Computer Operating Systems

Practice Session 3: Threads and Synchronization

Esin Ece Aydın Yıldırım (aydinesi16@itu.edu.tr)
Ömür Fatmanur Erzurumluoğlu (erzurumluoglu18@itu.edu.tr)
Uğur Ayvaz(ayvaz18@itu.edu.tr)

May 15, 2024

Today

Operating Systems, PS 3

- Thread Creation and Termination
- Joining Threads
- Using Global Variables in Threads
- Synchronization
- Mutex Usage
- Semaphore Usage
- Signal Mechanism in Linux
- Examples

Thread Creation and Termination

- Thread libraries provide functions for creating threads and other operations.
- To create a thread, `pthread_create()` function can be used which is defined in `<pthread.h>`
- To terminate threads, `pthread_exit()` function defined in the same library can be used.

man pthread_create

```
esinece@esinece-VirtualBox:~$ man pthread_create
```

```
PTHREAD_CREATE(3)              Linux Programmer's Manual              PTHREAD_CREATE(3)

NAME
       pthread_create - create a new thread

SYNOPSIS
       #include <pthread.h>

       int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
                          void *(*start_routine) (void *), void *arg);

       Compile and link with -pthread.

DESCRIPTION
       The  pthread_create()  function starts a new thread in the calling process.  The
       new thread starts execution by invoking start_routine(); arg is  passed  as  the
       sole argument of start_routine().

       The new thread terminates in one of the following ways:

       * It calls pthread_exit(3), specifying an exit status value that is available to
         another thread in the same process that calls pthread_join(3).

       * It  returns  from  start_routine().   This  is  equivalent  to  calling
         pthread_exit(3) with the value supplied in the return statement.

       * It is canceled (see pthread_cancel(3)).

       * Any of the threads in the process calls exit(3), or the main thread performs a
         return from main().  This  causes  the  termination  of  all  threads  in  the
 Manual page pthread_create(3) line 1 (press h for help or q to quit)
```

man pthread_exit

```
esinece@esinece-VirtualBox:~/Masaüstü$ man pthread_exit
```

```
PTHREAD_EXIT(3)          Linux Programmer's Manual          PTHREAD_EXIT(3)

NAME
       pthread_exit - terminate calling thread

SYNOPSIS
       #include <pthread.h>

       void pthread_exit(void *retval);

       Compile and link with -pthread.

DESCRIPTION
       The pthread_exit() function terminates the calling thread and returns a
       value via retval that (if the  thread  is  joinable)  is  available  to
       another thread in the same process that calls pthread_join(3).

       Any  clean-up handlers established by pthread_cleanup_push(3) that have
       not yet been popped, are popped (in the reverse of the order  in  which
       they  were pushed) and executed.  If the thread has any thread-specific
       data, then, after the clean-up handlers have been executed, the  corre-
       sponding destructor functions are called, in an unspecified order.

       When a thread terminates, process-shared resources (e.g., mutexes, con-
       dition variables, semaphores, and file descriptors) are  not  released,
       and functions registered using atexit(3) are not called.

 Manual page pthread_exit(3) line 1 (press h for help or q to quit)
```

Example Program 1

```
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  void* print_message_function(void *ptr){
6      char *message;
7      // interpreting as char *
8      message = (char *) ptr;
9      printf("\n %s \n", message);
10     // terminating the thread
11     pthread_exit(NULL);
12 }
13
```

Example Program 1

```
14  int main(){
15      pthread_t thread1, thread2, thread3;
16      char *message1 = "Hello";
17      char *message2 = "World";
18      char *message3 = "!...";
19      // creating 3 threads using print_message_function as the start routine
20      // and message1, message2 and message3 as the arguments for the start routine
21      if(pthread_create(&thread1,NULL,print_message_function,(void *)message1)){
22          fprintf(stderr,"pthread_create failure\n");
23          exit(-1);
24      }
25      if(pthread_create(&thread2,NULL,print_message_function,(void *)message2)){
26          fprintf(stderr,"pthread_create failure\n");
27          exit(-1);
28      }
29      if(pthread_create(&thread3,NULL,print_message_function,(void *)message3)){
30          fprintf(stderr,"pthread_create failure\n");
31          exit(-1);
32      }
33      // to block main to support the threads it created until they terminate
34      pthread_exit(NULL);
35  }
```

