est le `pid` du processus père ?
5. Expliquer le rôle du gestionnaire `sigusr`.
6. Quelle est la taille de la zone mémoire partagée par les processus.
7. Commenter cette valeur.
8. Commenter les nombres de fautes de page mineures des processus.
9. Commenter les nombres de fautes de page majeures des processus.
10. Combien de sémaphores sont utilisés ?
11. Lequel de ces sémaphores est utilisé pour protéger une section critique ?
12. Commenter sommairement les appels système. Utiliser une ligne au plus pour décrire un appel système, par exemple, `ftok` : création d’une clé IPC.
13. Ecrire l’algorithme de Dijkstra utilisé.

2 Un exemple d'exécution

```

    PID  PPID  CMD
6929  2204  ./philos.exe 1
    MINFL -
    220 -
004eb000-004ec000 r-xp 004eb000 00:00 0 [vdso]
08048000-0804a000 r-xp 00000000 fd:00 622926 /
0804a000-0804b000 rwxp 00001000 fd:00 622926 /
08208000-08229000 rwxp 08208000 00:00 0
42ddb000-42df4000 r-xp 00000000 fd:00 1345985 /lib/ld-2.5.so
42df4000-42df5000 r-xp 00018000 fd:00 1345985 /lib/ld-2.5.so
42df5000-42df6000 rwxp 00019000 fd:00 1345985 /lib/ld-2.5.so
437aa000-438e1000 r-xp 00000000 fd:00 1345986 /lib/libc-2.5.so
438e1000-438e3000 r-xp 00137000 fd:00 1345986 /lib/libc-2.5.so
438e3000-438e4000 rwxp 00139000 fd:00 1345986 /lib/libc-2.5.so
438e4000-438e7000 rwxp 438e4000 00:00 0
b7f35000-b7f37000 rw-p b7f35000 00:00 0
b7f42000-b7f43000 rw-p b7f42000 00:00 0
b7f43000-b7f44000 rw-s 00000000 00:08 2392076 /SYSV0000814e (deleted)
bfa13000-bfa28000 rw-p bfa13000 00:00 0 [stack]

```

————— quelques adresses —————

```

ptr=0xbfa271e0
*ptr=0xb7f43000
cmd=0x804a1d8
*cmd=0x8208008

```

```

    PID  PPID  PGRP  MINFL  MAJFL  START  CMD
6932  6929  6929    58      0 11:31 ./philos.exe 1
    PID  PPID  PGRP  MINFL  MAJFL  START  CMD
6931  6929  6929    58      0 11:31 ./philos.exe 1
    PID  PPID  PGRP  MINFL  MAJFL  START  CMD
6934  6929  6929    58      0 11:31 ./philos.exe 1
    PID  PPID  PGRP  MINFL  MAJFL  START  CMD
6933  6929  6929    58      0 11:31 ./philos.exe 1
    PID  PPID  PGRP  MINFL  MAJFL  START  CMD
6930  6929  6929    58      0 11:31 ./philos.exe 1

```

3 code

```

1 #define NB 5
2 char *cmd;
3 typedef unsigned short int ushort;
4 typedef struct {
5     int etat[ NB ];
6 } sdata;
7 typedef union semun {

```

```

8     int val;
9     struct semid_ds *bfr;
10    ushort * table;
11 } semun_t;
12
13 void sigusr(int num)
14 { sprintf( cmd, "ps -o pid,ppid,pgrp,minflt,majflt,start_time,cmd %d", getpid() );
15   system(cmd);
16   exit(0);
17 }
18
19 int P(int idt, int num)
20 { struct sembuf bfr;
21   bfr . sem_num = num; bfr . sem_op = -1; bfr . sem_flg = 0;
22   return semop(idt, & bfr, 1 );
23 }
24
25 int V(int idt, int num)
26 { struct sembuf bfr;
27   bfr . sem_num = num; bfr . sem_op = +1; bfr . sem_flg = 0;
28   return semop(idt, & bfr, 1 );
29 }
30 void test( int sem, int w, sdata *ptr)
31 {
32   if ( ptr->etat[ (w+1) % NB] == 2 ) return;
33   if ( ptr->etat[ (w+4) % NB] == 2 ) return;
34   ptr->etat[ w ] = 2;
35   V( sem, w );
36 }
37
38 int main(int argc, char* argv[] )
39 { sdata *ptr;
40   key_t key;
41   int i, duree, sem, shm;
42   ushort table[ NB + 1];
43   semun_t u_semun;
44   pid_t pid;
45   duree = atoi( argv[1] ) * 10;
46   cmd = (char*) malloc(1024);
47   if ( ( key = ftok( argv[0], 0 ) ) < 0 )
48     perror("ftok"), exit( 1 );
49
50   if ( ( shm =shmget( key, sizeof(sdata), IPC_CREAT | 0600 ) ) < 0 )
51     perror("shmget"), exit( 1 );
52
53   if ( ( ptr = shmat( shm, NULL, 0 )) == NULL )
54     perror("shmat"), exit ( 1 );
55
56   if ( ( sem = semget( key, NB + 1, IPC_CREAT | 0600 ) ) < 0 )
57     perror( "semget"), exit( 1 );

```

```

58
59     for( i = 0; i < NB ; i++) table[i] = 0;
60     table[ NB ] = 1;
61
62     u_semum . table = table;
63     if ( semctl( sem, 0, SETALL, u_semum) < 0 )
64         perror("semctl"), exit( 1 );
65
66     signal(SIGUSR1, sigusr);
67
68     for( i = 0; i < NB; i++){
69         if ( (pid = fork() ) < 0 )
70             perror("fork"), exit(1);
71         if ( ! pid )
72             while ( 1 ) {
73                 usleep( random() % 10000 );
74
75                 P( sem, 5 );
76                 ptr->etat[i] = 1;
77                 test( sem, i , ptr);
78                 V( sem, 5 );
79                 P( sem, i );
80
81                 P( sem, 5 );
82                 ptr->etat[i] = 0;
83                 test ( sem, ( i + 1 ) % NB , ptr);
84                 test ( sem, ( i + 4 ) % NB , ptr);
85                 V( sem, 5 );
86             }
87     }
88     signal( SIGUSR1, SIG_IGN );
89     sleep( duree );
90     sprintf( cmd, "ps -o pid,ppid,cmd,min_flt,share %d", getpid() );
91     printf("\n");
92     system( cmd );
93     sprintf( cmd, "cat /proc/%d/maps", getpid() );
94     system( cmd );
95     printf("\n-----quelques adresses -----\\n");
96     printf("\n ptr=%p", &ptr);
97     printf("\n*ptr=%p", ptr);
98     printf("\n cmd=%p", &cmd);
99     printf("\n*cmd=%p", cmd);
100    printf("\n\\n-----\\n");
101    fflush( stdout );
102    kill( - getpid(), SIGUSR1);
103    while ( wait(NULL) > 0 ) ;
104    shmctl( shm, IPC_RMID, NULL );
105    semctl( sem, IPC_RMID, 0 );
106    printf("\n");
107    return 0;

```

108 }