Homework 3 Report

CSC 4320 Operating Systems

Spring 2018

Name: Jimmy Tran

Email: [jtran25@student.gsu.edu](mailto:jtran25@student.gsu.edu)

As a side note regarding my program, I intentionally delayed when the producer and consumer threads started in order to avoid attempts to consume before producing and to avoid getting too much output at once.

1) Source Code

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define TRUE 1

typedef int buffer\_item;

#define BUFFER\_SIZE 8

buffer\_item START\_NUMBER;

int insert\_item(buffer\_item item);

int remove\_item(buffer\_item \*item);

buffer\_item buffer[BUFFER\_SIZE];

pthread\_mutex\_t mutex;

sem\_t empty;

sem\_t full;

int insertPointer = 0, removePointer = 0;

void \*producer(void \*param);

void \*consumer(void \*param);

int insert\_item(buffer\_item item)

{

/\* Implementation of the insert\_item() function \*/

if(insertPointer>=BUFFER\_SIZE) //return to beginning of buffer

insertPointer=0;

if(insertPointer<BUFFER\_SIZE){

buffer[insertPointer]=item;

insertPointer++;

}

else

return -1;

return 0;

}

int remove\_item(buffer\_item \*item)

{

/\* Implementation of the remove\_item function \*/

if(removePointer>=BUFFER\_SIZE) //return to beginning of buffer

removePointer=0;

if(removePointer>=0&&removePointer<BUFFER\_SIZE){

\*item = buffer[removePointer];

removePointer++;

}

else

return -1;

return 0;

}

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

{

/\* 1. Get command line arguments argv[1],argv[2],argv[3],argv[4] \*/

/\* 2. Initialize buffer \*/

/\* 3. Create producer thread(s) \*/

/\* 4. Create consumer thread(s) \*/

/\* 5. Sleep \*/

/\* 6. Exit \*/

int sleepTime, producerThreads, consumerThreads;

int i, j;

if(argc != 5)

{

fprintf(stderr, "Usage: <sleep time> <producer threads> <consumer threads> <start number>\n");

return -1;

}

sleepTime = atoi(argv[1]);

producerThreads = atoi(argv[2]);

consumerThreads = atoi(argv[3]);

START\_NUMBER = atoi(argv[4]);

/\* Initialize the synchronization tools \*/

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

sem\_init(&empty, 0, BUFFER\_SIZE);

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

/\* Create the producer and consumer threads \*/

pthread\_t pro, con;

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

pthread\_create(&pro, NULL, producer, START\_NUMBER); //create producers

}

for(int j=0; j<consumerThreads;j++){

pthread\_create(&con, NULL, consumer, NULL); //create consumers

}

/\* Sleep for user specified time \*/

sleep(sleepTime);

return 0;

}

void \*producer(void \*param)

{

/\* Implementation of the producer thread -- refer to Figure 5.26 on page 256 \*/

buffer\_item item;

while(TRUE){

sleep(1);

item=START\_NUMBER;

sem\_wait(&empty); //lock empty semaphore if !0

pthread\_mutex\_lock(&mutex); //mutex lock

if(insert\_item(item)){

fprintf(stderr, "Insert into buffer failed\n");

}

else{

printf("Producer %u produced %d \n",(unsigned int)pthread\_self(), item);

START\_NUMBER++;

}

pthread\_mutex\_unlock(&mutex); //mutex unlock

sem\_post(&full); //increment full semaphore

}

}

void \*consumer(void \*param)

{

/\* Implementation of the consumer thread -- refer to Figure 5.26 on page 256 \*/

buffer\_item item;

while(TRUE){

sleep(2);

sem\_wait(&full); //lock full semaphore

pthread\_mutex\_lock(&mutex); //mutex lock

if(remove\_item(&item)){

fprintf(stderr, "Removal from buffer failed\n");

}

else

printf("Consumer %u consumed %d \n",(unsigned int)pthread\_self(), item);

pthread\_mutex\_unlock(&mutex); //mutex unlock

sem\_post(&empty); //increments empty semaphore

}

}

2) Screenshots:

Scenario 1: More Producers than Consumers



Scenario 2: More Consumer than Producers



Scenario 3: Equal number of Consumers and Producers