Compiling a Program Including Thread/s

- These applications should be linked with thread library. Sample, proper compilation:
  `gcc source.c -o output -pthread`

Output of the Example Program 1

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder$ gcc -pthread
 Example1.c -o output
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder$ ./output

 !...

 World

 Hello
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder$
```

Join

- Sometimes a thread is created to do a specific part of a job. Another thread should wait until that thread completed. Waiting mechanism can be done with the pthread_join() function.
- "Join" is a way to achieve synchronization between threads
- pthread_join() stops the calling thread from executing until the thread at the specified id ends.

man pthread_join

```
esinece@esinece-VirtualBox:~/Masaüstü$ man pthread_join
```

```
PTHREAD_JOIN(3)            Linux Programmer's Manual           PTHREAD_JOIN(3)

NAME
       pthread_join - join with a terminated thread

SYNOPSIS
       #include <pthread.h>

       int pthread_join(pthread_t thread, void **retval);

       Compile and link with -pthread.

DESCRIPTION
       The pthread_join() function waits for the thread specified by thread to
       terminate.  If that thread has already terminated, then  pthread_join()
       returns immediately.  The thread specified by thread must be joinable.

       If  retval  is  not NULL, then pthread_join() copies the exit status of
       the target thread (i.e., the value that the target thread  supplied  to
       pthread_exit(3))  into the location pointed to by *retval.  If the tar-
       get thread was canceled, then PTHREAD_CANCELED is placed in *retval.

       If multiple threads simultaneously try to join with  the  same  thread,
       the  results  are  undefined.   If the thread calling pthread_join() is
       canceled, then the target thread will remain joinable  (i.e.,  it  will
       not be detached).

 Manual page pthread_join(3) line 1 (press h for help or q to quit)
```

Example Program 2

```
1  #include <pthread.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <math.h>
5  #define NUM_THREADS 4
6
7  void *BusyWork(void *t){
8      int i;
9      long tid;
10     double result=0.0;
11     tid = (long)t;
12     printf("Thread %ld starting...\n", tid);
13     for (i=0; i<1000000; i++){
14         result = result + sin(i) * tan(i);
15     }
16     printf("Thread %ld done. Result = %e\n", tid, result);
17     pthread_exit((void*) t);
18 }
19
```

Example Program 2

```
20  int main (int argc, char *argv[]){
21      pthread_t thread[NUM_THREADS];
22      pthread_attr_t attr;
23      int rc;
24      long t;
25      void *status;
26      // Initialize and set thread detach state attribute
27      // Only threads that are created as joinable can be joined
28      // If a thread is created as detached(PTHREAD_CREATE_DETACHED), it cannot be joined
29      pthread_attr_init(&attr);
30      pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
31      for(t=0; t<NUM_THREADS; t++) {
32          printf("Main: creating thread %ld\n", t);
33          // creating thread t
34          rc = pthread_create(&thread[t], &attr, BusyWork, (void *)t);
35          if (rc) {
36              printf("ERROR; return code from pthread_create() is %d\n", rc);
37              exit(-1);
38          }
39      }
```

Example Program 2

