Projection mémoire et sémaphores

Novembre 2008

Le partage de mémoire hérité lors d'une *projection mémoire*. Les conflits lors de l'utilisation de la mémoire partagée par des processus concurents et le moyen d'y remédier par l'utilisation d'un *sémaphore*.

1 Projection mémoire

L'exécution des commandes de la cible `demo1` est illustrée par la figure (1). Le résultat surprenant s'explique par l'analyse du code machine du corps de la boucle (2), lieu de prémption le plus probable des processus qui fait bien apparaître des opérations non atomiques. On peut se convaincre davantage en utilisant `gdb` pour attraper (6) les processus en cours d'exécution à la sortie du premier fils. Le programme est modifié (**??**) pour envoyer le signal SIGSTOP vers tous les processus à la sortie de la boucle.

```
1
2  #define MAX 256
3  int    NBFILS = 1, LOG = 4;
4
5  typedef struct {
6    long long  cpt;
7    unsigned long long  sum;
8  } sdata;
9
10 void checkargs(int argc, char* argv[] )
11 { int opt;
12   char *optlist ="l:f:";
13   while ( ( opt = getopt( argc, argv , optlist) ) > 0  )
14           switch( opt ){
15           case 'l': LOG    = atoi( optarg ); break;
16           case 'f': NBFILS = atoi( optarg ); break;
17           default : printf("\nusage %s : %s", argv[0], optlist);
18                     exit ( 1 );
19           }
20 }
21
22 int main(int argc, char* argv[] )
23 { char * fname;
```

```
24    int       fd;
25    sdata  *ptr, sd;
26    pid_t    pid = 0;
27    long  long qte =  0, N;
28    checkargs( argc, argv );
29
30    if ( NULL  == ( fname = tmpnam( NULL ) ) )
31         perror("tmpnam"), exit( 1 );
32
33    if ( ( fd = open( fname, O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR ) ) < 0 )
34      perror("open"), exit ( 1 );
35
36    sd . sum = 0;
37    sd .  cpt = 1LU << LOG;
38
39    if ( write( fd, &sd, sizeof( sdata) ) < 0 )
40      perror( "write" ), exit( 1 );
41
42    ptr = mmap( NULL, sizeof(sdata), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0 );
43    close( fd );
44    unlink( fname );
45    N = ptr->cpt;
46    printf("\nExpected = %Ld", (( N + 1 ) * N) / 2 );
47    fflush(stdout);
48    while ( NBFILS--) {
49      if ( ( pid = fork() ) < 0 )
50        perror("fork"), exit (1);
51      if ( ! pid ){
52        while ( ptr->cpt > 0 ) {
53          ptr -> sum += ptr -> cpt;
54          ptr -> cpt--;
55          qte++;
56        }
57        printf("\n%8d : %8Ld (%5.2f)", getpid(), qte, (100. * qte) / N);
58        fflush(stdout);
59        exit(0);
60      }
61    }
62
63    while ( wait( NULL ) > 0 ) ;
64
65    printf("\nSomme     = %Ld\n", ptr -> sum);
66    fflush( stdout );
67    return 0;
68  }
```

```
./proj.exe -f0 -l20

Expected = 549756338176
Somme    = 0
./proj.exe -f1 -l20

Expected = 549756338176
   27476 :  1048576 (100.00)
Somme    = 549756338176
./proj.exe -f2 -l20

Expected = 549756338176
   27478 :   916382 (87.39)
   27479 :   132195 (12.61)
Somme    = 549756338176
./proj.exe -f3 -l20

Expected = 549756338176
   27481 :  1048576 (100.00)
   27482 :   900240 (85.85)
   27483 :   822300 (78.42)
Somme    = 1293061873246
```

