Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

OPERATING SYSTEMS

LAB – ASSIGNMENT: 2

17BCE0581
SATYAM SINGH CHAUHAN

Under the Guidance
Of
Prof. GOPINATH M.P

Date: 19 September 2019

Task: Synchronization Problems

AIM

To develop a C program simulating **Reader Writer Problem for ‘n1’ number of Readers and ‘n2’ number of Writers**.

ALGORITHM

1. Get the Inputs for n1 storing Number of Readers
   n1 <- Number of Readers

2. Get the Inputs for n2 storing Number of Writers
   n1 <- Number of Writers

3. Writer Section
   i. Start
   ii. Writer requests the entry to critical section.
   iii. If allowed i.e. wait() gives a true value, it enters and performs the write. If not allowed, it keeps on waiting.
   iv. It exits the critical section.
   v. End

```
do {
    // writer requests for critical section
    wait(wrt);

    // performs the write

    // leaves the critical section
    signal(wrt);
} while(true);
```

4. Reader Section
   i. Start
   ii. Reader requests the entry to critical section.
   iii. If allowed:
      - It increments the count of number of readers inside the critical section. If this reader is the first reader entering, it locks the wrt semaphore to restrict the entry of writers if any reader is inside.
      - It then, signals mutex as any other reader is allowed to enter while others are already reading.

        - After performing reading, it exits the critical section. When exiting, it checks if no more reader is inside, it signals the semaphore “wrt” as now, writer can enter the critical section.

iv. If not allowed, it keeps on waiting.

v. End

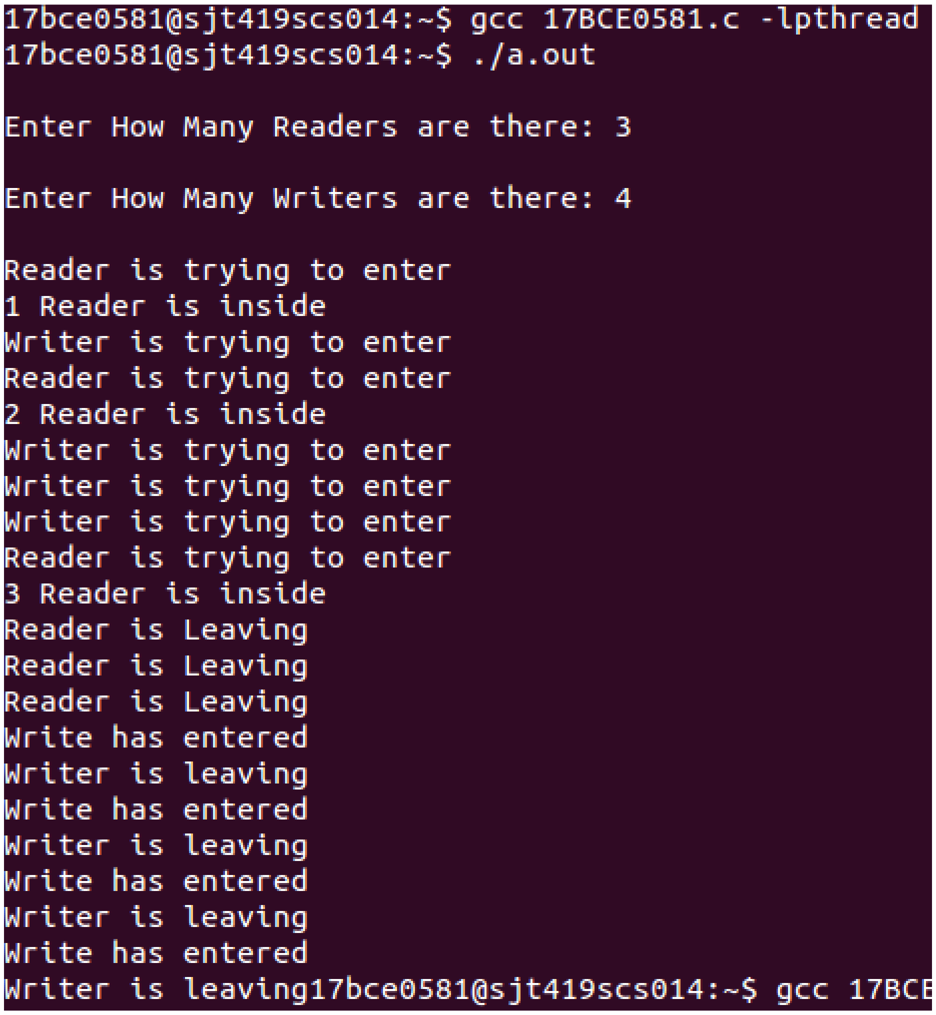
```
do {
   // Reader wants to enter the critical section
   wait(mutex);
   // The number of readers has now increased by 1
   readcnt++;
   // there is atleast one reader in the critical section
   // this ensure no writer can enter if there is even one reader
   // thus we give preference to readers here
   if (readcnt==1)
      wait(wrt);
   // other readers can enter while this current reader is inside
   // the critical section
   signal(mutex);

   // current reader performs reading here
   wait(mutex);   // a reader wants to leave

   readcnt--;

   // that is, no reader is left in the critical section,
   if (readcnt == 0)
       signal(wrt);         // writers can enter

   signal(mutex); // reader leaves

} while(true);
```