```
40      // Free library resources used by the attribute
41      pthread_attr_destroy(&attr);
42      // Join operation is used for synchronization between threads by blocking the
43      // calling thread until the specified thread (with given threadid) terminates
44      for(t=0; t<NUM_THREADS; t++) {
45          rc = pthread_join(thread[t], &status);
46          if (rc) {
47              printf("ERROR; return code from pthread_join() is %d\n", rc);
48              exit(-1);
49          }
50          printf("Main: completed join with thread %ld having a status of %ld\n",t,(long)status);
51      }
52      printf("Main: program completed. Exiting.\n");
53      // to block main to support the threads it created until they terminate
54      pthread_exit(NULL);
55  }
```

Output of the Example Program 2

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder$ gcc -pthread
 Example2.c -lm -o output
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder$ ./output
Main: creating thread 0
Main: creating thread 1
Main: creating thread 2
Main: creating thread 3
Thread 3 starting...
Thread 2 starting...
Thread 1 starting...
Thread 0 starting...
Thread 2 done. Result = -3.153838e+06
Thread 0 done. Result = -3.153838e+06
Main: completed join with thread 0 having a status of 0
Thread 3 done. Result = -3.153838e+06
Thread 1 done. Result = -3.153838e+06
Main: completed join with thread 1 having a status of 1
Main: completed join with thread 2 having a status of 2
Main: completed join with thread 3 having a status of 3
Main: program completed. Exiting.
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder$
```

Example Program 3

```c
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>

int myglobal;

void* thread_function(void *arg){
    int i,j;
    // changing the value of myglobal in thread_function
    for(i=0;i<20;i++){
        //myglobal++;
        j=myglobal;
        j=j+1;
        printf(".");
        // to force writing all user-space buffered data to stdout
        fflush(stdout);
        sleep(1);
        myglobal=j;
    }
    pthread_exit(NULL);
}
```

Example Program 3

```
23  int main(void){
24      pthread_t mythread;
25      int i;
26      myglobal=0;
27      // creating a thread using thread_function as the start routine
28      if(pthread_create(&mythread,NULL,thread_function,NULL)){
29          printf("error creating thread");
30          abort();
31      }
32      // changing the value of myglobal in main()
33      for(i=0;i<20;i++){
34          myglobal = myglobal+1;
35          printf("o");
36          // to force writing all user-space buffered data to stdout
37          fflush(stdout);
38          sleep(1);
39      }
40      printf("\nmyglobal equals %d\n",myglobal);
41      // to block main to support the threads it created until they terminate
42      pthread_exit(NULL);
43  }
```

Output of the Example Program 3

```
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder$ gcc -pthread
 Example3.c -o output
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder$ ./output
o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.o.
myglobal equals 20
musty@musty-VirtualBox:/media/sf_virtualbox_shared_folder$
```

References

1. https://computing.llnl.gov/tutorials/pthreads/
2. https://docs.google.com/presentation/d/1nfUGmM9W8tA4GSh4UucbOrZDwO01udlf/edit?usp=sharing&ouid=116515393822328287464&rtpof=true&sd=true

Synchronized Operation of Process or Threads

- Sometimes synchronization is needed between different processes or between threads implemented in the same process because these threads want to access a shared resource provided by the operating system or maintained by the process itself in order to perform their tasks.
- For example, threads are trying to access a log file. If two threads try to write to the log file at the same time, the logs written to the file will be mixed up and become unreadable.

Mutex Creation

- Where default mutex attributes are appropriate, the macro `PTHREAD_MUTEX_INITIALIZER` can be used to statically generate mutexes.
- Dynamically creation of mutex the parameter `attr` with a call to `int pthread_mutex_init()` with the specified parameter `NULL` produces the same result, except that no error checking is performed.

