OS Lab Assignment 6

## Name: Keshav Garg

## ID: 2018UCP1674

GitHub repo link: [**Here**](https://github.com/gargk747/OS-Lab/tree/master/Assignment%206)

1. **Producer-Consumer:**

Two Process-

**Producer.c**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <time.h>

#include <stdbool.h>

#define BUFFERSIZE 10

typedef struct {

bool read;

int buffer[BUFFERSIZE];

} shm\_struct;

int main(int argc, char\* argv[])

{

int shmid;

key\_t key;

shm\_struct \* shm;

int rand\_num;

FILE \*fp; //file object

//create a producer txt file

fp = fopen("producer.txt", "w+");

srand(time(NULL)); // should only be called once

if ((shmid = shmget(key, sizeof(shm\_struct), IPC\_CREAT | 0666)) < 0) {

perror("shmget");

exit(1);

}

if ((shm = shmat(shmid, NULL, 0)) == (shm\_struct \*) -1) {

perror("shmat");

exit(1);

}

shm->read = true;

//pushing values

for (int j=0; j<100; j++) //100x 10 = 1000 elements

{

while(shm->read); //wait for consumer to read

if (!shm->read) //if consumer has read, produce.

{

for (int i=0; i<10; i++)

{

rand\_num = rand() % 100; //generate random numbers

shm->buffer[i] = rand\_num;//store to array

fprintf(fp, "%d \n",rand\_num);//store to file

}

shm->read = true;// allow consumer to read

}

}

//destroy shared memory

shmdt (shm);

shmctl (shmid, IPC\_RMID, NULL);

//close file

fclose(fp);

exit(0);

}

**Consumer.c**

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <stdbool.h>

#define BUFFERSIZE 10

typedef struct {

bool read;//if the consumer should read or not

int buffer[BUFFERSIZE];

} shm\_struct;

int main(int argc, char\* argv[])

{

int shmid;

key\_t key;

shm\_struct \*shm;

FILE \*fp; //file object

//open file

fp = fopen("consumer.txt", "w+");

/\*

\* We need to get the segment named

\* "5678", created by the server.

\*/

key = 5671;

/\*

\* Locate the segment.

\*/

if ((shmid = shmget(key, sizeof(shm\_struct), 0666)) < 0) {

perror("shmget");

exit(1);

}

/\*

\* Now we attach the segment to our data space.

\*/

if ((shm = shmat(shmid, NULL, 0)) == (shm\_struct \*) -1) {

perror("shmat");

exit(1);

}

/\*

\* Now read what the server put in the memory.

\*/

shm->read=false;// have read now

for (int j=0; j<100; j++)//100 x 10 = 1000 elements

{

while (!shm->read); //wait for it to producer to write

if (shm->read) //read now

{

for (int i=0; i<10; i++){

fprintf(fp, "%d \n",shm->buffer[i]);//save to file

}

shm->read = false;// have read now

}

}

/\*

\* cleanup: detach shm

\*/

shmdt (shm);

//close file

fclose(fp);

exit(0);

}

Two Thread-

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <time.h>

#include <stdbool.h>

// Let us create a global queue to change it in threads

int mesg\_text[10];//queue size

bool read = false;// if the consumer should read or not

// The function to be executed by producer thread

void \*producer()

{

int queue\_index\_p = 0;

int rand\_num;

int total\_num\_sent = 0;

FILE \*fp; //file object- producer

//create a producer txt file

fp = fopen("producer.txt", "w+");

while(total\_num\_sent!=1000)//for all 1000 values

{

while(read);

// random number generate

rand\_num = rand() % 100;

//storing number in array of lenght 10

mesg\_text[queue\_index\_p] = rand\_num;

fprintf(fp, "%d \n",rand\_num);//storing in file

queue\_index\_p++;

total\_num\_sent++;

if (queue\_index\_p==10)

{

queue\_index\_p =0;

read = true;//consumer can read now

}

}

//close file

fclose(fp);

}

// The function to be executed by consumer thread

void \*consumer()