CODE

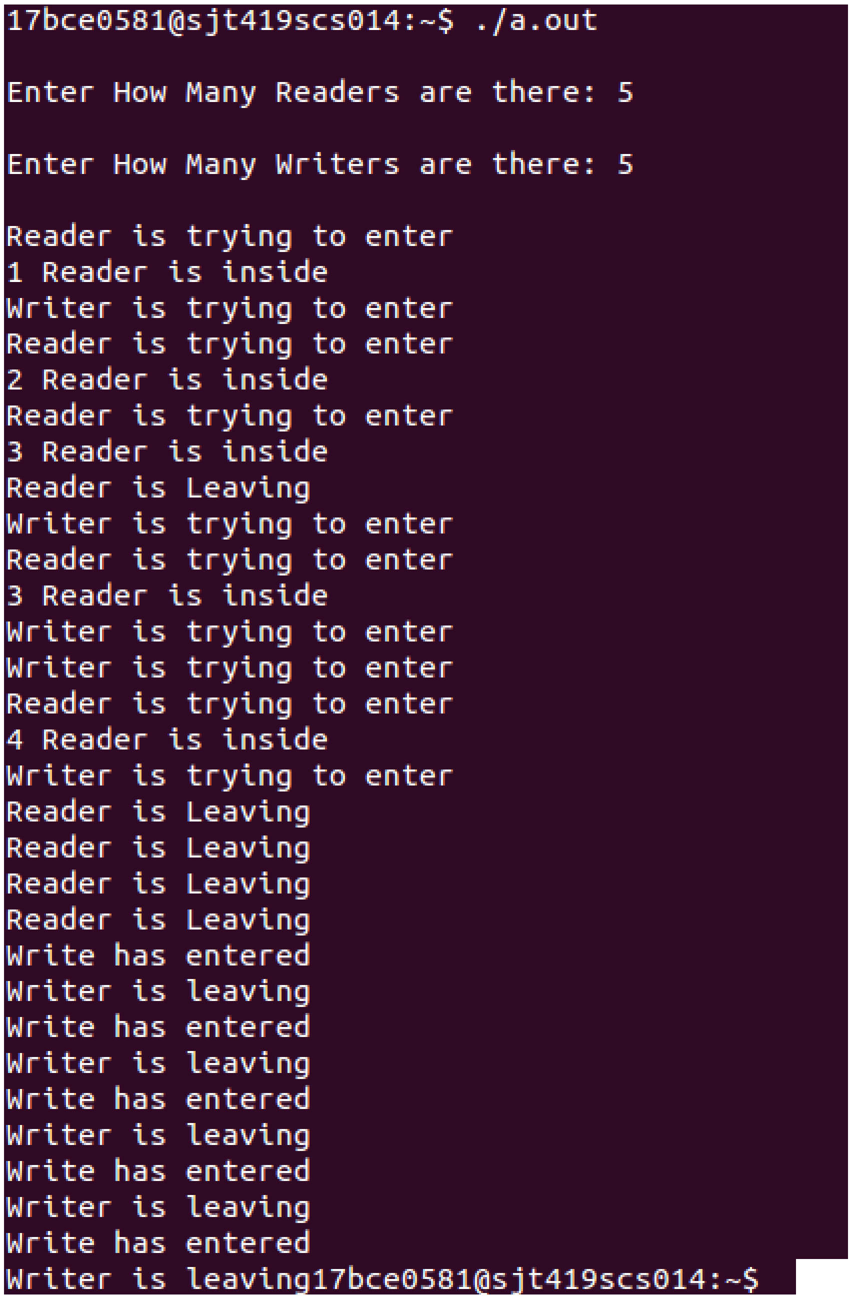
```
#include<pthread.h>
#include<semaphore.h>
#include<stdio.h>
#include<stdlib.h>

pthread_mutex_t x,wsem;
pthread_t tid;
int readcount;
void intialize()
{
pthread_mutex_init(&x,NULL);
pthread_mutex_init(&wsem,NULL);
```

```c
readcount=0;
}
void * reader (void * satyam)
{
int waittime;
waittime = rand() % 5;
printf("\nReader is trying to enter");
pthread_mutex_lock(&x);
readcount++;
if(readcount==1)
pthread_mutex_lock(&wsem);
printf("\n%d Reader is inside ",readcount);
pthread_mutex_unlock(&x);
sleep(waittime);
pthread_mutex_lock(&x);
readcount--;
if(readcount==0)
pthread_mutex_unlock(&wsem);
pthread_mutex_unlock(&x);
printf("\nReader is Leaving");
}
void * writer (void * satyam)
{
int waittime;
waittime=rand() % 3;
printf("\nWriter is trying to enter");
pthread_mutex_lock(&wsem);
printf("\nWrite has entered");
sleep(waittime);
pthread_mutex_unlock(&wsem);
printf("\nWriter is leaving");
sleep(30);
exit(0);
}
int main()
{
intialize();
int n1,n2,i;
printf("\nEnter How Many Readers are there: ");
scanf("%d",&n1);
printf("\nEnter How Many Writers are there: ");
scanf("%d",&n2);
for(i=0;i<n1;i++)
pthread_create(&tid,NULL,reader,NULL);
for(i=0;i<n2;i++)
pthread_create(&tid,NULL,writer,NULL);
sleep(30);
exit(0);
}
```