Mutex Operations

- For each resource shared by different threads, a Mutex is created to regulate access to the resource:
  - `pthread_mutex_t thread`
- The thread dealing with the source tries to take ownership of the Mutex (Acquire).
  - `int pthread_mutex_lock(pthread_mutex_t *mutex);`
- While the thread holding the Mutex completes the Critical Section(CS) and leaves the Mutex, the other thread waiting for the Mutex to be released is awakened and takes ownership of the Mutex and gains access to the shared resource.
  - `int pthread_mutex_unlock(pthread_mutex_t *mutex);`
- To terminate the mutex created at runtime afterwards:
  - `int pthread_mutex_destroy(pthread_mutex_t *mutex);`

Mutex Example

```
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
int myglobal;
pthread_mutex_t lock;
void* thread_function(void *arg){
        int i,j;
        // changing the value of myglobal in thread_function
        for(i=0;i<20;i++){
                pthread_mutex_lock(&lock); //Entering the critical
                j=myglobal;
                j=j+1;
                printf(".");
                // to force writing all user-space buffered data t
                fflush(stdout);
                myglobal=j;
                pthread_mutex_unlock(&lock); //Exiting the critica
                sleep(1);
        }
        pthread_exit(NULL);
```

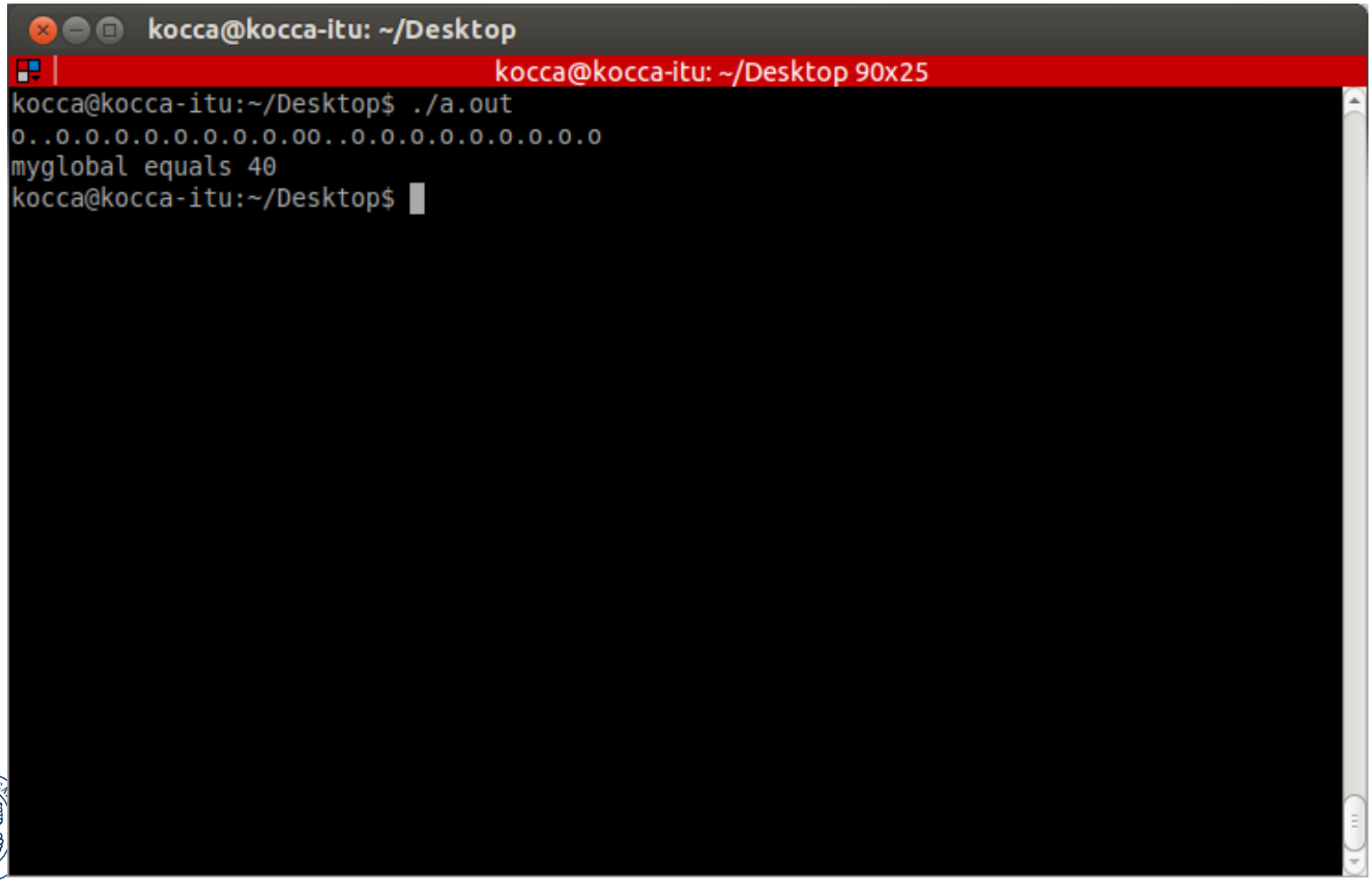
Mutex Example

```
int main(void){
        pthread_t mythread;
        int i;
        myglobal=0;
        if (pthread_mutex_init(&lock, NULL) != 0)
        {
                printf("\n mutex init failed\n");
                return 1;
        }
        // creating a thread using thread_function as the start ro
        if(pthread_create(&mythread,NULL,thread_function,NULL)){
                printf("error creating thread");
                abort();
        }
```

