Assignment 3:Synchronization Problems

# Name – Akshat Chandrapatle Div- CS-A Roll no.- 45 PRN- 12111449

1. Producer consumer using mutex

Code:

#include <stdio.h>

#include <pthread.h>

#include <stdlib.h>

#define BufferSize 10

int count = 0;

int buffer[BufferSize];

pthread\_mutex\_t mutex;

void \*producer(void \*item){

long int num = (long int)item;

while(1) {

pthread\_mutex\_lock(&mutex);

if (count < BufferSize) {

buffer[count] = rand() % 10;

printf(" \n Producer %ld Produced %d\n", num + 1, buffer[count]);

count++;

}

pthread\_mutex\_unlock(&mutex);

sleep(1);

}

pthread\_exit(NULL);

}

void \*consumer(void \*item){

long int num = (long int)item;

while(1) {

pthread\_mutex\_lock(&mutex);

if (count > 0) {

count--;

printf(" \n Consumer %ld consumed %d\n", num + 1, buffer[count]);

}

pthread\_mutex\_unlock(&mutex);

sleep(1);

}

pthread\_exit(NULL);

}

int main()

{

printf("Hello World");

pthread\_t p[10], c[10];

pthread\_mutex\_init(&mutex, NULL);

unsigned long int i;

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

pthread\_create(&p[i], NULL, producer, (void \*)i);

}

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

pthread\_create(&c[i], NULL, consumer, (void \*)i);

}

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

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

}

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

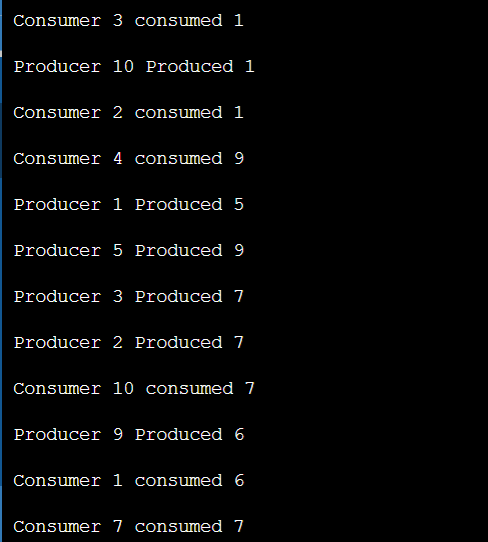
pthread\_join(c[i], NULL);

}

return 0;

}

Output:



1. Producer consumer using semaphore

Code:

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

#define BufferSize 10

sem\_t empty;

sem\_t full;

int count = 0;

int buffer[BufferSize];

void \*producer(void \*item){

long int num = (long int)item;

while (1) {

int item = rand() % 10;

sem\_wait(&empty);

buffer[count]=item;

printf("Producer %ld Produced %d\n", num+1, buffer[count]);

count++;

sem\_post(&full);

sleep(1);

}

pthread\_exit(NULL);

}

void \*consumer(void \*item){

long int num = (long int)item;

while (1) {

sem\_wait(&full);

count--;

printf("Consumer %ld consumed %d\n", num+1, buffer[count]);

sem\_post(&empty);

sleep(1);

}

pthread\_exit(NULL);

}

int main()

{

printf("Hello World");

pthread\_t p[10],c[10];

sem\_init(&empty, 0, BufferSize);

sem\_init(&full, 0, 0);

unsigned long int i;

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

pthread\_create(&p[i], NULL, producer, (void \*)i);

}

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

pthread\_create(&c[i], NULL, consumer, (void \*)i);

}

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

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

}

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

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

}

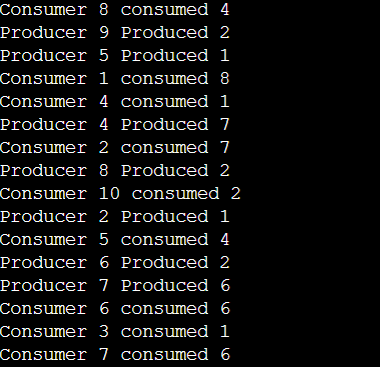
sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

Output:



1. Reader writer using semaphore

Code :

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define BUFFER\_SIZE 10

int buffer[BUFFER\_SIZE];

int reader\_count = 0;

sem\_t read\_sem, write\_sem, mutex;

void \*writer(void \*arg)