OUTPUT



```
17bce0581@sjt419scs014:~$ gcc 17BCE0581.c -lpthread
17bce0581@sjt419scs014:~$ ./a.out

Enter How Many Readers are there: 3

Enter How Many Writers are there: 4

Reader is trying to enter
1 Reader is inside
Writer is trying to enter
Reader is trying to enter
2 Reader is inside
Writer is trying to enter
Writer is trying to enter
Writer is trying to enter
Reader is trying to enter
3 Reader is inside
Reader is Leaving
Reader is Leaving
Reader is Leaving
Write has entered
Writer is leaving
Write has entered
Writer is leaving
Write has entered
Writer is leaving
Write has entered
Writer is leaving17bce0581@sjt419scs014:~$ gcc 17BCE
```



```
17bce0581@sjt419scs014:~$ ./a.out

Enter How Many Readers are there: 5

Enter How Many Writers are there: 5

Reader is trying to enter
1 Reader is inside
Writer is trying to enter
Reader is trying to enter
2 Reader is inside
Reader is trying to enter
3 Reader is inside
Reader is Leaving
Writer is trying to enter
Reader is trying to enter
3 Reader is inside
Writer is trying to enter
Writer is trying to enter
Reader is trying to enter
4 Reader is inside
Writer is trying to enter
Reader is Leaving
Reader is Leaving
Reader is Leaving
Reader is Leaving
Write has entered
Writer is leaving
Write has entered
Writer is leaving
Write has entered
Writer is leaving
Write has entered
Writer is leaving
Write has entered
Writer is leaving17bce0581@sjt419scs014:~$
```

AIM

To develop a C program simulating **Dining Philosopher Problem**.