Mutex Example

```
// changing the value of myglobal in main()
for(i=0;i<20;i++){
        pthread_mutex_lock(&lock); //Entering the critical
        myglobal = myglobal+1;
        printf("o");
        // to force writing all user-space buffered data t
        fflush(stdout);
        pthread_mutex_unlock(&lock); //Exiting the critica
        sleep(1);
}
pthread_join(mythread, NULL);
pthread_mutex_destroy(&lock);
printf("\nmyglobal equals %d\n",myglobal);
// to block main to support the threads it created until t
pthread_exit(NULL);
}
```

Mutex Example



Semaphore

- POSIX semaphores provide the necessary synchronization infrastructure to access a common resource used by different threads or processes.
- Instead of locking and unlocking on semaphores, operations such as increasing and decreasing the semaphore value can be performed.

Semaphore Operations

- By including the library `<semaphore.h>`, functions that can be used in semaphore operations can be accessed.
- To create semaphore:
  - `sem_init(sem_t *sem, int pshared, unsigned int value);`
    - `pshared == 0` –> The semaphore is used within the threads of the process. Therefore, it can be kept in a global variable or in a allocated space on the heap, without the need to use shared memory.
    - `pshared != 0` –> The semaphore can be used between different processes. In this case, the address in the first parameter should point to a location on shared memory.
  - If `sem_init()` is called more than once for the same semaphore, the system cannot be stable. Therefore, it should be guaranteed that each semaphore is initialized only once.
- To terminate the semaphore:
  - `int sem_destroy(sem_t *sem);`
    - Before the semaphore is terminated, the memory region where it is kept should not be freed.

Semaphore Operations

- To lock or wait for a semaphore:
  - `int sem_wait(sem_t *sem);`
    - The value of the semaphore is decremented by 1.
    - If the corresponding semaphore value is greater than 1, the reduction is performed instantly and the function returns.
    - If the value of the semaphore is already equal to 0, the sem_wait function waits for the value of the semaphore to increase by 1. When it increases, it immediately decreases by 1 and the function returns.
- To release or signal a semaphore:
  - `int sem_post(sem_t *sem);`
    - It is used to increase the value of the semaphore by 1.
    - If the value of the semaphore is already 0 and is blocked because another process is waiting for the same semaphore, the corresponding process is awakened.
    - In the above case, if multiple processes are blocked while waiting for the same semaphore, it is not guaranteed which process will be woken up.
- To get the current value of an existing semaphore:
  - `sem_getvalue(sem_t *sem, int *value)`

Handling Signals

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

```c
// 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

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

```
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
```

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  }
```

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  }
```

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
```

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);
```

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 The 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
```