{

int data = \*(int\*)arg;

sem\_wait(&write\_sem);

buffer[0] = data;

printf("Writer wrote %d\n", data);

sem\_post(&write\_sem);

pthread\_exit(NULL);

}

void \*reader(void \*arg)

{

int index = \*(int\*)arg;

sem\_wait(&mutex);

reader\_count++;

if (reader\_count == 1)

sem\_wait(&write\_sem);

sem\_post(&mutex);

sem\_wait(&read\_sem);

printf("Reader %d read %d\n", index, buffer[0]);

sem\_post(&read\_sem);

sem\_wait(&mutex);

reader\_count--;

if (reader\_count == 0)

sem\_post(&write\_sem);

sem\_post(&mutex);

pthread\_exit(NULL);

}

int main()

{

pthread\_t writers[3], readers[5];

sem\_init(&read\_sem, 0, 1);

sem\_init(&write\_sem, 0, 1);

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

int i;

for (i = 0; i < 3; i++)

pthread\_create(&writers[i], NULL, writer, &i);

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

pthread\_create(&readers[i], NULL, reader, &i);

for (i = 0; i < 3; i++)

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

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

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

sem\_destroy(&read\_sem);

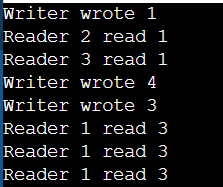
sem\_destroy(&write\_sem);

sem\_destroy(&mutex);

return 0;

}

Output:



1. Reader writer using mutex

Code:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define BUFFER\_SIZE 10

int buffer[BUFFER\_SIZE];

int reader\_count = 0;

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

void \*writer(void \*arg)

{

int data = \*(int\*)arg;

pthread\_mutex\_lock(&mutex);

buffer[0] = data;

printf("Writer wrote %d\n", data);

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

}

void \*reader(void \*arg)

{

int index = \*(int\*)arg;

pthread\_mutex\_lock(&mutex);

reader\_count++;

if (reader\_count == 1)

pthread\_mutex\_lock(&mutex);

pthread\_mutex\_unlock(&mutex);

printf("Reader %d read %d\n", index, buffer[0]);

pthread\_mutex\_lock(&mutex);

reader\_count--;

if (reader\_count == 0)

pthread\_mutex\_unlock(&mutex);

pthread\_mutex\_unlock(&mutex);

pthread\_exit(NULL);

}

int main()

{

pthread\_t writers[3], readers[5];

int i;

for (i = 0; i < 3; i++)

pthread\_create(&writers[i], NULL, writer, &i);

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

pthread\_create(&readers[i], NULL, reader, &i);

for (i = 0; i < 3; i++)

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

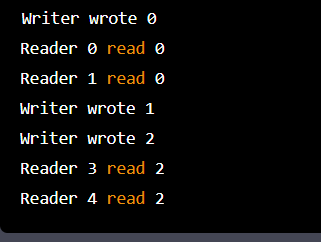
for (i = 0; i < 5; i++)

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

return 0;

}

Output:



1. Dinning Philosopher problem using semaphore

Code:

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define NUM\_PHILOSOPHERS 5

sem\_t chopsticks[NUM\_PHILOSOPHERS];

pthread\_t philosophers[NUM\_PHILOSOPHERS];

void \*philosopher(void \*arg)

{

int id = \*(int\*)arg;

int left = id;

int right = (id + 1) % NUM\_PHILOSOPHERS;

while (1) {

printf("Philosopher %d is thinking\n", id);

sleep(rand() % 10);

printf("Philosopher %d is hungry\n", id);

sem\_wait(&chopsticks[left]);

sem\_wait(&chopsticks[right]);

printf("Philosopher %d is eating\n", id);

sleep(rand() % 10);

sem\_post(&chopsticks[left]);

sem\_post(&chopsticks[right]);

}

}

int main()

{

int i;

for (i = 0; i < NUM\_PHILOSOPHERS; i++)

sem\_init(&chopsticks[i], 0, 1);

for (i = 0; i < NUM\_PHILOSOPHERS; i++)

pthread\_create(&philosophers[i], NULL, philosopher, &i);

for (i = 0; i < NUM\_PHILOSOPHERS; i++)

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

return 0;

}

Output:

