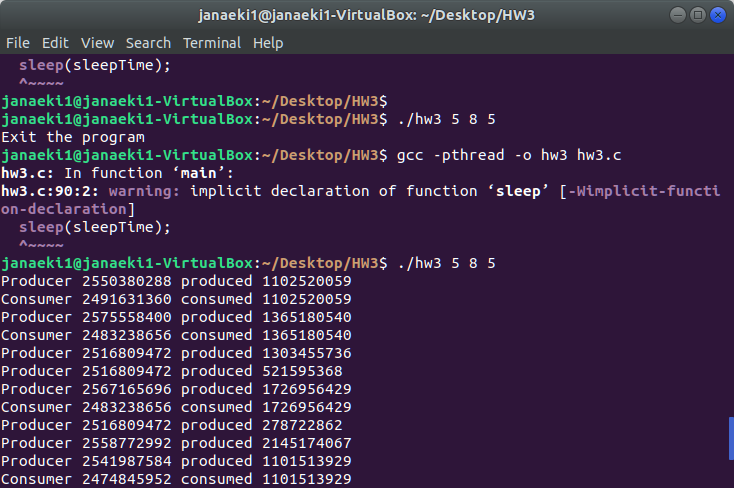
Jerome Anaeki

**Homework 3**

**Screenshot**



**Code**

#include "buffer.h"

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

buffer\_item buffer[BUFFER\_SIZE]; //buffer

int counter; //counter for buffer

sem\_t empty, full; //empty and full semaphores

pthread\_mutex\_t mutex; //mutex lock

pthread\_t pid; //ID for producer threads

pthread\_t cid; //ID for consumer threads

void \*producer(void \*param);

void \*consumer(void \*param);

int insert\_item(buffer\_item item)

{

if(counter < BUFFER\_SIZE) {

buffer[counter] = item; //add item to buffer

counter++; //increment counter

}

return 0;

}

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

{

if(counter > 0) {

\*item = buffer[(counter-1)]; //remove item from buffer

counter--; //decrement counter

}

return 0;

}

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

{

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

int sleepTime, prod, cons;

int i;

if (argc != 4) {

fprintf(stderr,"Usage: ./hw3 <sleep time> <# of producer threads> <# of consumer threads>\n");

return -1;

}

sleepTime = atoi(argv[1]); //user input for sleep time

prod = atoi(argv[2]); //user input for number of producer threads

cons = atoi(argv[3]); //user input for number of consumer threads

if (prod < 2) {

fprintf(stderr,"# of producer threads must be larger than 1\n");

return -1;

}

if (cons < 2) {

fprintf(stderr,"# of consumer threads must be larger than 1\n");

return -1;

}

/\* Initialize buffer related synchronization tools \*/

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

sem\_init(&full, 0, 0); //create full semaphore

pthread\_mutex\_init(&mutex, NULL); //create mutex lock

counter = 0; //initialize counter for buffer

/\* Create producer threads based on the command line input \*/

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

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

}

/\* Create consumer threads based on the command line input \*/

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

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

}

/\*Join threads\*/

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

pthread\_join(pid,NULL);

}

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

pthread\_join(cid,NULL);

}

/\* Sleep for user specified time based on the command line input \*/

sleep(sleepTime);

return 0;

}

void \*producer(void \*param)

{

/\* producer thread that calls insert\_item() \*/

buffer\_item produced\_number;

while(1) {

/\* sleep for a random period of time \*/

int random = rand() / 10000000;

sleep(random);

/\* generate a random number \*/

produced\_number = rand();

sem\_wait(&empty); //acquire empty semaphore

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

if (insert\_item(produced\_number)){

fprintf(stderr, "report error condition");

}

else{

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

}

pthread\_mutex\_unlock(&mutex); //release mutex lock

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

}

}

void \*consumer(void \*param)

{

/\* consumer thread that calls remove\_item() \*/

buffer\_item consumed\_number;

while(1) {

/\* sleep for a random period of time \*/

int random = rand() / 10000000;

sleep(random);

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

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

if (remove\_item(&consumed\_number)){

fprintf(stderr, "report error condition");

}

else{

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

}

pthread\_mutex\_unlock(&mutex); //release mutex lock

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

}

}