Exp# 5e **Producer-Consumer problem**

Aim

To synchronize producer and consumer processes using semaphore.

Algorithm

1. Create a shared memory segment *BUFSIZE* of size 1 and attach it.
2. Obtain semaphore id for variables *empty*, *mutex* and *full* using semget function.
3. Create semaphore for *empty*, *mutex* and *full* as follows:
   a. Declare *semun*, a union of specific commands.
   b. The initial values are: 1 for mutex, N for empty and 0 for full
   c. Use semctl function with SETVAL command
4. Create a child process using fork system call.
   a. Make the parent process to be the *producer*
   b. Make the child process to the *consumer*
5. The *producer* produces 5 items as follows:
   a. Call *wait* operation on semaphores *empty* and *mutex* using semop function.
   b. Gain access to buffer and produce data for consumption
   c. Call *signal* operation on semaphores *mutex* and *full* using semop function.
6. The *consumer* consumes 5 items as follows:
   a. Call *wait* operation on semaphores *full* and *mutex* using semop function.
   b. Gain access to buffer and consume the available data.
   c. Call *signal* operation on semaphores *mutex* and *empty* using semop function.
7. Remove shared memory from the system using shmctl with IPC_RMID argument
8. Stop

Result

Thus synchronization between producer and consumer process for access to a shared memory segment is implemented.

Program

```
/* Producer-Consumer problem using semaphore – pcsem.c */

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/sem.h>

#define N 5
#define BUFSIZE 1
#define PERMS 0666

int *buffer;
int nextp = 0, nextc = 0;
int mutex, full, empty;    /* semaphore variables */

void producer()
{
   int data;
   if(nextp == N)
      nextp = 0;
   printf("Enter data for producer to produce : ");
   scanf("%d",(buffer + nextp));
   nextp++;
}

void consumer()
{
   int g;
   if(nextc == N)
      nextc = 0;
   g = *(buffer + nextc++);
   printf("\nConsumer consumes data %d", g);
}

void sem_op(int id, int value)
{
   struct sembuf op;
   int v;
   op.sem_num = 0;
   op.sem_op = value;
   op.sem_flg = SEM_UNDO;
   if((v = semop(id, &op, 1)) < 0)
      printf("\nError executing semop instruction");
}
```

```
void sem_create(int semid, int initval)
{
   int semval;
   union semun
   {
      int val;
      struct semid_ds *buf;
      unsigned short *array;
   } s;

   s.val = initval;
   if((semval = semctl(semid, 0, SETVAL, s)) < 0)
      printf("\nError in executing semctl");
}

void sem_wait(int id)
{
   int value = -1;
   sem_op(id, value);
}

void sem_signal(int id)
{
   int value = 1;
   sem_op(id, value);
}

main()
{
   int shmid, i;
   pid_t pid;

   if((shmid = shmget(1000, BUFSIZE, IPC_CREAT|PERMS)) < 0)
   {
      printf("\nUnable to create shared memory");
      return;
   }
   if((buffer = (int*)shmat(shmid, (char*)0, 0)) == (int*)-1)
   {
      printf("\nShared memory allocation error\n");
      exit(1);
   }

   if((mutex = semget(IPC_PRIVATE, 1, PERMS|IPC_CREAT)) == -1)
   {
      printf("\nCan't create mutex semaphore");
      exit(1);
   }
```

```
   if((empty = semget(IPC_PRIVATE, 1, PERMS|IPC_CREAT)) == -1)
   {
      printf("\nCan't create empty semaphore");
      exit(1);
   }
   if((full = semget(IPC_PRIVATE, 1, PERMS|IPC_CREAT)) == -1)
   {
      printf("\nCan't create full semaphore");
      exit(1);
   }

   sem_create(mutex, 1);
   sem_create(empty, N);
   sem_create(full, 0);

   if((pid = fork()) < 0)
   {
      printf("\nError in process creation");
      exit(1);
   }
   else if(pid > 0)
   {
      for(i=0; i<N; i++)
      {
         sem_wait(empty);
         sem_wait(mutex);
         producer();
         sem_signal(mutex);
         sem_signal(full);
      }
   }
   else if(pid == 0)
   {
      for(i=0; i<N; i++)
      {
         sem_wait(full);
         sem_wait(mutex);
         consumer();
         sem_signal(mutex);
         sem_signal(empty);
      }
      printf("\n");
   }
}
```

Output

```
$ gcc pcsem.c

$ ./a.out
Enter data for producer to produce : 5

Enter data for producer to produce : 8
Consumer consumes data 5
Enter data for producer to produce : 4
Consumer consumes data 8
Enter data for producer to produce : 2
Consumer consumes data 4
Enter data for producer to produce : 9
Consumer consumes data 2
Consumer consumes data 9
```