Course Title : Operating system Lab 22MAI0015
Course code : MCSE504P NIDHI SINGH

LAB :- 3

DATE:- 03-11-2022

Process Synchronisation-IPC

1. Implement the deadlock-free solution to producer consumer problem using Semaphore.

2. Implement the deadlock-free solution to Reader writer problem using Semaphore.

3.Implement the deadlock-free solution to Dining philosopher problem using Semaphore.

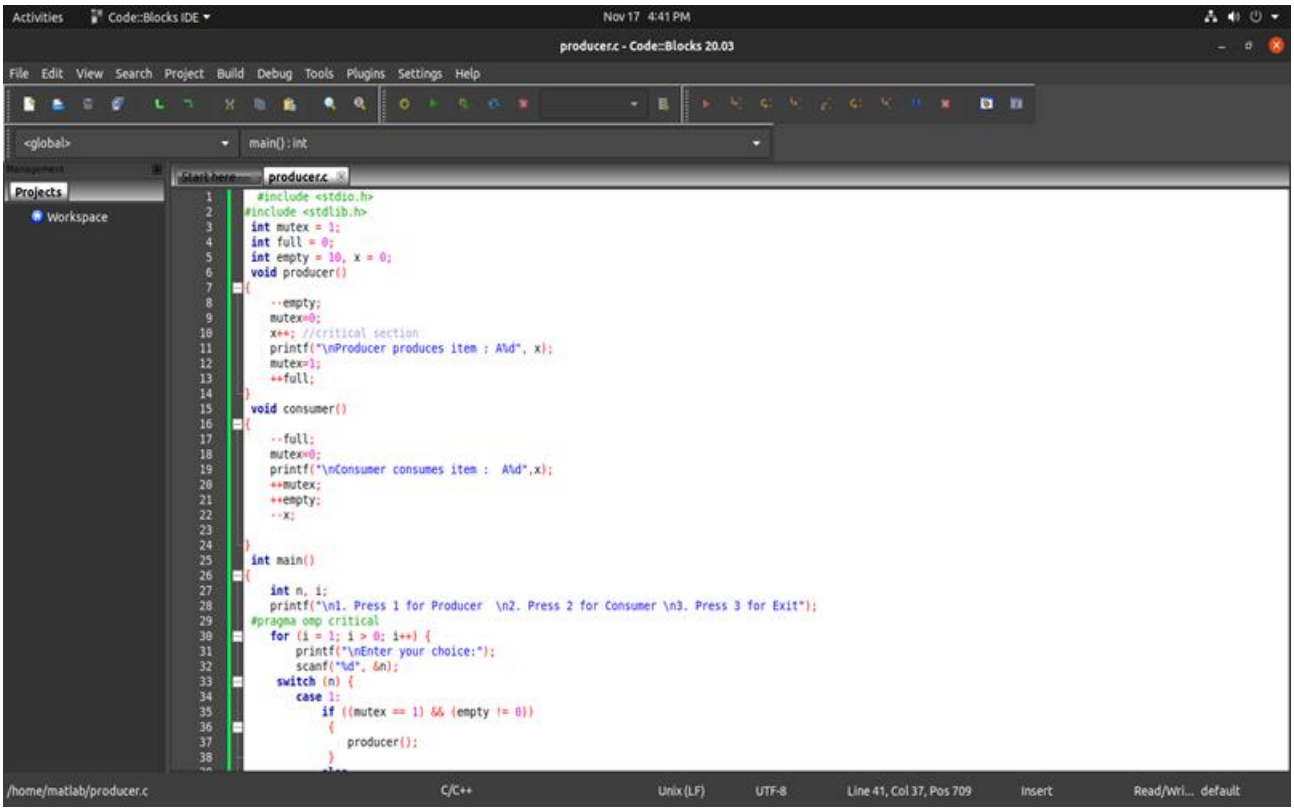
4.Implement the deadlock-free solution to Reader writer problem using monitors.

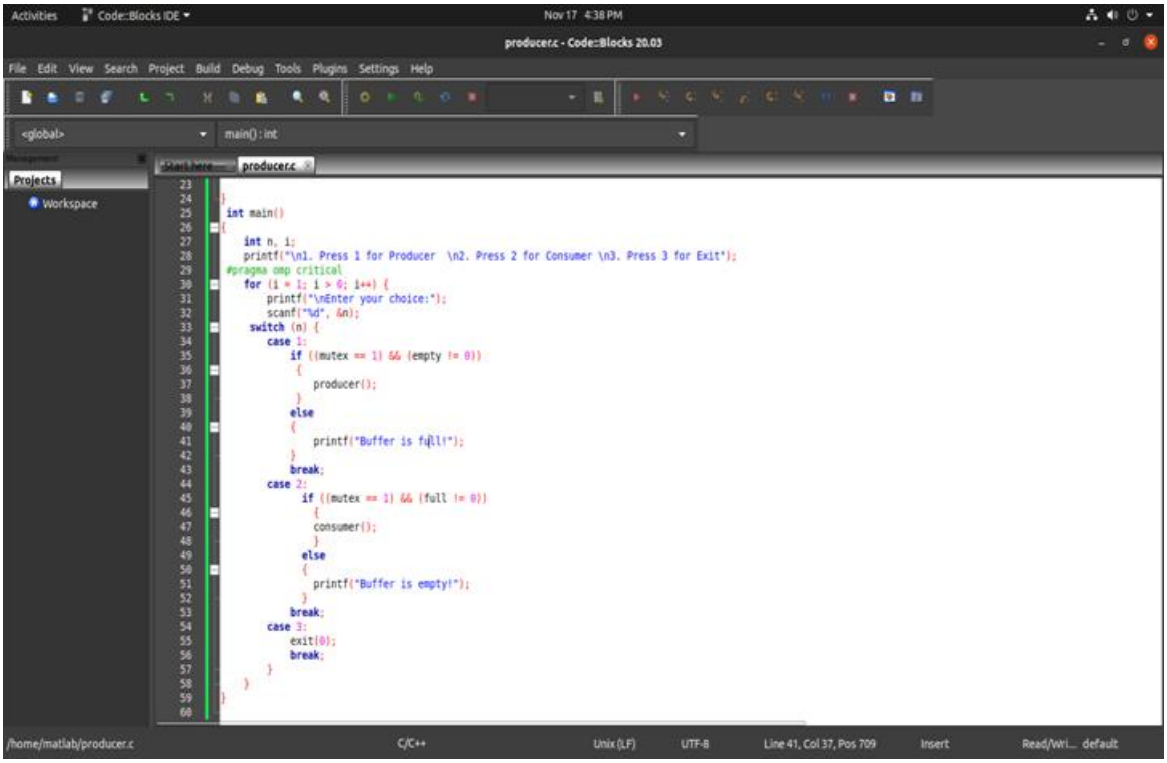
1. Implement the deadlock-free solution to producer consumer problem using Semaphore.