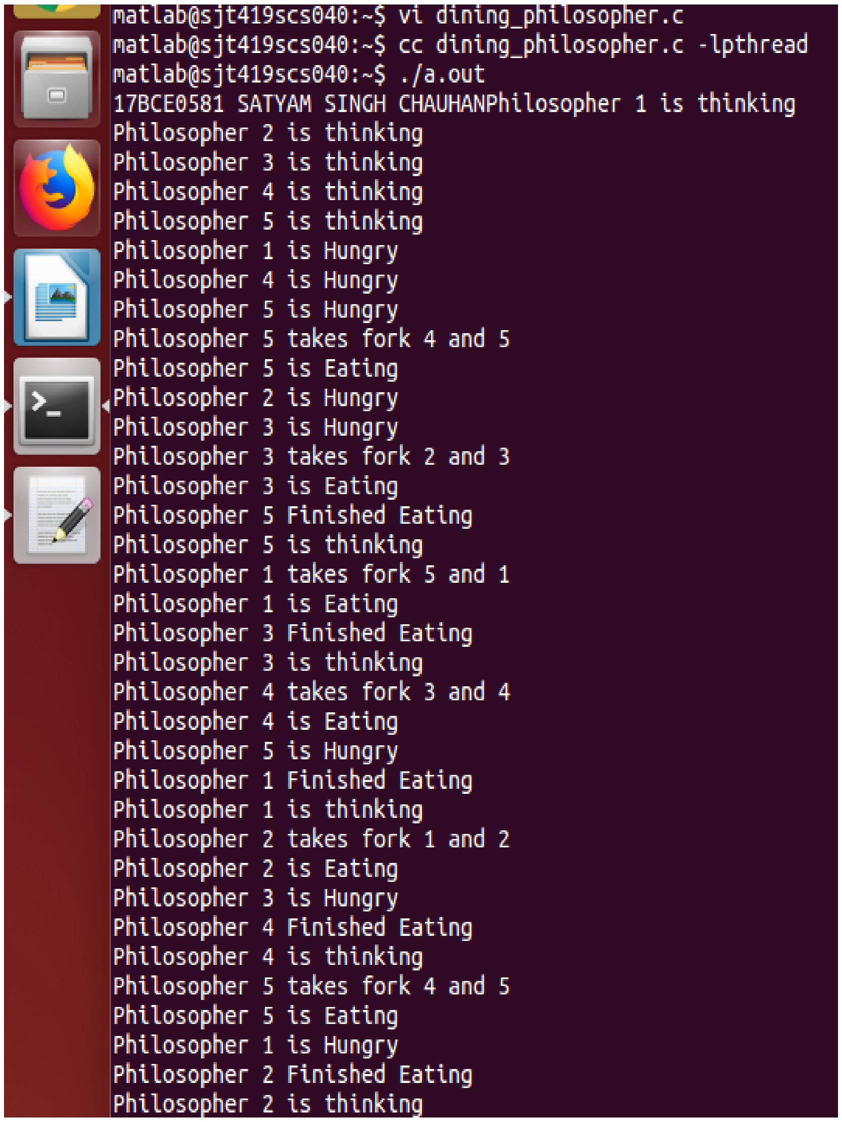
ALGORITHM

```
process P[i]
 while true do
   {  THINK;
      PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);
      EAT;
      PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])
   }
```

CODE

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

#define N 5
#define THINKING 2
#define HUNGRY 1
#define EATING 0
#define LEFT (phnum + 4) % N
#define RIGHT (phnum + 1) % N

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };

sem_t mutex;
sem_t S[N];

void test(int phnum)
{
    if (state[phnum] == HUNGRY
        && state[LEFT] != EATING
        && state[RIGHT] != EATING) {
        state[phnum] = EATING;
        sleep(2);
        printf("Philosopher %d takes fork %d and %d\n",
                      phnum + 1, LEFT + 1, phnum + 1);

        printf("Philosopher %d is Eating\n", phnum + 1);
        sem_post(&S[phnum]);
    }
}

void take_fork(int phnum)
{
    sem_wait(&mutex);

    state[phnum] = HUNGRY;

    printf("Philosopher %d is Hungry\n", phnum + 1);
    test(phnum);

    sem_post(&mutex);

    sem_wait(&S[phnum]);

    sleep(1);
}

void put_fork(int phnum)
{

    sem_wait(&mutex);

    state[phnum] = THINKING;

    printf("Philosopher %d Finished Eating\n", phnum + 1);
    printf("Philosopher %d is thinking\n", phnum + 1);

    test(LEFT);
    test(RIGHT);

    sem_post(&mutex);
}

void* philospher(void* num)
{

    while (1) {

        int* i = num;

        sleep(1);

        take_fork(*i);

        sleep(0);

        put_fork(*i);
    }
} ;

int main()
{
    printf("17BCE0581 SATYAM SINGH CHAUHAN");
    int i;
    pthread_t thread_id[N];

    sem_init(&mutex, 0, 1);
    for (i = 0; i < N; i++)

        sem_init(&S[i], 0, 0);
    for (i = 0; i < N; i++) {
        pthread_create(&thread_id[i], NULL,
                       philospher, &phil[i]);
        printf("Philosopher %d is thinking\n", i + 1);
    }
    for (i = 0; i < N; i++)

        pthread_join(thread_id[i], NULL);
}
```

OUTPUT



```
matlab@sjt419scs040:~$ vi dining_philosopher.c
matlab@sjt419scs040:~$ cc dining_philosopher.c -lpthread
matlab@sjt419scs040:~$ ./a.out
17BCE0581 SATYAM SINGH CHAUHANPhilosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 1 is Hungry
Philosopher 4 is Hungry
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 Finished Eating
Philosopher 5 is thinking
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 3 Finished Eating
Philosopher 3 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 5 is Hungry
Philosopher 1 Finished Eating
Philosopher 1 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 3 is Hungry
Philosopher 4 Finished Eating
Philosopher 4 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 1 is Hungry
Philosopher 2 Finished Eating
Philosopher 2 is thinking
```

AIM

To develop a C program simulating **Producer Consumer Problem**.

ALGORITHM

Wait and Signal Function

