**Name: Debjyoti Sarkar Sumonta**

**Student ID: 22341019**

**CSE321 Section 5**

**Task 1**

#include <pthread.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#define MAX 10

#define BUFLEN 6

#define NUMTHREAD 2

void \*consumer(void \*id);

void \*producer(void \*id);

char buffer[BUFLEN + 1];

char source[BUFLEN + 1];

int pCount = 0;

int cCount = 0;

int buflen;

// Initialize mutex and condition variables

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

pthread\_cond\_t empty = PTHREAD\_COND\_INITIALIZER;

pthread\_cond\_t full = PTHREAD\_COND\_INITIALIZER;

int thread\_id[NUMTHREAD] = {0, 1};

int src\_index = 0;

int count = 0; // for accessing values

int main() {

pthread\_t thread[NUMTHREAD];

strcpy(source, "abcdef");

buflen = strlen(source);

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

int \*a = malloc(sizeof(int));

\*a = thread\_id[i];

if (i == 0) {

pthread\_create(&thread[i], NULL, &producer, a);

} else {

pthread\_create(&thread[i], NULL, &consumer, a);

}

}

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

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

}

}

// Producer

void \*producer(void \*id) {

int thread\_id = \*(int \*)id;

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

pthread\_mutex\_lock(&count\_mutex);

while (count >= BUFLEN) {

pthread\_cond\_wait(&full, &count\_mutex);

}

char x = source[src\_index % BUFLEN];

buffer[count] = x;

src\_index++;

count++;

pCount++;

printf("%d produced %c by Thread %d\n", i, x, thread\_id);

pthread\_cond\_signal(&empty);

pthread\_mutex\_unlock(&count\_mutex);

}

pthread\_exit(NULL);

}

// Consumer

void \*consumer(void \*id) {

int thread\_id = \*(int \*)id;

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

pthread\_mutex\_lock(&count\_mutex);

while (count <= 0) {

pthread\_cond\_wait(&empty, &count\_mutex);

}

char y = buffer[count - 1];

count--;

cCount++;

printf("%d consumed %c by Thread %d\n", i, y, thread\_id);

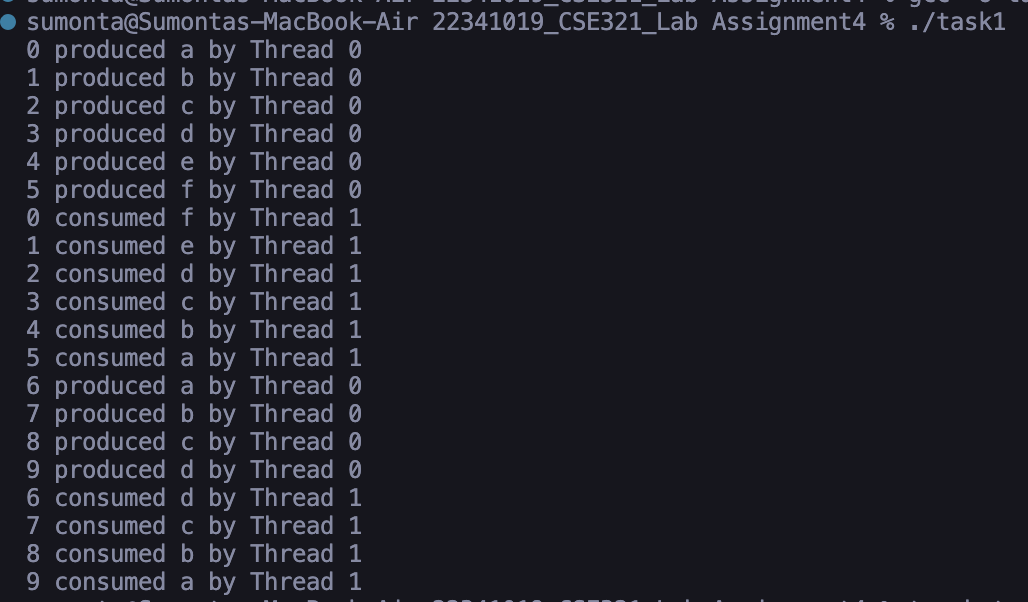
pthread\_cond\_signal(&full);

pthread\_mutex\_unlock(&count\_mutex);

}

pthread\_exit(NULL);

}



**Task 2**

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

#define MaxCrops 5

#define warehouseSize 5

sem\_t empty;

sem\_t full;

int c\_index = 0;

int w\_index = 0;

char crops[warehouseSize] = {'R', 'W', 'P', 'S', 'M'};

char warehouse[warehouseSize] = {'N', 'N', 'N', 'N', 'N'};

pthread\_mutex\_t mutex;

void \*Farmer(void \*arg) {

int thread\_id = \*(int \*)arg;

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

sem\_wait(&empty);

pthread\_mutex\_lock(&mutex);

char x = crops[c\_index % warehouseSize];

warehouse[w\_index % warehouseSize] = x;

printf("Farmer %d: Insert crops %c at %d\n", thread\_id, x, w\_index);

c\_index = (c\_index + 1) % warehouseSize;

w\_index = (w\_index + 1) % warehouseSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&full);

}

free(arg);

pthread\_exit(NULL);

}

void \*ShopOwner(void \*arg) {

int thread\_id = \*(int \*)arg;

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

sem\_wait(&full);

pthread\_mutex\_lock(&mutex);

char y = warehouse[(w\_index - 1) % warehouseSize];

warehouse[(w\_index - 1) % warehouseSize] = 'N';

printf("Shop owner %d: Remove crops %c from %d\n", thread\_id, y, (w\_index - 1) % warehouseSize);

w\_index = (w\_index - 1) % warehouseSize;

pthread\_mutex\_unlock(&mutex);

sem\_post(&empty);

}

printf("ShopOwner %d: ", thread\_id);

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

printf("%c ", warehouse[i]);

}

printf("\n");

free(arg);

pthread\_exit(NULL);

}

int main() {

// Initialize thread, mutex, and semaphores

pthread\_t Far[5], Sho[5];

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

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

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

int a[5] = {1, 2, 3, 4, 5};

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

int \*p = malloc(sizeof(int));

int \*q = malloc(sizeof(int));

\*p = a[i];

\*q = a[i];

pthread\_create(&Far[i], NULL, &Farmer, p);

pthread\_create(&Sho[i], NULL, &ShopOwner, q);

}

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

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

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

}

pthread\_mutex\_destroy(&mutex);

sem\_destroy(&empty);

sem\_destroy(&full);

return 0;

}