Code:-

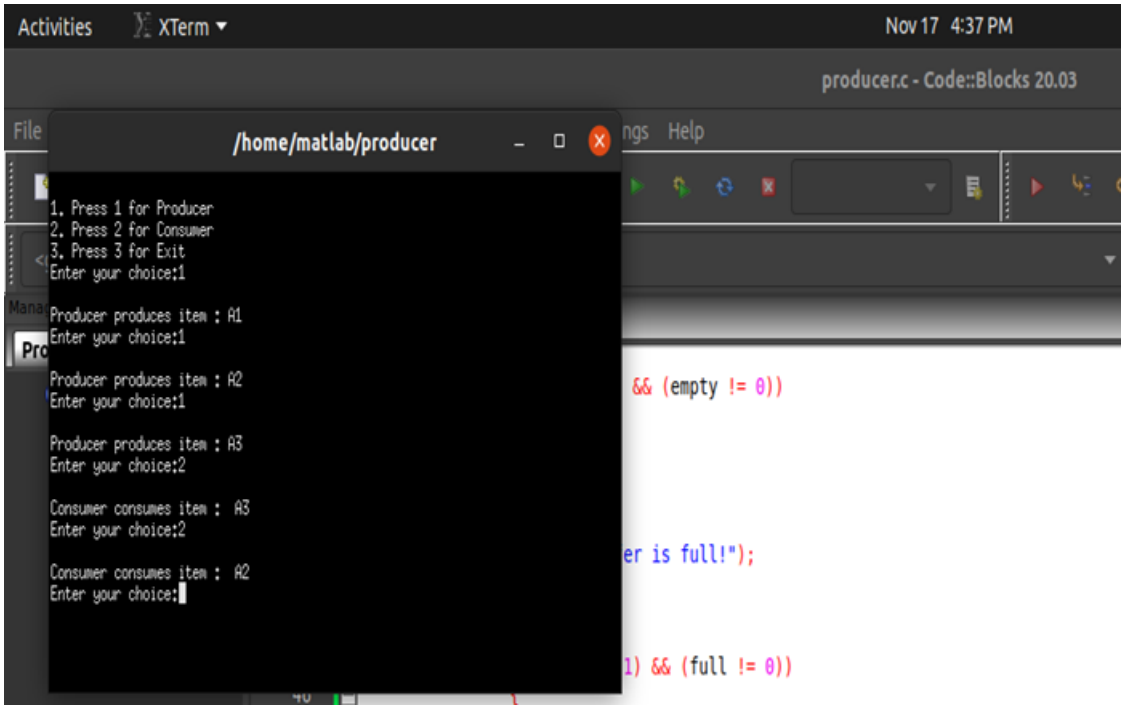
```
#include <stdio.h>
#include <stdlib.h>
int mutex = 1;
int full = 0;
int empty = 10, x = 0;
void producer()
{
  --empty;
  mutex=0;
  x++; //critical section
  printf("\nProducer produces item : A%d", x);
  mutex=1;
  ++full;
}
void consumer()
{
  --full;
  mutex=0;
  printf("\nConsumer consumes item :  A%d",x);
  ++mutex;
  ++empty;
  --x;

}
int main()
{
  int n, i;
  printf("\n1. Press 1 for Producer  \n2. Press 2 for Consumer \n3. Press 3
for Exit");
#pragma omp critical
  for (i = 1; i > 0; i++) {
    printf("\nEnter your choice:");
    scanf("%d", &n);
   switch (n) {
     case 1:
       if ((mutex == 1) && (empty != 0))
        {
          producer();
        }
       else
       {
          printf("Buffer is full!");
       }
```

```
        break;
      case 2:
          if ((mutex == 1) && (full != 0))
           {
           consumer();
           }
          else
          {
            printf("Buffer is empty!");
          }
        break;
      case 3:
        exit(0);
        break;
      }
   }
}
```




Activities
Code::Blocks IDE
Nov 17 4:38 PM
producer.c - Code::Blocks 20.03
File Edit View Search Project Build Debug Tools Plugins Settings Help
<global>
main() : int
Projects
Workspace
producer.c
}
int main()
{
int n, i;
printf("\n1. Press 1 for Producer \n2. Press 2 for Consumer \n3. Press 3 for Exit");
#pragma omp critical
for (i = 1; i > 0; i++) {
printf("\nEnter your choice:");
scanf("%d", &n);
switch (n) {
case 1:
if ((mutex == 1) && (empty != 0))
{
producer();
}
else
{
printf("Buffer is full!");
}
break;
case 2:
if ((mutex == 1) && (full != 0))
{
consumer();
}
else
{
printf("Buffer is empty!");
}
break;
case 3:
exit(0);
break;
}
}
}
/home/matlab/producer.c
C/C++
Unix (LF)
UTF-8
Line 41, Col 37, Pos 709
Insert
Read/Wri... default