Figure 1: proj.out

2 Sémaphore

L'exécution des commandes de la cible `demo2` est illustrée par la figure (3). Une zone de 26 octets est partagée par deux processus. La commande `ipcs` est utilisée pour tracer la mémoire partagée.

```
1  #define MAX 256
2  int    NBFILS = 1, LOG = 4;
3
4  typedef unsigned short int ushort;
5
6  typedef struct {
7    long long  cpt;
8    unsigned long long  sum;
9  } sdata;
10
11 void checkargs(int argc, char* argv[] )
12 { int opt;
13   char *optlist ="l:f:";
14   while ( ( opt = getopt( argc, argv , optlist) ) > 0  )
15           switch( opt ){
16           case 'l': LOG    = atoi( optarg ); break;
17           case 'f': NBFILS = atoi( optarg ); break;
```

```
.L23:
        movl    -40(%ebp), %eax ; charge ptr dans eax
        movl    8(%eax), %ecx   ; charge sum dans
        movl    12(%eax), %ebx  ;             ebx:ecx
        movl    -40(%ebp), %eax ; charge ptr dans eax
        movl    4(%eax), %edx   ; charge cpt
        movl    (%eax), %eax    ;             edx:eax
        addl    %ecx, %eax      ; edx:eax += ebx:ecx
        adcl    %ebx, %edx      ;
        movl    -40(%ebp), %ecx ; charge ptr dans ecx
        movl    %eax, 8(%ecx)   ; maj de sum
        movl    %edx, 12(%ecx)  ; ptr->sum = edx:eax
        movl    -40(%ebp), %eax ; charge ptr dans eax
        movl    4(%eax), %edx   ; charge cpt
        movl    (%eax), %eax    ;              edx:eax
        addl    $-1, %eax       ; decremente
        adcl    $-1, %edx       ;              edx:eax
        movl    -40(%ebp), %ecx ; charge ptr dans eax
        movl    %eax, (%ecx)    ; maj de cpt
        movl    %edx, 4(%ecx)   ; ptr->cpt = edx:ecx
        leal    -32(%ebp), %eax ; chg @qte dans eax
        addl    $1, (%eax)      ; inc
        adcl    $0, 4(%eax)     ;     qte
```

Figure 2: code assembleur du corps de boucle.

```
18              default : printf("\nusage %s : %s", argv[0], optlist);
19                        exit ( 1 );
20              }
21  }
22
23  typedef union semun {
24      int val;
25     struct semid_ds *bfr;
26     ushort * table;
27  } semun_t;
28
29  int P(int idt)
30  { struct sembuf bfr;
31    bfr . sem_num =  0;
32    bfr . sem_op  = -1;
33    bfr . sem_flg =  0;
34    return semop(idt, & bfr, 1 );
35  }
36
37  int V(int idt)
38  { struct sembuf bfr;
39    bfr . sem_num =  0;
```

```
40    bfr . sem_op  = +1;
41    bfr . sem_flg =  0;
42    return  semop(idt, & bfr, 1 );
43  }
44
45  int main(int argc, char* argv[] )
46  { char * fname;
47    int       fd;
48    sdata  *ptr, sd;
49    pid_t    pid = 0;
50    long  long qte =  0, N;
51    int ok = 1;
52    key_t  key;
53    int sem;
54    ushort table[1];
55    semun_t u_semun;
56
57    if ( ( key = ftok( argv[0], 0 ) ) < 0  )
58      perror("ftok"), exit( 1 );
59
60    if ( ( sem = semget( key, 1, IPC_CREAT | 0660 ) ) < 0 )
61        perror( "semget"), exit( 1 );
62
63      table[0] = 1;
64      u_semun . table = table;
65      if ( semctl( sem, 0, SETALL, u_semun) < 0 )
66        perror("semctl"), exit( 1 );
67
68    checkargs( argc, argv );
69
70    if ( NULL  == ( fname = tmpnam( NULL ) ) )
71         perror("tmpnam"), exit( 1 );
72
73    if ( ( fd = open( fname, O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR ) ) < 0 )
74      perror("open"), exit ( 1 );
75
76    sd . sum = 0;
77    sd . cpt = 1LU << LOG;
78
79    if ( write( fd, &sd, sizeof( sdata) ) < 0 )
80      perror( "write" ), exit( 1 );
81
82    ptr = mmap( NULL, sizeof(sdata), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0 );
83    close( fd );
84    unlink( fname );
85
86
87    N = ptr->cpt;
88    printf("\nExpected = %Ld", (( N + 1 ) * N) / 2 );
89    fflush(stdout);
```

```
90    while ( NBFILS--) {
91      if ( ( pid = fork() ) < 0 )
92        perror("fork"), exit (1);
93      if ( ! pid ){
94        while ( ok ) {
95          P( sem );
96          if ( ptr->cpt > 0 ) {
97          ptr -> sum += ptr -> cpt;
98          ptr -> cpt--;
99          qte++;
100         } else ok = 0;
101         V( sem );
102       }
103       printf("\n%8d : %8Ld (%5.2f)", getpid(), qte, (100. * qte) / N);
104       fflush(stdout);
105       exit(0);
106     }
107   }
108
109   while ( wait( NULL ) > 0 ) ;
110
111   printf("\nSomme     = %Ld\n", ptr -> sum);
112   fflush( stdout );
113   system("ipcs -s");
114   semctl( sem, IPC_RMID, 0 );
115   return 0;
116 }
```

3 Point de rencontre

L'exécution des commandes de la cible `demo3` est illustrée par la figure (5).

