Computer Operating Systems, Practice Session 5

Semaphore Operations in Unix

Mustafa Ersen (ersenm@itu.edu.tr)

Istanbul Technical University
34469 Maslak, İstanbul

18 March 2015

Today

Computer Operating Systems, PS 5

Semaphore Operations

Signal Mechanism in Linux

Examples

Semaphore Creation

- Header files in Unix to be used in semaphore operations:
  - sys/ipc.h
  - sys/sem.h
  - sys/types.h

- Semaphore Creation:

```
int semget(key_t key, int nsems, int semflg);
```

If successful, a nonnegative integer is returned as the semaphore set identifier, otherwise -1 is returned, with errno of the error.

`semflg:` `IPC_CREAT|0700` (Last 9 bits: permission flags)

A semaphore set including `nsems` semaphores is created and associated with `key` if one of the following holds:

- `key` is equal to IPC_PRIVATE
- IPC_CREAT & No semaphore set exists associated with `key` value

Semaphore Operations

- `int semop(int semid, struct sembuf *sops, unsigned nsops);`
  - `semop` operates on semaphores selected from semaphore set associated with `semid`
  - Each of the `nsops` elements, pointed by `sops`, determines operation on a specific semaphore (each element is of type: `sembuf`)
- ```
  struct sembuf{
      unsigned short sem_num; // semaphore number starts with 0
      short sem_op; // semaphore operation
      short sem_flg; // operation flags
  };
  ```
- The operations contained in sops are performed in array order **atomically** (i.e., the operations are performed either as a complete unit, or not at all)
- sem_flg
  - `SEM_UNDO`: Allows individual operations in the array to be automatically undone when the process exits.
  - `IPC_NOWAIT`: (Do not allow to wait) If you can not decrease, give error message and return
- sem_op
  - `== 0 : wait for it to be 0 (Must have read permission)`
  - `!= 0 : value is added to the semaphore value (The process must have alter permission on the semaphore set)`
  ```

Semaphore Control

- Control of the Value

```
int semctl(int semid, int semnum, int cmd, arg);
```

- cmd
  - IPC_RMID : Remove the semaphore set, awakening all processes blocked
    GETVAL : Return the value of `semval` for the corresponding semaphore
    SETVAL : Set the value of `semval` of the corresponding semaphore to `arg.val`
    SETALL : Set `semval` values for all semaphores of the set using `arg.array`
    GETALL : Return all of the `semval` values for all semaphores of the set into `arg.array`

Basic Semaphore Operations

```
1  // increment operation
2  void sem_signal(int semid, int val){
3      struct sembuf semaphore;
4      semaphore.sem_num=0;
5      semaphore.sem_op=val;
6      semaphore.sem_flg=1;    // relative: add sem_op to value
7      semop(semid, &semaphore, 1);
8  }
9
10 // decrement operation
11 void sem_wait(int semid, int val){
12     struct sembuf semaphore;
13     semaphore.sem_num=0;
14     semaphore.sem_op=(-1*val);
15     semaphore.sem_flg=1;    // relative: add sem_op to value
16     semop(semid, &semaphore, 1);
17 }
```

Handling Signals

- Necessary header files for handling signals:
  - signal.h
  - sys/types.h

```
1  // signal-handling function
2  void mysignal(int signum){
3      printf("Received signal with num=%d\n", signum);
4  }
5
6  void mysigset(int num){
7      struct sigaction mysigaction;
8      mysigaction.sa_handler=(void *)mysignal;
9      // using the signal-catching function identified by sa_handler
10     mysigaction.sa_flags=0;
11     // sigaction() system call is used to change the action taken by
12     // a process on receipt of a specific signal(specified with num)
13     sigaction(num,&mysigaction,NULL);
14 }
```

Handling Signals

- Sending a signal (specified with `num=sig`) from a process to another process (with given `pid`):

```
int kill(pid_t pid, int sig);
```

- Waiting for a signal:

```
int pause(void);
```

Example 1

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/types.h>
#include <signal.h>   // sigaction

#define SEMKEY 8
int sem_id;

// increment operation
void sem_signal(int semid, int val){
    struct sembuf semaphore;
    semaphore.sem_num=0;
    semaphore.sem_op=val;
    semaphore.sem_flg=1;    // relative: add sem_op to value
    semop(semid, &semaphore, 1);
}
```