Output:-

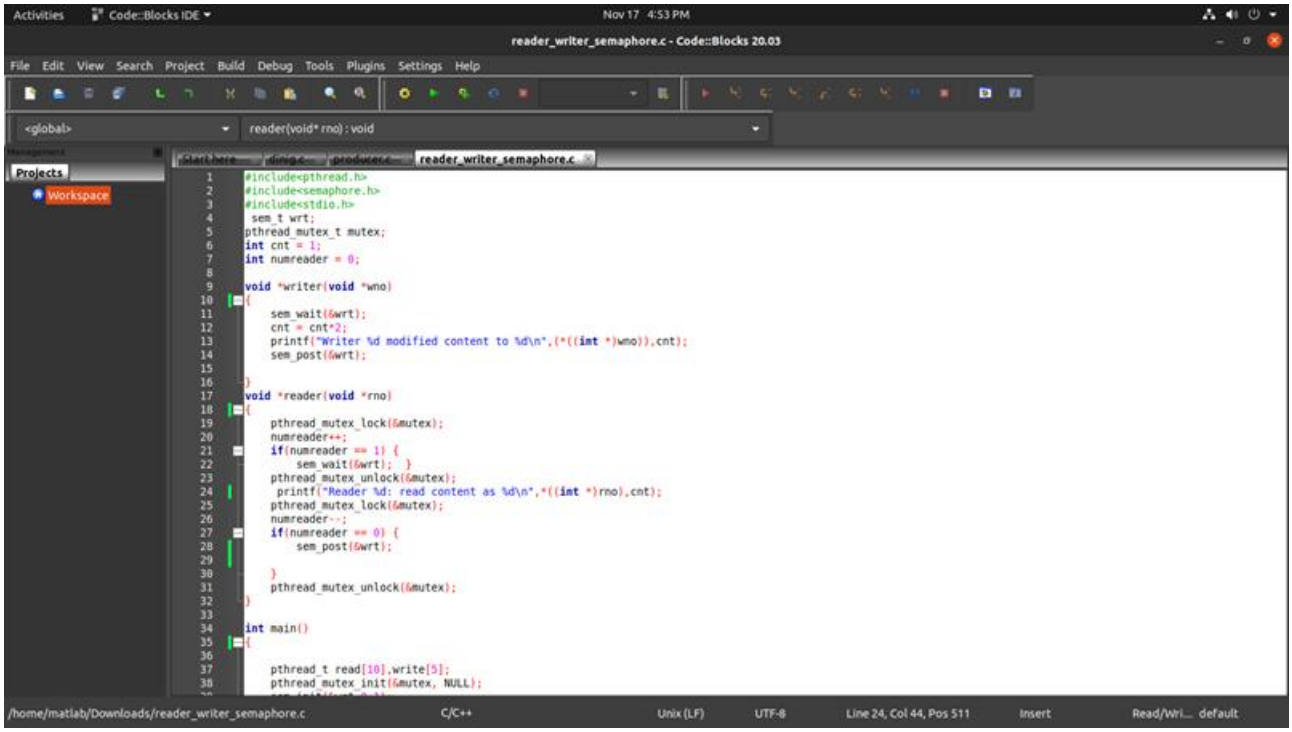


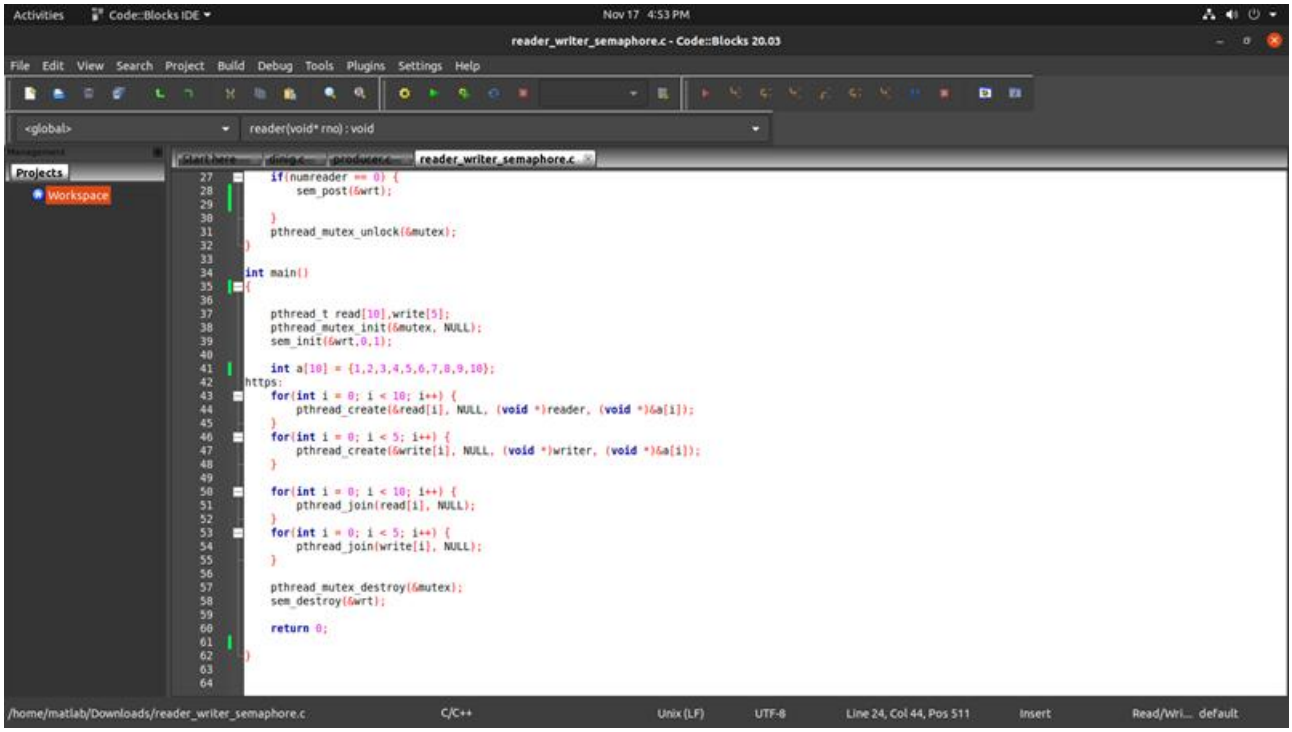
2. Implement the deadlock-free solution to Reader writer problem using Semaphore.

