# Computer Operating Systems, Practice Session 5

## Semaphore Operations in Unix

Mustafa Ersen (ersenm@itu.edu.tr)

Istanbul Technical University
34469 Maslak, İstanbul

12 March 2014


## 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`:
- 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: Increment

```
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);
}
```

# Basic Semaphore Operations: Decrement

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

# Handling Signals

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

```
// signal-handling function
void mysignal(int signum){
    printf("Received signal with num=%d\n", signum);
}

void mysigset(int num){
    struct sigaction mysigaction;
    mysigaction.sa_handler=(void *)mysignal;
    // using the signal-catching function identified by sa_handler
    mysigaction.sa_flags=0;
    // sigaction() system call is used to change the action taken by a
    // process on receipt of a specific signal (specified with num)
    sigaction(num,&mysigaction,NULL);
}
```

# 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

```
22  // decrement operation
23  void sem_wait(int semid, int val){
24      struct sembuf semaphore;
25      semaphore.sem_num=0;
26      semaphore.sem_op=(-1*val);
27      semaphore.sem_flg=1;    // relative: add sem_op to value
28      semop(semid, &semaphore, 1);
29  }
30
31  // signal-handling function
32  void mysignal(int signum){
33      printf("Received signal with num=%d\n", signum);
34  }
35  void mysigset(int num){
36      struct sigaction mysigaction;
37      mysigaction.sa_handler=(void *)mysignal;
38      // using the signal-catching function identified by sa_handler
39      mysigaction.sa_flags=0;
40      // sigaction() system call is used to change the action taken by a
41      // process on receipt of a specific signal (specified with num)
42      sigaction(num,&mysigaction,NULL);
43  }
```

# Example 1

```
45  int main(void){
46      // signal handler with num=12
47      mysigset(12);
48      int f=1, i, children[10];
49      // creating 10 child processes
50      for(i=0; i<10; i++){
51          if (f>0)
52              f=fork();
53          if (f==-1){
54              printf("fork error....\n");
55              exit(1);
56          }
57          if (f==0)
58              break;
59          else
60              children[i]=f; // get pid of each child process
61      }
```

# Example 1

```
62      // parent process
63      if(f>0){
64          // creating a semaphore with key=SEMKEY
65          sem_id = semget(SEMKEY, 1, 0700|IPC_CREAT);
66          // setting value of the 0th semaphore of the set identified with sem_id to 0
67          semctl(sem_id, 0, SETVAL, 0);
68          // waiting for a second
69          sleep(1);
70          // sending the signal 12 to all child processes
71          for (i=0; i<10; i++)
72              kill(children[i], 12);
73          // decrease semaphore value by 10 (i.e., wait for all childs to increase semaphore value)
74          sem_wait(sem_id, 10);
75          printf("ALL CHILDREN HAS Finished ...\n");
76          // remove the semaphore set identified with sem_id
77          semctl(sem_id, 0, IPC_RMID, 0);
78          exit(0);
79      }
```

# Example 1

```
80      // child process
81      else{
82          // wait for a signal
83          pause();
84          // returning the sem_id associated with SEMKEY
85          sem_id = semget(SEMKEY, 1, 0);
86          printf("I am the CHILD Process created in %d th order. My PROCESS ID: %d\n", i, getpid());
87          // getting value of the 0th semaphore of the set identified with sem_id
88          printf("SEMAPHORE VALUE: %d\n",semctl(sem_id,0,GETVAL,0));
89          // increase semaphore value by 1
90          sem_signal(sem_id, 1);
91      }
92
93      return 0;
94  }
```

# Output of Example 1

```
Received signal with num=12
I am the CHILD Process created in 5 th order. My PROCESS ID: 2367
SEMAPHORE VALUE: 0
Received signal with num=12
I am the CHILD Process created in 2 th order. My PROCESS ID: 2364
SEMAPHORE VALUE: 1
Received signal with num=12
I am the CHILD Process created in 3 th order. My PROCESS ID: 2365
SEMAPHORE VALUE: 2
Received signal with num=12
I am the CHILD Process created in 1 th order. My PROCESS ID: 2363
SEMAPHORE VALUE: 3
Received signal with num=12
Received signal with num=12
Received signal with num=12
```