```
wait(S){
while(S<=0);   // busy waiting
S--;
}

signal(S){
S++;
}
```

Producer

```
do {
      .
      . PRODUCE ITEM
      .
      wait(empty);
      wait(mutex);
      .
      . PUT ITEM IN BUFFER
      .
      signal(mutex);
      signal(full);

} while(1);
```

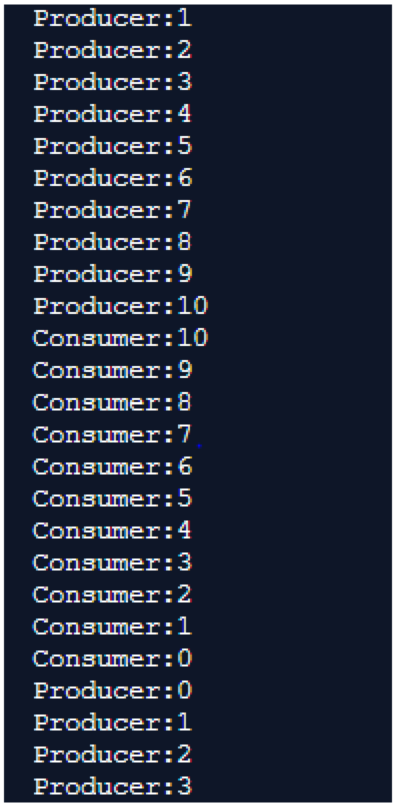
Consumer

```
do {

      wait(full);
      wait(mutex);
      .       .
      .  REMOVE ITEM FROM BUFFER
      .
      signal(mutex);
      signal(empty);
      .
      . CONSUME ITEM
      .
} while(1);
```

CODE

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

#define Buffer_Limit 10

int Buffer_Index_Value = 0;
char *Buffer_Queue;

pthread_mutex_t mutex_variable = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t Buffer_Queue_Not_Full = PTHREAD_COND_INITIALIZER;
pthread_cond_t Buffer_Queue_Not_Empty = PTHREAD_COND_INITIALIZER;

void *Consumer()
{
      while(1)
      {
            pthread_mutex_lock(&mutex_variable);
            if(Buffer_Index_Value == -1)
            {
                  pthread_cond_wait(&Buffer_Queue_Not_Empty, &mutex_variable);
            }
            printf("Consumer:%d\n", Buffer_Index_Value--);
            pthread_mutex_unlock(&mutex_variable);
            pthread_cond_signal(&Buffer_Queue_Not_Full);
      }
}

void *Producer()
{
      while(1)
      {
            pthread_mutex_lock(&mutex_variable);
            if(Buffer_Index_Value == Buffer_Limit)
            {
                  pthread_cond_wait(&Buffer_Queue_Not_Full, &mutex_variable);
            }
            Buffer_Queue[Buffer_Index_Value++] = '@';
            printf("Producer:%d\n", Buffer_Index_Value);
            pthread_mutex_unlock(&mutex_variable);
            pthread_cond_signal(&Buffer_Queue_Not_Empty);
      }
}

int main()
{
      pthread_t producer_thread_id, consumer_thread_id;
      Buffer_Queue = (char *) malloc(sizeof(char) * Buffer_Limit);
      pthread_create(&producer_thread_id, NULL, Producer, NULL);
      pthread_create(&consumer_thread_id, NULL, Consumer, NULL);
      pthread_join(producer_thread_id, NULL);
      pthread_join(consumer_thread_id, NULL);
      return 0;
}
```

OUTPUT



```
Producer:1
Producer:2
Producer:3
Producer:4
Producer:5
Producer:6
Producer:7
Producer:8
Producer:9
Producer:10
Consumer:10
Consumer:9
Consumer:8
Consumer:7
Consumer:6
Consumer:5
Consumer:4
Consumer:3
Consumer:2
Consumer:1
Consumer:0
Producer:0
Producer:1
Producer:2
Producer:3
```