Code:-

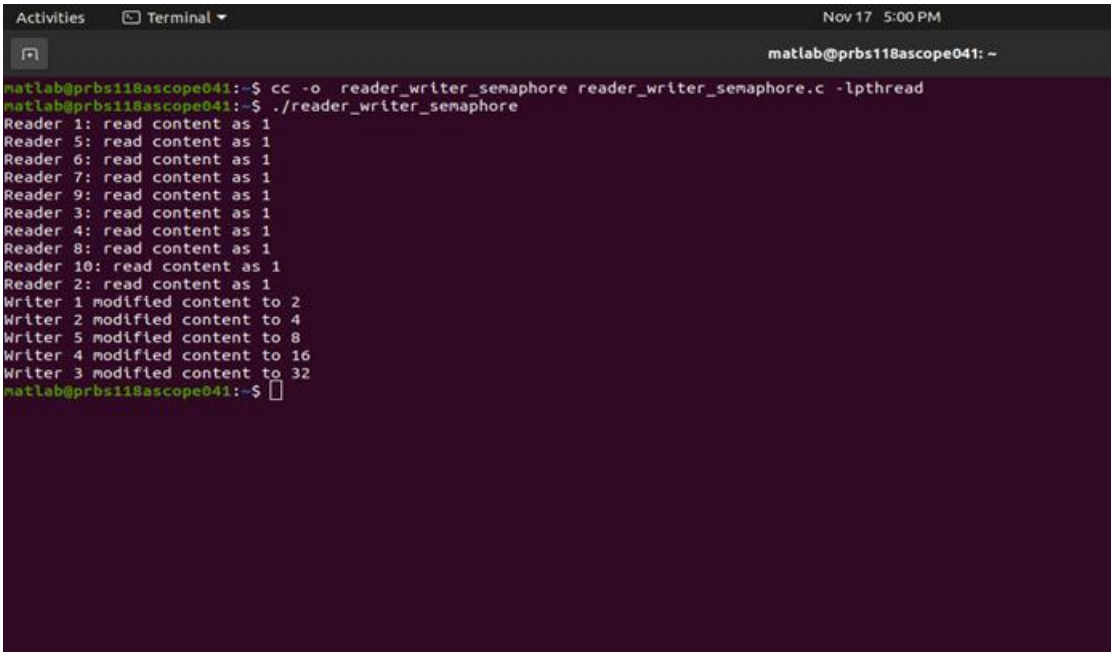
```
#include<pthread.h>
#include<semaphore.h>
#include<stdio.h>
 sem_t wrt;
pthread_mutex_t mutex;
int cnt = 1;
int numreader = 0;

void *writer(void *wno)
{
   sem_wait(&wrt);
   cnt = cnt*2;
   printf("Writer %d modified content to %d\n",(*((int *)wno)),cnt);
   sem_post(&wrt);

}
void *reader(void *rno)
{
   pthread_mutex_lock(&mutex);
   numreader++;
   if(numreader == 1) {
      sem_wait(&wrt);  }
   pthread_mutex_unlock(&mutex);
    printf("Reader %d: read content as %d\n",*((int *)rno),cnt);
```

```
   pthread_mutex_lock(&mutex);
   numreader--;
   if(numreader == 0) {
      sem_post(&wrt);
    }
   pthread_mutex_unlock(&mutex);
}
int main()
{
   pthread_t read[10],write[5];
   pthread_mutex_init(&mutex, NULL);
   sem_init(&wrt,0,1);
   int a[10] = {1,2,3,4,5,6,7,8,9,10};
https:
   for(int i = 0; i < 10; i++) {
      pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);
   }
   for(int i = 0; i < 5; i++) {
      pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);
   }
   for(int i = 0; i < 10; i++) {
      pthread_join(read[i], NULL);
   }
   for(int i = 0; i < 5; i++) {
      pthread_join(write[i], NULL);
   }
 pthread_mutex_destroy(&mutex);
   sem_destroy(&wrt);

   return 0;

}
```


Activities
Code::Blocks IDE
Nov 17 4:53 PM
reader_writer_semaphore.c - Code::Blocks 20.03
File Edit View Search Project Build Debug Tools Plugins Settings Help
<global>
reader(void* rno) : void
Projects
Workspace
reader_writer_semaphore.c
#include<pthread.h>
#include<semaphore.h>
#include<stdio.h>
sem_t wrt;
pthread_mutex_t mutex;
int cnt = 1;
int numreader = 0;
void *writer(void *wno)
{
sem_wait(&wrt);
cnt = cnt*2;
printf("Writer %d modified content to %d\n",(*((int *)wno)),cnt);
sem_post(&wrt);
}
void *reader(void *rno)
{
pthread_mutex_lock(&mutex);
numreader++;
if(numreader == 1) {
sem_wait(&wrt); }
pthread_mutex_unlock(&mutex);
printf("Reader %d: read content as %d\n",*((int *)rno),cnt);
pthread_mutex_lock(&mutex);
numreader--;
if(numreader == 0) {
sem_post(&wrt);
}
pthread_mutex_unlock(&mutex);
}
int main()
{
pthread_t read[10],write[5];
pthread_mutex_init(&mutex, NULL);
/home/matlab/Downloads/reader_writer_semaphore.c
C/C++
Unix (LF)
UTF-8
Line 24, Col 44, Pos 511
Insert
Read/Wri... default