4 Makefile

```
cible = proj.exe sem.exe rdv.exe projstop.exe

all : $(cible)


$(cible):%.exe:%.c
        gcc -Wall -g $< -o $@
        grep -v include $< > $<ut

mklistings: mklistings.lex
        flex mklistings.lex
        gcc -Wall lex.yy.c -omklistings -lfl
```

```
sem.pdf: listings.tex sem.tex
	pdflatex sem.tex
	pdflatex sem.tex

semex.pdf: semex.tex
	pdflatex semex.tex
	pdflatex semex.tex

listings.tex : $(cible)
	./mklistings proj.c     > listings.tex
	./mklistings  sem.c    >> listings.tex
	./mklistings  rdv.c    >> listings.tex
	./mklistings projstop.c >> listings.tex

demo1:
	 ./proj.exe -f0 -l20
	 ./proj.exe -f1 -l20
	 ./proj.exe -f2 -l20
	 ./proj.exe -f3 -l20
demo2:
	./sem.exe -f1 -l20
	./sem.exe -f2 -l20
	./sem.exe -f3 -l20

demo3:
	./rdv.exe -f5

demo4:
	./projstop.exe -f3 -l20
output:
	make demo1 | grep -v make > proj.out
	make demo2 | grep -v make >  sem.out
	make demo3 | grep -v make >  rdv.out
```

```
./sem.exe -f1 -l20

Expected = 549756338176
   27488 :  1048576 (100.00)
Somme    = 549756338176

------ Tableaux de s maphores --------
cl         semid      propri taire perms      nsems
0x4d257b10 131075     langevin  600        1
0x25c875da 163844     langevin  600        1
0x00007ab7 2195461    langevin  660        1
0x00000000 2359302    langevin  660        1

./sem.exe -f2 -l20

Expected = 549756338176
   27493 :   552028 (52.65)
   27494 :   496548 (47.35)
Somme    = 549756338176

------ Tableaux de s maphores --------
cl         semid      propri taire perms      nsems
0x4d257b10 131075     langevin  600        1
0x25c875da 163844     langevin  600        1
0x00007ab7 2195461    langevin  660        1
0x00000000 2392070    langevin  660        1

./sem.exe -f3 -l20

Expected = 549756338176
   27500 :   416802 (39.75)
   27499 :   448559 (42.78)
   27501 :   183215 (17.47)
Somme    = 549756338176

------ Tableaux de s maphores --------
cl         semid      propri taire perms      nsems
0x4d257b10 131075     langevin  600        1
0x25c875da 163844     langevin  600        1
0x00007ab7 2195461    langevin  660        1
0x00000000 2424838    langevin  660        1
```

Figure 3: sem.out

```
int main(int argc, char* argv[] )
{
  pid_t    pid = 0;
  key_t  key;
  int sem;
  ushort table[ 1 ];
  semun_t u_semun;
  int duree;
  time_t t;
  checkargs( argc, argv );

  if ( ( key = ftok( argv[0], 0 ) ) < 0  )
    perror("ftok"), exit( 1 );

  if ( ( sem = semget( key, 1, IPC_CREAT  | 0660 ) ) < 0 )
      perror( "semget"), exit( 1 );

  table[ 0 ] = NBFILS;

  u_semun . table = table;

  if ( semctl( sem, 0, SETALL, u_semun) < 0 )
      perror("semctl"), exit( 1 );

  while ( NBFILS--) {
    if ( ( pid = fork() ) < 0 )
      perror("fork"), exit (1);
    if ( ! pid ){
      t      = time( NULL );
      srandom( t * getpid() );
      duree = random() % 60;
      printf("\n%s --> %d s'endort %d sec...", ctime( &t ), getpid(), duree);
      fflush(stdout);
      sleep( duree );
      P( sem );  Z( sem );   V( sem );
      t      = time( NULL );
      printf("\n%d reveil ! %s", getpid(), ctime( &t ) );
      fflush(stdout);
      exit( 0 );
    }
  }

  while ( wait( NULL ) > 0 ) ;

  semctl( sem, IPC_RMID, 0 );
  return 0;
}
```