# Output of Example 1 (Continues)

```
I am the CHILD Process created in 0 th order. My PROCESS ID: 2362
I am the CHILD Process created in 8 th order. My PROCESS ID: 2370
SEMAPHORE VALUE: 4
Received signal with num=12
I am the CHILD Process created in 7 th order. My PROCESS ID: 2369
SEMAPHORE VALUE: 4
SEMAPHORE VALUE: 6
I am the CHILD Process created in 9 th order. My PROCESS ID: 2371
SEMAPHORE VALUE: 6
Received signal with num=12
Received signal with num=12
I am the CHILD Process created in 4 th order. My PROCESS ID: 2366
SEMAPHORE VALUE: 8
I am the CHILD Process created in 6 th order. My PROCESS ID: 2368
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

```
23  // decrement operation
24  void sem_wait(int semid, int val){
25      struct sembuf semaphore;
26      semaphore.sem_num=0;
27      semaphore.sem_op=(-1*val);
28      semaphore.sem_flg=1;    // relative: add sem_op to value
29      semop(semid, &semaphore, 1);
30  }
31
32  // signal-handling function
33  void mysignal(int signum){
34      printf("Received signal with num=%d\n", signum);
35  }
36
37  void mysigset(int num){
38      struct sigaction mysigaction;
39      mysigaction.sa_handler=(void *)mysignal;
40      // using the signal-catching function identified by sa_handler
41      mysigaction.sa_flags=0;
42      // sigaction() system call is used to change the action taken by a
43      // process on receipt of a specific signal (specified with num)
44      sigaction(num,&mysigaction,NULL);
45  }
```

# Example 2 - Deadlock

```
47  int main(void){
48      // signal handler with num=12
49      mysigset(12);
50      int semA,semB,semC,c[2],f=1,i,myOrder;
51      // creating 2 child processes
52      for(i=0; i<2; i++){
53          if (f>0)
54              f=fork();
55          if (f==-1){
56              printf("fork error....\n");
57              exit(1);
58          }
59          if (f==0)
60              break;
61          else
62              c[i]=f; // get pid of each child process
63      }
```

# Example 2 - Deadlock

```
64      // parent process
65      if (f!=0){
66          printf("PARENT is starting to CREATE RESOURCES....\n");
67          // creating 3 semaphores and setting two of them as 1 and the other as 0
68          semA=semget(SEMKEY_A,1,0700|IPC_CREAT);
69          semctl(semA, 0, SETVAL, 1);
70          semB=semget(SEMKEY_B,1,0700|IPC_CREAT);
71          semctl(semB, 0, SETVAL, 1);
72          semC=semget(SEMKEY_C,1,0700|IPC_CREAT);
73          semctl(semC, 0, SETVAL, 0);
74          sleep(2);
75          printf("PARENT is starting CHILD Processes .......\n");
76          // sending the signal 12 to all child processes
77          for (i=0; i<2; i++)
78              kill(c[i],12);
79          // decrease semaphore value by 2 (i.e., wait for all children)
80          sem_wait(semC,2);
81          printf("PARENT: Child processes has done, resources are removed back...\n");
82          // remove the created semaphore sets
83          semctl(semC,0,IPC_RMID,0);
84          semctl(semA,0,IPC_RMID,0);
85          semctl(semB,0,IPC_RMID,0);
86          exit(0);
87      }
```

# Example 2 - Deadlock