Activities
Code::Blocks IDE
Nov 17 4:53 PM
reader_writer_semaphore.c - Code::Blocks 20.03
File Edit View Search Project Build Debug Tools Plugins Settings Help
<global>
reader(void* rno) : void
Projects
Workspace
reader_writer_semaphore.c
if(numreader == 0) {
sem_post(&wrt);
}
pthread_mutex_unlock(&mutex);
}
int main()
{
pthread_t read[10],write[5];
pthread_mutex_init(&mutex, NULL);
sem_init(&wrt,0,1);
int a[10] = {1,2,3,4,5,6,7,8,9,10};
https:
for(int i = 0; i < 10; i++) {
pthread_create(&read[i], NULL, (void *)reader, (void *)&a[i]);
}
for(int i = 0; i < 5; i++) {
pthread_create(&write[i], NULL, (void *)writer, (void *)&a[i]);
}
for(int i = 0; i < 10; i++) {
pthread_join(read[i], NULL);
}
for(int i = 0; i < 5; i++) {
pthread_join(write[i], NULL);
}
pthread_mutex_destroy(&mutex);
sem_destroy(&wrt);
return 0;
}
/home/matlab/Downloads/reader_writer_semaphore.c
C/C++
Unix (LF)
UTF-8
Line 24, Col 44, Pos 511
Insert
Read/Wri... default