Example 1

```
1  // decrement operation
2  void sem_wait(int semid, int val){
3      struct sembuf semaphore;
4      semaphore.sem_num=0;
5      semaphore.sem_op=(-1*val);
6      semaphore.sem_flg=1;    // relative: add sem_op to value
7      semop(semid, &semaphore, 1);
8  }
9
10 // signal-handling function
11 void mysignal(int signum){
12     printf("Received signal with num=%d\n", signum);
13 }
14
15 void mysigset(int num){
16     struct sigaction mysigaction;
17     mysigaction.sa_handler=(void *)mysignal;
18     // using the signal-catching function identified by sa_handler
19     mysigaction.sa_flags=0;
20     // sigaction() system call is used to change the action taken by
21     // a process on receipt of a specific signal(specified with num)
22     sigaction(num,&mysigaction,NULL);
23 }
```

Example 1

```
1  int main(void){
2      // signal handler with num=12
3      mysigset(12);
4      int f=1, i, children[10];
5      // creating 10 child processes
6      for(i=0; i<10; i++){
7          if (f>0)
8              f=fork();
9          if (f==-1){
10             printf("fork error....\n");
11             exit(1);
12         }
13         if (f==0)
14             break;
15         else
16             children[i]=f; // get pid of each child process
17     }
```

Example 1

```
1   // parent process
2   if(f>0){
3       // creating a semaphore with key=SEMKEY
4       sem_id = semget(SEMKEY, 1, 0700|IPC_CREAT);
5       // setting value of the 0th semaphore of the set identified
6       // with sem_id to 0
7       semctl(sem_id, 0, SETVAL, 0);
8       // waiting for a second
9       sleep(1);
10      // sending the signal 12 to all child processes
11      for (i=0; i<10; i++)
12          kill(children[i], 12);
13      // decrease semaphore value by 10 (i.e., wait for all children
14      // to increase semaphore value)
15      sem_wait(sem_id, 10);
16      printf("ALL CHILDREN HAS Finished ...\n");
17      // remove the semaphore set identified with sem_id
18      semctl(sem_id, 0, IPC_RMID, 0);
19      exit(0);
20  }
```

Example 1

```
    // child process
    else{
      // wait for a signal
      pause();
      // returning the sem_id associated with SEMKEY
      sem_id = semget(SEMKEY, 1, 0);
      printf("I am the CHILD Process created in %dth order. My PROCESS ID: %d\n",
       i, getpid());
      // getting value of the 0th semaphore of the set identified
      // with sem_id
      printf("SEMAPHORE VALUE: %d\n",semctl(sem_id,0,GETVAL,0));
      // increase semaphore value by 1
      sem_signal(sem_id, 1);
    }

    return 0;
}
```

Output of Example 1

```
Received signal with num=12
I am the CHILD Process created in 0th order. My PROCESS ID: 1629
SEMAPHORE VALUE: 0
Received signal with num=12
I am the CHILD Process created in 1th order. My PROCESS ID: 1630
SEMAPHORE VALUE: 1
Received signal with num=12
I am the CHILD Process created in 2th order. My PROCESS ID: 1631
Received signal with num=12
I am the CHILD Process created in 3th order. My PROCESS ID: 1632
SEMAPHORE VALUE: 2
SEMAPHORE VALUE: 3
Received signal with num=12
I am the CHILD Process created in 4th order. My PROCESS ID: 1633
SEMAPHORE VALUE: 3
Received signal with num=12
I am the CHILD Process created in 6th order. My PROCESS ID: 1635
SEMAPHORE VALUE: 5
```

Output of Example 1 (Continues)

```
Received signal with num=12
Received signal with num=12
I am the CHILD Process created in 9th order. My PROCESS ID: 1638
SEMAPHORE VALUE: 6
Received signal with num=12
Received signal with num=12
I am the CHILD Process created in 5th order. My PROCESS ID: 1634
SEMAPHORE VALUE: 7
I am the CHILD Process created in 7th order. My PROCESS ID: 1636
SEMAPHORE VALUE: 8
I am the CHILD Process created in 8th order. My PROCESS ID: 1637
SEMAPHORE VALUE: 9
ALL CHILDREN HAS Finished ...
```

Example 2 - Deadlock

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/types.h>
#include <signal.h>