```
88      // child process
89      else{
90          myOrder=i;
91          printf("CHILD %d: waiting permission from PARENT ....\n", myOrder);
92          // wait for a signal
93          pause();
94          // returning the sem_ids associated with SEMKEY_A, SEMKEY_B and SEMKEY_C
95          semA=semget(SEMKEY_A,1,0);
96          semB=semget(SEMKEY_B,1,0);
97          semC=semget(SEMKEY_C,1,0);
98          printf("CHILD %d has permission from PARENT, is starting ....\n", myOrder);
99          if (myOrder==0){
100             printf("CHILD %d: DECREASING sem A.\n", myOrder);
101             sem_wait(semA, 1);
102             sleep(1);
103             printf("CHILD %d: sem A is completed, DECREASING sem B.\n", myOrder);
104             sem_wait(semB, 1);
105             printf("CHILD %d: I am in the CRITICAL REGION.\n", myOrder);
106             sleep(5); /* Critical Region Operations */
107             // increase all the semaphore values by 1
108             sem_signal(semB, 1);
109             sem_signal(semA, 1);
110             sem_signal(semC, 1);
111         }
```

# Example 2 - Deadlock

```
112         else if (myOrder==1){
113             printf("CHILD %d: DECREASING sem B.\n", myOrder);
114             sem_wait(semB, 1);
115             sleep(1);
116             printf("CHILD %d: sem B is completed, DECREASING sem A.\n", myOrder);
117             sem_wait(semA, 1);
118             printf("CHILD %d: I am in the CRITICAL REGION.\n", myOrder);
119             sleep(5); /* Critical Region Operations */
120             // increase all the semaphore values by 1
121             sem_signal(semA,1);
122             sem_signal(semB,1);
123             sem_signal(semC,1);
124         }
125     }
126     return 0;
127 }
```

# Output of Example 2

```
PARENT is starting to CREATE RESOURCES....
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 B.
Received signal with num=12
CHILD 0 has permission from PARENT, is starting ....
CHILD 0: DECREASING sem A.
CHILD 1: sem B is completed, DECREASING sem A.
CHILD 0: sem A is completed, DECREASING sem B.
```

# Example 3 - Preventing 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>
#include <sys/errno.h>

#define SEMKEY_AB 5
#define SEMKEY_C 6
```

# Example 3 - Preventing Deadlock

```
14  // increment operation
15  void sem_signal(int semid, int val){
16      struct sembuf semaphore;
17      semaphore.sem_num=0;
18      semaphore.sem_op=val;
19      semaphore.sem_flg=1;    // relative: add sem_op to value
20      semop(semid, &semaphore, 1);
21  }
22
23  // increment operation using two semaphores
24  void sem_multi_signal(int semid, int val, int nsems){
25      struct sembuf semaphore[2];
26      int i;
27      for (i=0; i<nsems; i++){
28          semaphore[i].sem_num=i;
29          semaphore[i].sem_op=val;
30          semaphore[i].sem_flg=1;
31      }
32      // TWO Operations are performed on SAME SEMAPHORE SET
33      semop(semid, semaphore, 2);
34      for (i=0; i<nsems; i++){
35          printf("SIGNAL : SEM %d IS NOW: .... %d\n", i, semctl(semid,i,GETVAL,0));
36      }
37  }
```

# Example 3 - Preventing Deadlock

```
39  // decrement operation
40 void sem_wait(int semid, int val){
41      struct sembuf semaphore;
42      semaphore.sem_num=0;
43      semaphore.sem_op=(-1*val);
44      semaphore.sem_flg=1;    // relative: add sem_op to value
45      semop(semid, &semaphore, 1);
46  }
47
48  // decrement operation using two semaphores
49 void sem_multi_wait(int semid, int val, int nsems){
50      struct sembuf semaphore[2];
51      int i;
52      for (i=0; i<nsems; i++){
53          semaphore[i].sem_num=i;
54          semaphore[i].sem_op=(-1*val);
55          semaphore[i].sem_flg=1;
56      }
57      //TWO Operations are performed on SAME SEMAPHORE SET:
58      semop(semid, semaphore, 2);
59      for (i=0; i<nsems; i++){
60          printf("WAIT : SEM %d is NOW .... %d\n", i, semctl(semid,i,GETVAL,0));
61      }
62  }
```

