```c
/*                              SEMAPHORES

Semaphores are system variables used for process synchronization. You may
think of a semaphore, s, as a variable maintained by the system. A semaphore
can be obtained by a semget() system call. Its initial value can be set by
the semctl() system call. There are two common operations that a process
can perform on a semaphore, s, namely:

        P(s) or wait(s)   :  If the value of s is greater than 0, then
                             this operation decrements the value of s and
                             the calling process continues. Otherwise, if
                             s is 0, then the calling process is blocked on s.

        V(s) or signal(s) :  If any process is blocked on s, then this
                             unblocks (wakes up) the earliest among the
                             processes blocked on s. Otherwise, the value
                             of the semaphore is incremented.

In UNIX/Linux, both P(s) and V(s) can be done with the semop() system call
with appropriate parameters.

*/

#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>/* Include this to use semaphores */

/* We will define the P(s) and V(s) operations in terms of the semop()
  system call. The syntax of semop is as follows:

        int semop ( int semid, struct sembuf *sops, unsigned nsops)

  where semid is the semaphore identifier returned by the semget()
  system call. The second parameter is a pointer to a structure whichs
  we must pass. The fields of this structure indicates whether we wish
  to perform a P(s) operation or a V(s) operation. Refer to the system
  manual for the third parameter -- we will always use 1 for this
  parameter.
*/

#define P(s) semop(s, &pop, 1)  /* pop is the structure we pass for doing
                                   the P(s) operation */
#define V(s) semop(s, &vop, 1)  /* vop is the structure we pass for doing
                                   the V(s) operation */

main()
{
        int *a, *b;
        int i,j, count = 50, status;

        int semid1, semid2 ;
```

```
struct sembuf pop, vop ;

/* In the following  system calls, the second parameter indicates the
   number of semaphores under this semid. Throughout this lab,
   give this parameter as 1. If we require more semaphores, we
   will take them under different semids through separate semget()
   calls.
*/

semid1 = semget(IPC_PRIVATE, 1, 0777|IPC_CREAT);
semid2 = semget(IPC_PRIVATE, 1, 0777|IPC_CREAT);

/* The following system calls sets the values of the semaphores
   semid1 and semid2 to 0 and 1 respectively. */

semctl(semid1, 0, SETVAL, 0);
semctl(semid2, 0, SETVAL, 1);

/*  We now initialize the sembufs pop and vop so that pop is used
    for P(semid) and vop is used for V(semid). For the fields
    sem_num and sem_flg refer to the system manual. The third
    field, namely sem_op indicates the value which should be added
    to the semaphore when the semop() system call is made. Going
    by the semantics of the P and V operations, we see that
    pop.sem_op should be -1 and vop.sem_op should be 1.
*/
pop.sem_num = vop.sem_num = 0;
pop.sem_flg = vop.sem_flg = 0;
pop.sem_op = -1 ; vop.sem_op = 1 ;

/*  We now illustrate a producer-consumer situation. The parent process
    acts as the producer and the child process acts as the consumer.
    Initially semid1 is zero, hence the consumer blocks. Since
    semid2 is one, the producer produces (in this case writes some
    values into the file). After this it wakes up the
    consumer through the V(semid1) call. The consumer reads the
    value and in turn performs V(semid2) to wake up the producer.
    Trace through the code and work out the values of the two
    semaphores and see how they synchronize the producer and the
    consumer to wait for each other.
*/
if (fork() == 0) {

        /* Child Process::  Consumer */
        FILE *fp;
        int data;

        while (count) {

                P(semid1);
                fp = fopen("datafile","r");
                fscanf(fp, "%d", &data);
```

```
                printf("\t\t\t\t Consumer reads %d\n",data);
                fclose(fp);
                V(semid2);
                count--;
            }


    }
    else {

            /* Parent Process:: Producer */
            FILE *fp;
            int data = 0;

            while (count) {

                sleep(1);

                P(semid2);
                fp = fopen("datafile","w");
                fprintf(fp, "%d\n", data);
                printf("Producer writes %d\n", data);
                data++;
                fclose(fp);
                V(semid1);
                count--;
            }
            wait(&status);


            /* Semaphores need to be
              deleted after they are used. In this case also,
              exactly one process should delete it after making
              sure that noone else is using it.
             */

            semctl(semid1, 0, IPC_RMID, 0);
            semctl(semid2, 0, IPC_RMID, 0);
    }
}
```