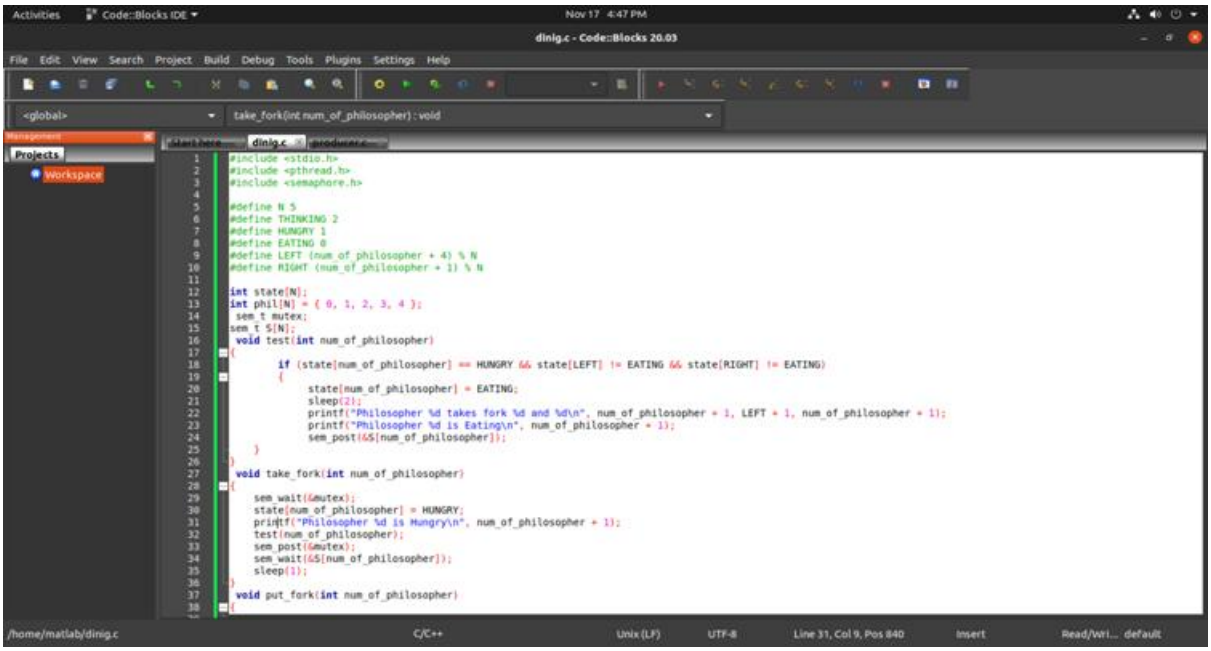
Output:-

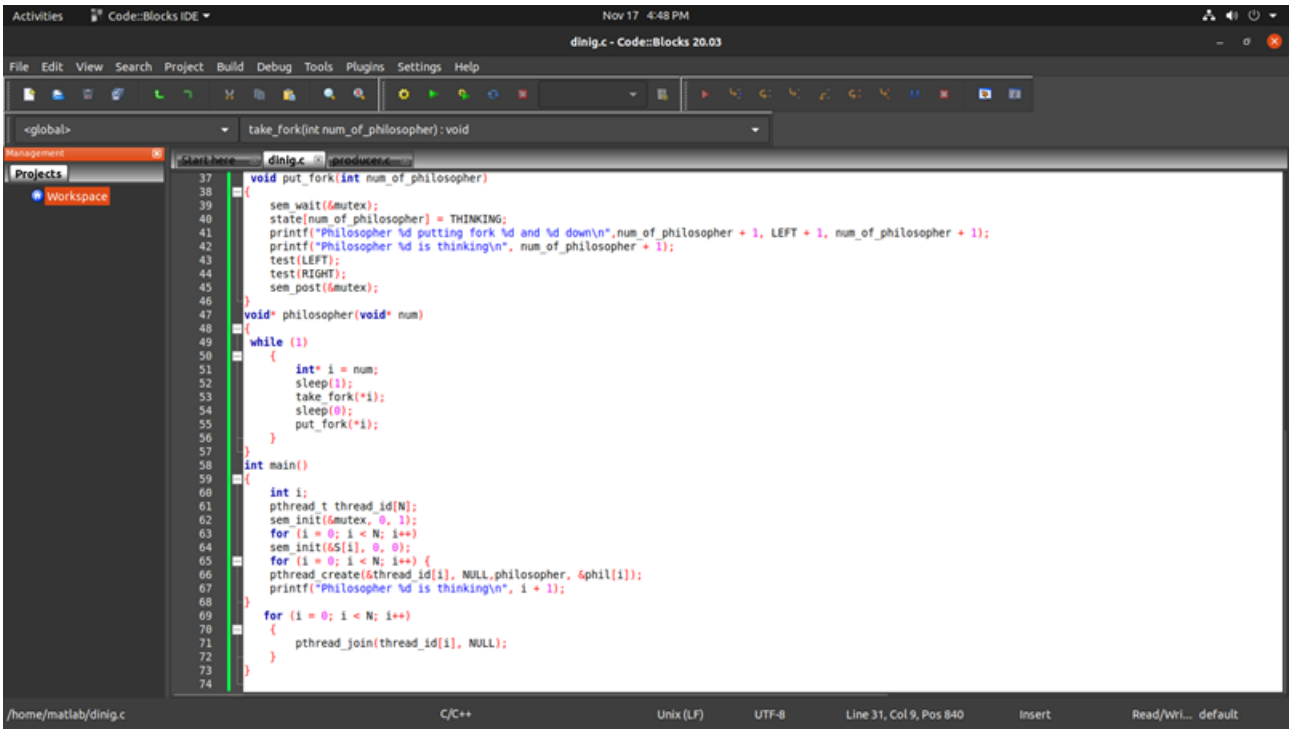

Activities
Terminal
Nov 17 5:00 PM
matlab@prbs118ascope041: ~
matlab@prbs118ascope041:~$ cc -o reader_writer_semaphore reader_writer_semaphore.c -lpthread
matlab@prbs118ascope041:~$ ./reader_writer_semaphore
Reader 1: read content as 1
Reader 5: read content as 1
Reader 6: read content as 1
Reader 7: read content as 1
Reader 9: read content as 1
Reader 3: read content as 1
Reader 4: read content as 1
Reader 8: read content as 1
Reader 10: read content as 1
Reader 2: read content as 1
Writer 1 modified content to 2
Writer 2 modified content to 4
Writer 5 modified content to 8
Writer 4 modified content to 16
Writer 3 modified content to 32
matlab@prbs118ascope041:~$

3.Implement the deadlock-free solution to Dining philosopher problem using Semaphore.

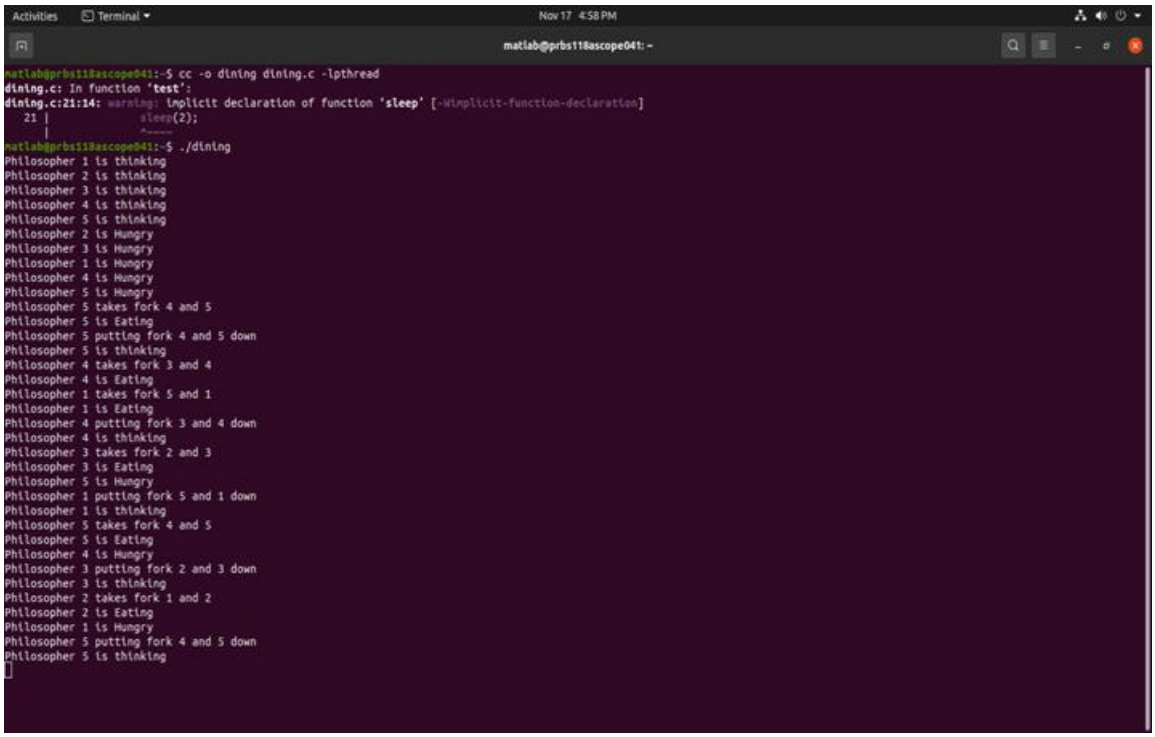
```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

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

int state[N];
int phil[N] = { 0, 1, 2, 3, 4 };
 sem_t mutex;
sem_t S[N];
 void test(int num_of_philosopher)
{
      if (state[num_of_philosopher] == HUNGRY && state[LEFT] != EATING && state[RIGHT] !=
EATING)
      {
          state[num_of_philosopher] = EATING;
          sleep(2);
          printf("Philosopher %d takes fork %d and %d\n", num_of_philosopher + 1, LEFT + 1,
num_of_philosopher + 1);
          printf("Philosopher %d is Eating\n", num_of_philosopher + 1);
          sem_post(&S[num_of_philosopher]);
   }
}
 void take_fork(int num_of_philosopher)
{
   sem_wait(&mutex);
   state[num_of_philosopher] = HUNGRY;
   printf("Philosopher %d is Hungry\n", num_of_philosopher + 1);
   test(num_of_philosopher);
   sem_post(&mutex);
   sem_wait(&S[num_of_philosopher]);
   sleep(1);
}
 void put_fork(int num_of_philosopher)
{
   sem_wait(&mutex);
   state[num_of_philosopher] = THINKING;
   printf("Philosopher %d putting fork %d and %d down\n",num_of_philosopher + 1, LEFT + 1,
num_of_philosopher + 1);
   printf("Philosopher %d is thinking\n", num_of_philosopher + 1);
   test(LEFT);
   test(RIGHT);
   sem_post(&mutex);
}
```

```
void* philosopher(void* num)
{
 while (1)
   {
      int* i = num;
      sleep(1);
      take_fork(*i);
      sleep(0);
       put_fork(*i);
   }
}
int main()
{
   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,philosopher, &phil[i]);
   printf("Philosopher %d is thinking\n", i + 1);
}
  for (i = 0; i < N; i++)
   {
      pthread_join(thread_id[i], NULL);
   }
}
```