#define SEMKEY_A 1
#define SEMKEY_B 2
#define SEMKEY_C 3

// increment operation
void sem_signal(int semid, int val){
    struct sembuf semaphore;
    semaphore.sem_num=0;
    semaphore.sem_op=val;
    semaphore.sem_flg=1;    // relative: add sem_op to value
    semop(semid, &semaphore, 1);
}
```

Example 2 - Deadlock

```
1  // decrement operation
2  void sem_wait(int semid, int val){
3      struct sembuf semaphore;
4      semaphore.sem_num=0;
5      semaphore.sem_op=(-1*val);
6      semaphore.sem_flg=1;    // relative: add sem_op to value
7      semop(semid, &semaphore, 1);
8  }
9
10 // signal-handling function
11 void mysignal(int signum){
12     printf("Received signal with num=%d\n", signum);
13 }
14
15 void mysigset(int num){
16     struct sigaction mysigaction;
17     mysigaction.sa_handler=(void *)mysignal;
18     // using the signal-catching function identified by sa_handler
19     mysigaction.sa_flags=0;
20     // sigaction() system call is used to change the action taken by
21     // a process on receipt of a specific signal(specified with num)
22     sigaction(num,&mysigaction,NULL);
23 }
```

Example 2 - Deadlock

```
1  int main(void){
2      // signal handler with num=12
3      mysigset(12);
4      int semA,semB,semC,c[2],f=1,i,myOrder;
5      // creating 2 child processes
6      for(i=0; i<2; i++){
7          if (f>0)
8              f=fork();
9          if (f==-1){
10             printf("fork error....\n");
11             exit(1);
12         }
13         if (f==0)
14             break;
15         else
16             c[i]=f; // get pid of each child process
17     }
```

Example 2 - Deadlock

```
1   // parent process
2   if (f!=0){
3       printf("PARENT is starting to CREATE RESOURCES....\n");
4       // creating 3 semaphores and setting two of them as 1 and the other as 0
5       semA=semget(SEMKEY_A,1,0700|IPC_CREAT);
6       semctl(semA, 0, SETVAL, 1);
7       semB=semget(SEMKEY_B,1,0700|IPC_CREAT);
8       semctl(semB, 0, SETVAL, 1);
9       semC=semget(SEMKEY_C,1,0700|IPC_CREAT);
10      semctl(semC, 0, SETVAL, 0);
11      sleep(2);
12      printf("PARENT is starting CHILD Processes .......\n");
13      // sending the signal 12 to all child processes
14      for (i=0; i<2; i++)
15          kill(c[i],12);
16      // decrease semaphore value by 2 (i.e., wait for all children)
17      sem_wait(semC,2);
18      printf("PARENT: Child processes has done, resources are removed back.\n");
19      // remove the created semaphore sets
20      semctl(semC,0,IPC_RMID,0);
21      semctl(semA,0,IPC_RMID,0);
22      semctl(semB,0,IPC_RMID,0);
23      exit(0);
24  }
```

Example 2 - Deadlock

```
1   // child process
2   else{
3       myOrder=i;
4       printf("CHILD %d: waiting permission from PARENT ....\n", myOrder);
5       // wait for a signal
6       pause();
7       // returning the sem_ids associated with SEMKEY_A, SEMKEY_B and SEMKEY_C
8       semA=semget(SEMKEY_A,1,0);
9       semB=semget(SEMKEY_B,1,0);
10      semC=semget(SEMKEY_C,1,0);
11      printf("CHILD %d has permission from PARENT, is starting ...\n", myOrder);
12      if (myOrder==0){
13          printf("CHILD %d: DECREASING sem A.\n", myOrder);
14          sem_wait(semA, 1);
15          sleep(1);
16          printf("CHILD %d: sem A is completed, DECREASING sem B.\n", myOrder);
17          sem_wait(semB, 1);
18          printf("CHILD %d: I am in the CRITICAL REGION.\n", myOrder);
19          sleep(5); /* Critical Region Operations */
20          // increase all the semaphore values by 1
21          sem_signal(semB, 1);
22          sem_signal(semA, 1);
23          sem_signal(semC, 1);
24      }
```

Example 2 - Deadlock