{

int queue\_index\_c = 0;

int total\_num\_recvd = 0;

FILE \*fc; //file object- consumer

//create a consumer txt file

fc = fopen("consumer.txt", "w+");

while(total\_num\_recvd<1000)

{

while(!read);

//storing number in file

fprintf(fc, "%d \n",mesg\_text[queue\_index\_c]);

queue\_index\_c++;

total\_num\_recvd++;

if (queue\_index\_c==10)

{

queue\_index\_c =0;

read = false;// the producer can produce now

}

}

//close file

fclose(fc);

}

int main()

{

//create thread id

pthread\_t tid;

srand(time(NULL)); // should only be called once

//create two threads

//producer thread

pthread\_create(&tid, NULL, producer, NULL);

//consumer thread

pthread\_create(&tid, NULL, consumer, NULL);

//destroy all threads

pthread\_exit(NULL);

return 0;

}

1. **Reader-Writer:**

RW1-

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#include<stdlib.h>

#include<unistd.h>

sem\_t lock,resource;

int arr[10];

int count = 0;

void\* writer(void\* num){

int k = (int)num;

sem\_wait(&resource);

printf("Writer %d is writing\n",k);

sleep(2);

for(int i=0;i<10;i++){

arr[i] = rand()%50;

}

printf("Reader count = %d\n",count);

sem\_post(&resource);

}

void\* reader(void\* num){

int k = (int)num;

sem\_wait(&lock);

count++;

if(count == 1){

sem\_wait(&resource);

}

printf("Reader %d is reading\n",k);

printf("Reader count = %d\n",count);

sem\_post(&lock);

//cs

sleep(2);

sem\_wait(&lock);

count--;

if(count==0){

sem\_post(&resource);

}

sem\_post(&lock);

}

int main(){

sem\_init(&resource,0,1);

sem\_init(&lock,0,1);

pthread\_t w[5],r[5];

for(int i=0;i<5;i++){

pthread\_create(&w[i],NULL,&writer,(void\*)((int)(i+1)));

pthread\_create(&r[i],NULL,&reader,(void\*)((int)(i+1)));

}

for(int i=0;i<5;i++){

pthread\_join(w[i],NULL);

pthread\_join(r[i],NULL);

}

return 0;

}

**Output:**

****

RW2-

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#include<stdlib.h>

#include<unistd.h>

sem\_t rlock,wlock,readTry,resource;

int rcount=0,wcount=0;

void\* writer(void\* num){

while(1){

int k = (int)num;

sem\_wait(&wlock);

wcount++;

if(wcount == 1){

sem\_wait(&readTry);

}

sem\_post(&wlock);

sem\_wait(&resource);

printf("Writer %d is writing\n",k);

printf("Readers = %d and Writers = %d\n",rcount,wcount);

sem\_post(&resource);

sem\_wait(&wlock);

wcount--;

if(wcount==0){

sem\_post(&readTry);

}

sem\_post(&wlock);

sleep(1);

}

}

void\* reader(void\* num){

while(1){

int k = (int)num;

sem\_wait(&readTry);

sem\_wait(&rlock);

rcount++;

if(rcount==1){

sem\_wait(&resource);

}

sem\_post(&rlock);

sem\_post(&readTry);

printf("Reader %d is reading\n",k);

printf("Readers = %d and Writers = %d\n",rcount,wcount);

sem\_wait(&rlock);

rcount--;

if(rcount==0){

sem\_post(&resource);

}

sem\_post(&rlock);

sleep(1);

}

}

int main(){

sem\_init(&rlock,0,1);

sem\_init(&wlock,0,1);

sem\_init(&resource,0,1);

sem\_init(&readTry,0,1);

pthread\_t w[5],r[5];

for(int i=0;i<5;i++){

pthread\_create(&w[i],NULL,&writer,(void\*)((int)(i+1)));

pthread\_create(&r[i],NULL,&reader,(void\*)((int)(i+1)));

}

for(int i=0;i<5;i++){

pthread\_join(w[i],NULL);