Activities
Code::Blocks IDE
Nov 17 4:48 PM
dinig.c - Code::Blocks 20.03
File Edit View Search Project Build Debug Tools Plugins Settings Help
<global>
take_fork(int num_of_philosopher) : void
Management
Projects
Workspace
dinig.c
producer.c
void put_fork(int num_of_philosopher)
{
sem_wait(&mutex);
state[num_of_philosopher] = THINKING;
printf("Philosopher %d putting fork %d and %d down\n",num_of_philosopher + 1, LEFT + 1, num_of_philosopher + 1);
printf("Philosopher %d is thinking\n", num_of_philosopher + 1);
test(LEFT);
test(RIGHT);
sem_post(&mutex);
}
void* philosopher(void* num)
{
while (1)
{
int* i = num;
sleep(1);
take_fork(*i);
sleep(0);
put_fork(*i);
}
}
int main()
{
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,philosopher, &phil[i]);
printf("Philosopher %d is thinking\n", i + 1);
}
for (i = 0; i < N; i++)
{
pthread_join(thread_id[i], NULL);
}
}
/home/matlab/dinig.c
C/C++
Unix (LF)
UTF-8
Line 31, Col 9, Pos 840
Insert
Read/Wri... default

Output:-

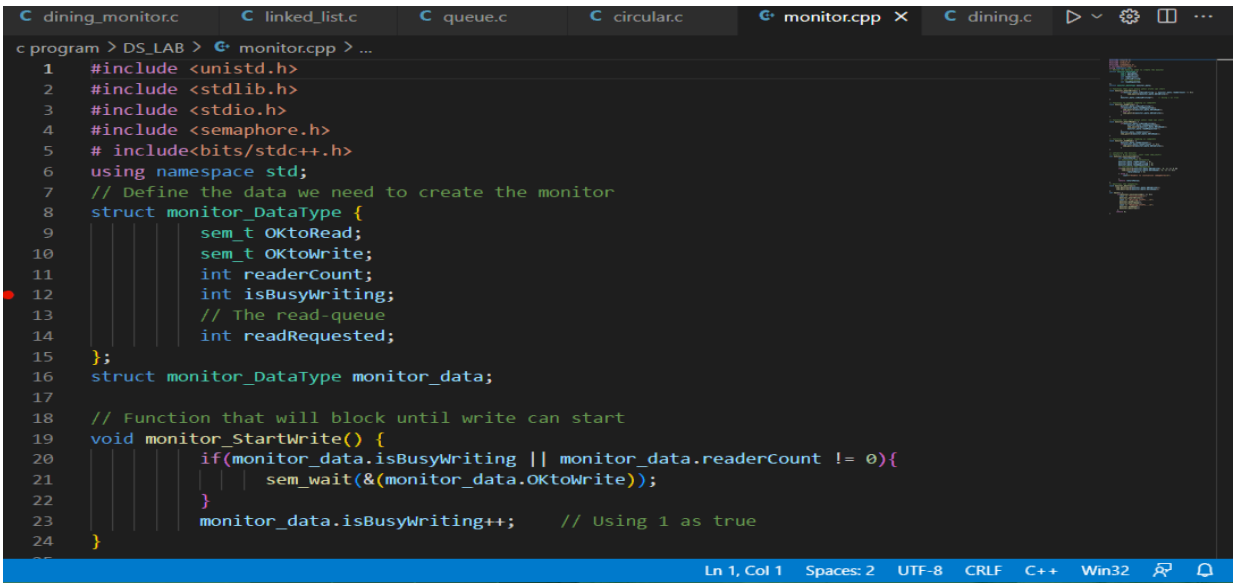

Activities
Terminal
Nov 17 4:58 PM
matlab@prbs118ascope041: ~
matlab@prbs118ascope041:~$ cc -o dining dining.c -lpthread
dining.c: In function 'test':
dining.c:21:14: warning: implicit declaration of function 'sleep' [-Wimplicit-function-declaration]
21 | sleep(2);
| ^~~~~
matlab@prbs118ascope041:~$ ./dining
Philosopher 1 is thinking
Philosopher 2 is thinking
Philosopher 3 is thinking
Philosopher 4 is thinking
Philosopher 5 is thinking
Philosopher 2 is Hungry
Philosopher 3 is Hungry
Philosopher 1 is Hungry
Philosopher 4 is Hungry
Philosopher 5 is Hungry
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking
Philosopher 4 takes fork 3 and 4
Philosopher 4 is Eating
Philosopher 1 takes fork 5 and 1
Philosopher 1 is Eating
Philosopher 4 putting fork 3 and 4 down
Philosopher 4 is thinking
Philosopher 3 takes fork 2 and 3
Philosopher 3 is Eating
Philosopher 5 is Hungry
Philosopher 1 putting fork 5 and 1 down
Philosopher 1 is thinking
Philosopher 5 takes fork 4 and 5
Philosopher 5 is Eating
Philosopher 4 is Hungry
Philosopher 3 putting fork 2 and 3 down
Philosopher 3 is thinking
Philosopher 2 takes fork 1 and 2
Philosopher 2 is Eating
Philosopher 1 is Hungry
Philosopher 5 putting fork 4 and 5 down
Philosopher 5 is thinking