# Example 3 - Preventing Deadlock

```
65  void mysignal(int signum){ printf("Received signal with num=%d\n", signum);}
67  void mysigset(int num){
68      struct sigaction mysigaction;
69      mysigaction.sa_handler=(void *)mysignal;
70      // using the signal-catching function identified by sa_handler
71      mysigaction.sa_flags=0;
72      // sigaction() system call is used to change the action taken by a
73      // process on receipt of a specific signal (specified with num)
74      sigaction(num,&mysigaction,NULL);
75  }
77  int main(void){
78      // signal handler with num=12
79      mysigset(12);
80      int semAB,semC,c[2],f=1,i,myOrder;
81      // creating 2 child processes
82      for(i=0; i<2; i++){
83          if (f>0)
84              f=fork();
85          if (f==-1){
86              printf("fork error....\n");
87              exit(1);
88          }
89          if (f==0)
90              break;
91          else
92              c[i]=f; // get pid of each child process
93      }
```

# Example 3 - Preventing Deadlock

```
96      // parent process
97      if (f!=0){
98          printf("PARENT is starting to CREATE RESOURCES....\n");
99          // creating a set of 2 semaphores and setting their values as 1
100         semAB=semget(SEMKEY_AB, 2, 0700|IPC_CREAT);
101         if(semAB == -1)
102             printf("SEMGET ERROR on SEM SET, Error Code: %d \n", errno);
103         if (semctl(semAB, 0, SETVAL, 1) == -1)
104             printf("SMCTL ERROR on SEM A, Error Code: %d \n", errno);
105         if (semctl(semAB, 1, SETVAL, 1) == -1)
106             printf("SMCTL ERROR on SEM B, Error Code: %d \n", errno);
107         printf("PARENT: SEM A is NOW .... %d\n", semctl(semAB,0,GETVAL,0));
108         printf("PARENT: SEM B is NOW .... %d\n", semctl(semAB,1,GETVAL,0));
109         //creating another semaphore and setting its value as 0
110         semC=semget(SEMKEY_C,1,0700|IPC_CREAT);
111         semctl(semC, 0, SETVAL, 0);
112         printf("PARENT: SEM C is NOW .... %d\n", semctl(semC,0,GETVAL,0));
113         sleep(2);
114         printf("PARENT is starting CHILD Processes .......\n");
115         for (i=0; i<2; i++)
116             kill(c[i],12);
117         sleep(5);
118         // decrease semaphore value by 2 (i.e., wait for all children)
119         sem_wait(semC,2);
120         printf("PARENT: SEM C is NOW .... %d\n", semctl(semC,0,GETVAL,0));
121         printf("PARENT: Child processes has done, resources are removed back...\n");
122         semctl(semC,0,IPC_RMID,0);
123         semctl(semAB,0,IPC_RMID,0);
124         exit(0);
125     }
```

# Example 3 - Preventing Deadlock

```
126     // child process
127     else{
128         myOrder=i;
129         printf("CHILD %d: waiting permission from PARENT ....\n", myOrder);
130         // wait for a signal
131         pause();
132         // returning the sem_ids associated with SEMKEY_AB and SEMKEY_C
133         semAB=semget(SEMKEY_AB,2,0);
134         semC=semget(SEMKEY_C,1,0);
135         printf("CHILD %d has permission from PARENT, is starting ....\n", myOrder);
136         printf("CHILD %d: DECREASING sem AB.\n", myOrder);
137         // decrease two semaphores in the set specified by semAB by 1
138         sem_multi_wait(semAB,1,2);
139         printf("CHILD %d: I am in the CRITICAL REGION.\n", myOrder);
140         sleep(5);
141         // increase two semaphores in the set specified by semAB by 1
142         sem_multi_signal(semAB,1,2);
143         // increase the third semaphore by 1
144         sem_signal(semC,1);
145     }
146     return 0;
147 }
```

# Output of Example 3

```
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...
```