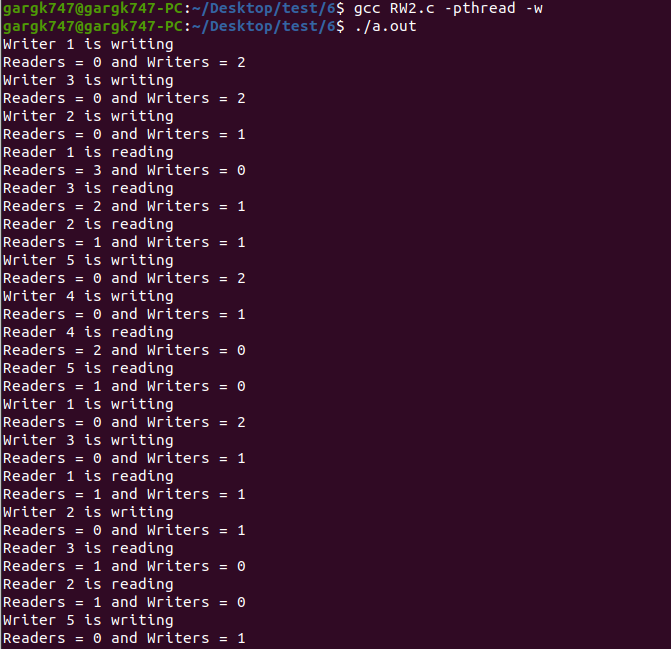
pthread\_join(r[i],NULL);

}

return 0;

}

**Output:**



RW3-

#include<stdio.h>

#include<pthread.h>

#include<semaphore.h>

#include<stdlib.h>

#include<unistd.h>

sem\_t resource,readCount,service;

int count=0;

void\* writer(void\* num){

while(1){

int k = (int)num;

sem\_wait(&service);

sem\_wait(&resource);

sem\_post(&service);

printf("Writer %d is writing\n",k);

printf("Number of readers = %d\n",count);

sem\_post(&resource);

sleep(1);

}

}

void\* reader(void\* num){

while(1){

int k = (int)num;

sem\_wait(&service);

sem\_wait(&readCount);

if(count==0){

sem\_wait(&resource);

}

count++;

sem\_post(&service);

sem\_post(&readCount);

printf("Reader %d is reading\n",k);

printf("Number of readers = %d\n",count);

sem\_wait(&readCount);

count--;

if(count==0){

sem\_post(&resource);

}

sem\_post(&readCount);

sleep(1);

}

}

int main(){

sem\_init(&resource,0,1);

sem\_init(&readCount,0,1);

sem\_init(&service,0,1);

pthread\_t w[5],r[5];

for(int i=0;i<5;i++){

pthread\_create(&w[i],NULL,&writer,(void\*)((int)(i+1)));

pthread\_create(&r[i],NULL,&reader,(void\*)((int)(i+1)));

}

for(int i=0;i<5;i++){

pthread\_join(w[i],NULL);

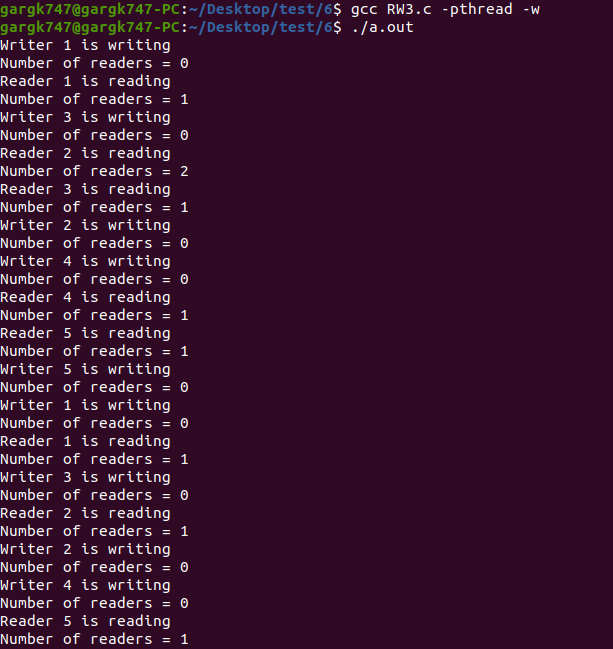
pthread\_join(r[i],NULL);