4. Reader writer problem using monitors.

Code:-

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <semaphore.h>
# include<bits/stdc++.h>
using namespace std;
// Define the data we need to create the monitor
struct monitor_DataType {
          sem_t OKtoRead;
          sem_t OKtoWrite;
          int readerCount;
          int isBusyWriting;
          // The read-queue
          int readRequested;
};
struct monitor_DataType monitor_data;

// Function that will block until write can start
void monitor_StartWrite() {
          if(monitor_data.isBusyWriting || monitor_data.readerCount != 0){
                sem_wait(&(monitor_data.OKtoWrite));
          }
          monitor_data.isBusyWriting++;    // Using 1 as true
}

// Function to signal reading is complete
void monitor_EndWrite() {
          monitor_data.isBusyWriting--;
          if(monitor_data.readRequested){
            sem_post(&(monitor_data.OKtoRead));
          } else {
            sem_post(&(monitor_data.OKtoWrite));
          }
}

// Function that will block until read can start
void monitor_StartRead() {
          if(monitor_data.isBusyWriting){
                monitor_data.readRequested++;
                sem_wait(&(monitor_data.OKtoRead));
                monitor_data.readRequested--;
          }
          monitor_data.readerCount++;
          sem_post(&(monitor_data.OKtoRead));
}

// Function to signal reading is complete
void monitor_EndRead() {
          monitor_data.readerCount--;
          if(monitor_data.readerCount == 0){
            sem_post(&(monitor_data.OKtoWrite));
          }
```

```
}

// intialize the monitor
// return's 0 on success, just like sem_init()
int monitor_Initialized(){
        int returnValue = 1;
        // Initialize the structure
        monitor_data.readerCount = 0;
        monitor_data.isBusyWriting = 0;
        monitor_data.readRequested = 0;
        // initialize the semaphores
        if(sem_init(&(monitor_data.OKtoWrite), 0, 1) == 0 &&
           sem_init(&(monitor_data.OKtoRead), 0, 1) == 0){
                returnValue = 0;
        } else {
            cout<<"Unable to initialize semaphores\n";

        }
        return returnValue;
}
// Destroys the semphores.
void monitor_Destroy(){
      sem_destroy(&(monitor_data.OKtoWrite));
      sem_destroy(&(monitor_data.OKtoRead));
}
int main() {
       if(monitor_Initialized() == 0){
         cout << "Initialized\n";
         monitor_StartWrite();
         cout << "Writing stuffs...\n";
         monitor_EndWrite();
         monitor_StartRead();
         cout << "Reading stuffs...\n";
         monitor_EndRead();
         monitor_Destroy();
       }
      return 0;
}
```



```
}

// Function to signal reading is complete
void monitor_EndWrite() {
        monitor_data.isBusyWriting--;
        if(monitor_data.readRequested){
          sem_post(&(monitor_data.OKtoRead));
        } else {
          sem_post(&(monitor_data.OKtoWrite));
        }
}

// Function that will block until read can start
void monitor_StartRead() {
        if(monitor_data.isBusyWriting){
            monitor_data.readRequested++;
            sem_wait(&(monitor_data.OKtoRead));
            monitor_data.readRequested--;
        }
        monitor_data.readerCount++;
        sem_post(&(monitor_data.OKtoRead));
}
```

c program > DS_LAB > monitor.cpp > monitor_EndRead()

```
void monitor_EndRead() {
        monitor_data.readerCount--;
        if(monitor_data.readerCount == 0){
          sem_post(&(monitor_data.OKtoWrite));
        }
}

// intialize the monitor
// return's 0 on success, just like sem_init()
int monitor_Initialized(){
        int returnValue = 1;
        // Initialize the structure
        monitor_data.readerCount = 0;
        monitor_data.isBusyWriting = 0;
        monitor_data.readRequested = 0;
        // initialize the semaphores
        if(sem_init(&(monitor_data.OKtoWrite), 0, 1) == 0 &&
           sem_init(&(monitor_data.OKtoRead), 0, 1) == 0){
                returnValue = 0;
        } else {
           cout<<"Unable to initialize semaphores\n";

        }
        return returnValue;
```

Ln 48, Col 1 Spaces: 2 UTF-8 CRLF C++ Win32

c program > DS_LAB > monitor.cpp > ...

```
69
70            }
71            return returnValue;
72    }
73    // Destroys the semphores.
74    void monitor_Destroy(){
75          sem_destroy(&(monitor_data.OKtoWrite));
76          sem_destroy(&(monitor_data.OKtoRead));
77    }
78    int main() {
79          if(monitor_Initialized() == 0){
80            cout << "Initialized\n";
81            monitor_StartWrite();
82            cout << "Writing stuffs...\n";
83            monitor_EndWrite();
84            monitor_StartRead();
85            cout << "Reading stuffs...\n";
86            monitor_EndRead();
87            monitor_Destroy();
88          }
89          return 0;
90    }
91
```

Output:-

```
student@sh-4.4-desktop:~$ gcc rw1.c -lpthread
student@sh-4.4-desktop:~$ ./a.out
------------------------
reader-0 is reading
updated value : 5

writer-0 is writing
------------------------
reader-1 is reading
updated value : 10
------------------------
reader-2 is reading
updated value : 10
------------------------
reader-3 is reading
updated value : 10

writer-3 is writing
------------------------
reader-4 is reading
```