```
        else if (myOrder==1){
            printf("CHILD %d: DECREASING sem B.\n", myOrder);
            sem_wait(semB, 1);
            sleep(1);
            printf("CHILD %d: sem B is completed, DECREASING sem A.\n", myOrder);
            sem_wait(semA, 1);
            printf("CHILD %d: I am in the CRITICAL REGION.\n", myOrder);
            sleep(5); /* Critical Region Operations */
            // increase all the semaphore values by 1
            sem_signal(semA,1);
            sem_signal(semB,1);
            sem_signal(semC,1);
        }
    }
    return 0;
}
```

Output of Example 2 - Deadlock

```
1  PARENT is starting to CREATE RESOURCES....
2  CHILD 1: waiting permission from PARENT ....
3  CHILD 0: waiting permission from PARENT ....
4  PARENT is starting CHILD Processes .......
5   Received signal with num=12
6  CHILD 1 has permission from PARENT, is starting ...
7  CHILD 1: DECREASING sem B.
8   Received signal with num=12
9  CHILD 0 has permission from PARENT, is starting ...
10 CHILD 0: DECREASING sem A.
11 CHILD 1: sem B is completed, DECREASING sem A.
12 CHILD 0: sem A is completed, DECREASING sem B.
```

Example 3 - Preventing Deadlock

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <unistd.h>
4  #include <sys/wait.h>
5  #include <sys/ipc.h>
6  #include <sys/sem.h>
7  #include <sys/types.h>
8  #include <signal.h>
9  #include <sys/errno.h>
10
11 #define SEMKEY_AB 5
12 #define SEMKEY_C 6
```

Example 3 - Preventing Deadlock

```
1  // increment operation
2  void sem_signal(int semid, int val){
3      struct sembuf semaphore;
4      semaphore.sem_num=0;
5      semaphore.sem_op=val;
6      semaphore.sem_flg=1;    // relative: add sem_op to value
7      semop(semid, &semaphore, 1);
8  }
9
10 // increment operation using two semaphores
11 void sem_multi_signal(int semid, int val, int nsems){
12     struct sembuf semaphore[2];
13     int i;
14     for (i=0; i<nsems; i++){
15         semaphore[i].sem_num=i;
16         semaphore[i].sem_op=val;
17         semaphore[i].sem_flg=1;
18     }
19     // TWO Operations are performed on SAME SEMAPHORE SET
20     semop(semid, semaphore, 2);
21     for (i=0; i<nsems; i++){
22         printf("SIGNAL : SEM %d IS NOW: .... %d\n", i, semctl(semid,i,GETVAL,0));
23     }
24 }
```

Example 3 - Preventing Deadlock

```
1  // decrement operation
2  void sem_wait(int semid, int val){
3      struct sembuf semaphore;
4      semaphore.sem_num=0;
5      semaphore.sem_op=(-1*val);
6      semaphore.sem_flg=1;    // relative: add sem_op to value
7      semop(semid, &semaphore, 1);
8  }
9
10 // decrement operation using two semaphores
11 void sem_multi_wait(int semid, int val, int nsems){
12     struct sembuf semaphore[2];
13     int i;
14     for (i=0; i<nsems; i++){
15         semaphore[i].sem_num=i;
16         semaphore[i].sem_op=(-1*val);
17         semaphore[i].sem_flg=1;
18     }
19     //TWO Operations are performed on SAME SEMAPHORE SET:
20     semop(semid, semaphore, 2);
21     for (i=0; i<nsems; i++){
22         printf("WAIT : SEM %d is NOW .... %d\n", i, semctl(semid,i,GETVAL,0));
23     }
24 }
```

Example 3 - Preventing Deadlock

```
1  // signal-handling function
2  void mysignal(int signum){
3      printf("Received signal with num=%d\n", signum);
4  }
5
6  void mysigset(int num){
7      struct sigaction mysigaction;
8      mysigaction.sa_handler=(void *)mysignal;
9      // using the signal-catching function identified by sa_handler
10     mysigaction.sa_flags=0;
11     // sigaction() system call is used to change the action taken by a
12     // process on receipt of a specific signal (specified with num)
13     sigaction(num,&mysigaction,NULL);
14 }
```

Example 3 - Preventing Deadlock

```
1  int main(void){
2      // signal handler with num=12
3      mysigset(12);
4      int semAB,semC,c[2],f=1,i,myOrder;
5      // creating 2 child processes
6      for(i=0; i<2; i++){
7          if (f>0)
8              f=fork();
9          if (f==-1){
10             printf("fork error....\n");
11             exit(1);
12         }
13         if (f==0)
14             break;
15         else
16             c[i]=f; // get pid of each child process
17     }
```