}

return 0;

}

**Output:**



1. **Dining Philosopher:**

#include <pthread.h>

#include <stdio.h>

#include <unistd.h>

#include <time.h>

#include <stdlib.h>

#define NUMBER\_OF\_PHILOSOPHERS 5

void \*philosopher(void \*);

void think(int);

void pickUp(int);

void eat(int);

void putDown(int);

pthread\_mutex\_t chopsticks[NUMBER\_OF\_PHILOSOPHERS];

pthread\_t philosophers[NUMBER\_OF\_PHILOSOPHERS];

pthread\_attr\_t attributes[NUMBER\_OF\_PHILOSOPHERS];

int main() {

int i;

srand(time(NULL));

for (i = 0; i < NUMBER\_OF\_PHILOSOPHERS; ++i) {

pthread\_mutex\_init(&chopsticks[i], NULL);

}

for (i = 0; i < NUMBER\_OF\_PHILOSOPHERS; ++i) {

pthread\_attr\_init(&attributes[i]);

}

for (i = 0; i < NUMBER\_OF\_PHILOSOPHERS; ++i) {

pthread\_create(&philosophers[i], &attributes[i], philosopher, (void \*)(i));

}

for (i = 0; i < NUMBER\_OF\_PHILOSOPHERS; ++i) {

pthread\_join(philosophers[i], NULL);

}

return 0;

}

void \*philosopher(void \*philosopherNumber) {

while (1) {

think(philosopherNumber);

pickUp(philosopherNumber);

eat(philosopherNumber);

putDown(philosopherNumber);

}

}

void think(int philosopherNumber) {

int sleepTime = rand() % 3 + 1;

printf("Philosopher %d will think for %d seconds\n", philosopherNumber, sleepTime);

sleep(sleepTime);

}

void pickUp(int philosopherNumber) {

int right = (philosopherNumber + 1) % NUMBER\_OF\_PHILOSOPHERS;

int left = (philosopherNumber + NUMBER\_OF\_PHILOSOPHERS) % NUMBER\_OF\_PHILOSOPHERS;

if (philosopherNumber & 1) {

printf("Philosopher %d is waiting to pick up chopstick %d\n", philosopherNumber, right);

pthread\_mutex\_lock(&chopsticks[right]);

printf("Philosopher %d picked up chopstick %d\n", philosopherNumber, right);

printf("Philosopher %d is waiting to pick up chopstick %d\n", philosopherNumber, left);

pthread\_mutex\_lock(&chopsticks[left]);

printf("Philosopher %d picked up chopstick %d\n", philosopherNumber, left);

}

else {

printf("Philosopher %d is waiting to pick up chopstick %d\n", philosopherNumber, left);

pthread\_mutex\_lock(&chopsticks[left]);

printf("Philosopher %d picked up chopstick %d\n", philosopherNumber, left);

printf("Philosopher %d is waiting to pick up chopstick %d\n", philosopherNumber, right);

pthread\_mutex\_lock(&chopsticks[right]);

printf("Philosopher %d picked up chopstick %d\n", philosopherNumber, right);

}

}

void eat(int philosopherNumber) {

int eatTime = rand() % 3 + 1;

printf("Philosopher %d will eat for %d seconds\n", philosopherNumber, eatTime);

sleep(eatTime);

}

void putDown(int philosopherNumber) {

printf("Philosopher %d will will put down her chopsticks\n", philosopherNumber);

pthread\_mutex\_unlock(&chopsticks[(philosopherNumber + 1) % NUMBER\_OF\_PHILOSOPHERS]);

pthread\_mutex\_unlock(&chopsticks[(philosopherNumber + NUMBER\_OF\_PHILOSOPHERS) % NUMBER\_OF\_PHILOSOPHERS]);

}

**Output:**