Figure 4: rdv.out

```
./rdv.exe -f5

Fri Nov 14 09:35:13 2008
 --> 27509 s'endort 16 sec...
Fri Nov 14 09:35:13 2008
 --> 27510 s'endort 55 sec...
Fri Nov 14 09:35:13 2008
 --> 27511 s'endort 40 sec...
Fri Nov 14 09:35:13 2008
 --> 27512 s'endort 47 sec...
Fri Nov 14 09:35:13 2008
 --> 27513 s'endort 45 sec...
27510 reveil ! Fri Nov 14 09:36:08 2008

27512 reveil ! Fri Nov 14 09:36:08 2008

27513 reveil ! Fri Nov 14 09:36:08 2008

27511 reveil ! Fri Nov 14 09:36:08 2008

27509 reveil ! Fri Nov 14 09:36:08 2008
```

Figure 5: main de rdv.c

```
[langevin@ou812 SEM]$ gdb ./projstop.exe  31850

GNU gdb Red Hat Linux (6.3.0.0-1.21rh) ...

Attaching to program:
/home/langevin/web-docs/cours/SEM/projstop.exe, process 31 850

Reading symbols from /lib/libc.so.6...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2

0x08048a5f in main (argc=3, argv=0xbffcabf4) at projstop.c:62
62            while ( ptr->cpt > 0 ) {

(gdb) x/i $pc
0x8048a5f <main+590>:    mov    0x4(%eax),%ecx

(gdb) display ptr->cpt
1: ptr->cpt = 0

(gdb) b 63
Breakpoint 1 at 0x8048a1c: file projstop.c, line 63.

(gdb) ni

Program received signal SIGCONT, Continued.
0x08048a5f in main (argc=3, argv=0xbffcabf4) at projstop.c:62
62            while ( ptr->cpt > 0 ) {
1: ptr->cpt = 0
(gdb) n

Breakpoint 1, main (argc=3, argv=0xbffcabf4) at projstop.c:63
63              ptr -> sum += ptr -> cpt;
1: ptr->cpt = 0
(gdb) n
64              ptr -> cpt--;
1: ptr->cpt = 0
(gdb) n
65              qte++;
1: ptr->cpt = -1
(gdb) n
62            while ( ptr->cpt > 0 ) {
1: ptr->cpt = -1
```

Figure 6: analyse avec gdb

```
int main(int argc, char* argv[] )
{ char * fname;
  int       fd;
  sdata  *ptr, sd;
  pid_t    pid = 0, gid;
  long  long qte =  0, N;
  checkargs( argc, argv );
  setpgid( 0, 0 );
  gid = getpgid( 0 );
  if ( NULL  == ( fname = tmpnam( NULL ) ) )
       perror("tmpnam"), exit( 1 );

  if ( ( fd = open( fname, O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR ) ) < 0 )
    perror("open"), exit ( 1 );

  sd . sum = 0;
  sd .  cpt = 1LU << LOG;

  if ( write( fd, &sd, sizeof( sdata) ) < 0 )
    perror( "write" ), exit( 1 );

  ptr = mmap( NULL, sizeof(sdata), PROT_READ | PROT_WRITE, MAP_SHARED, fd, 0 );
  close( fd );
  unlink( fname );
  N = ptr->cpt;
  printf("\nExpected = %Ld", (( N + 1 ) * N) / 2 );
  fflush(stdout);
  while ( NBFILS--) {
    if ( ( pid = fork() ) < 0 )
      perror("fork"), exit (1);
    if ( ! pid ){
      while ( ptr->cpt > 0 ) {
        ptr -> sum += ptr -> cpt;
        ptr -> cpt--;
        qte++;
      }
      kill( - getpgid(0), SIGSTOP );
      printf("\n%8d : %8Ld (%5.2f)", getpid(), qte, (100. * qte) / N);
      fflush(stdout);
      exit(0);
    }
  }

  while ( wait( NULL ) > 0 ) ;

  printf("\nSomme     = %Ld\n", ptr -> sum);
  fflush( stdout );
  return 0;
}
```

Figure 7: main de proj.c modifié.