Example 3 - Preventing Deadlock

```
1   if (f!=0){    // parent process
2       printf("PARENT is starting to CREATE RESOURCES....\n");
3       // creating a set of 2 semaphores and setting their values as 1
4       semAB=semget(SEMKEY_AB, 2, 0700|IPC_CREAT);
5       if(semAB == -1)
6           printf("SEMGET ERROR on SEM SET, Error Code: %d \n", errno);
7       if (semctl(semAB, 0, SETVAL, 1) == -1)
8           printf("SMCTL ERROR on SEM A, Error Code: %d \n", errno);
9       if (semctl(semAB, 1, SETVAL, 1) == -1)
10          printf("SMCTL ERROR on SEM B, Error Code: %d \n", errno);
11      printf("PARENT: SEM A is NOW .... %d\n", semctl(semAB,0,GETVAL,0));
12      printf("PARENT: SEM B is NOW .... %d\n", semctl(semAB,1,GETVAL,0));
13      //creating another semaphore and setting its value as 0
14      semC=semget(SEMKEY_C,1,0700|IPC_CREAT);
15      semctl(semC, 0, SETVAL, 0);
16      printf("PARENT: SEM C is NOW .... %d\n", semctl(semC,0,GETVAL,0));
17      sleep(2);
18      printf("PARENT is starting CHILD Processes .......\n");
19      for (i=0; i<2; i++)
20          kill(c[i],12);
21      sleep(5);
22      sem_wait(semC,2); //decrease sem. value by 2 (i.e., wait for all children)
23      printf("PARENT: SEM C is NOW .... %d\n", semctl(semC,0,GETVAL,0));
24      printf("PARENT: Child processes has done, resources are removed back.\n");
25      semctl(semC,0,IPC_RMID,0);
26      semctl(semAB,0,IPC_RMID,0);
27      exit(0);
28  }
```

Example 3 - Preventing Deadlock

```
1     else{    // child process
2         myOrder=i;
3         printf("CHILD %d: waiting permission from PARENT ....\n", myOrder);
4         // wait for a signal
5         pause();
6         // returning the sem_ids associated with SEMKEY_AB and SEMKEY_C
7         semAB=semget(SEMKEY_AB,2,0);
8         semC=semget(SEMKEY_C,1,0);
9         printf("CHILD %d has permission from PARENT, is starting ....\n",myOrder);
10        printf("CHILD %d: DECREASING sem AB.\n", myOrder);
11        // decrease two semaphores in the set specified by semAB by 1
12        sem_multi_wait(semAB,1,2);
13        printf("CHILD %d: I am in the CRITICAL REGION.\n", myOrder);
14        sleep(5);
15        // increase two semaphores in the set specified by semAB by 1
16        sem_multi_signal(semAB,1,2);
17        // increase the third semaphore by 1
18        sem_signal(semC,1);
19    }
20    return 0;
21 }
```

Output of Example 3 - Preventing Deadlock

```
PARENT is starting to CREATE RESOURCES....
PARENT: SEM A is NOW .... 1
PARENT: SEM B is NOW .... 1
PARENT: SEM C is NOW .... 0
CHILD 1: waiting permission from PARENT ....
CHILD 0: waiting permission from PARENT ....
PARENT is starting CHILD Processes .......
Received signal with num=12
CHILD 1 has permission from PARENT, is starting ....
CHILD 1: DECREASING sem AB.
WAIT : SEM 0 is NOW .... 0
WAIT : SEM 1 is NOW .... 0
CHILD 1: I am in the CRITICAL REGION.
Received signal with num=12
CHILD 0 has permission from PARENT, is starting ....
CHILD 0: DECREASING sem AB.
SIGNAL : SEM 0 IS NOW: .... 0
SIGNAL : SEM 1 IS NOW: .... 0
WAIT : SEM 0 is NOW .... 0
WAIT : SEM 1 is NOW .... 0
CHILD 0: I am in the CRITICAL REGION.
SIGNAL : SEM 0 IS NOW: .... 1
SIGNAL : SEM 1 IS NOW: .... 1
PARENT: SEM C is NOW .... 0
PARENT: Child processes has done, resources are removed back.
```