Experiment 7

1.

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#define BUFFER\_SIZE 5

int buffer[BUFFER\_SIZE];

int count = 0;

pthread\_mutex\_tmutex = PTHREAD\_MUTEX\_INITIALIZER;

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

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

void \*producer(void \*arg) {

int item = 1;

while (1) {

pthread\_mutex\_lock(&mutex);

while (count > 0) {

pthread\_cond\_wait(&empty, &mutex);

}

buffer[0] = item;

printf("Produced item %d\n", item);

item++;

count++;

pthread\_cond\_signal(&full);

pthread\_mutex\_unlock(&mutex);

}

return NULL;

}

void \*consumer(void \*arg) {

while (1) {

pthread\_mutex\_lock(&mutex);

while (count == 0) {

pthread\_cond\_wait(&full, &mutex);

}

int item = buffer[0];

printf("Consumed item %d\n", item);

count--;

pthread\_cond\_signal(&empty);

pthread\_mutex\_unlock(&mutex);

}

return NULL;

}

int main() {

pthread\_tproducer\_thread, consumer\_thread;

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

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

pthread\_join(producer\_thread, NULL);

pthread\_join(consumer\_thread, NULL);

return 0;

}

2.

#include <stdio.h>

#include <stdlib.h>

#include <pthread.h>

#include <semaphore.h>

#define NUM\_READERS 3

#define NUM\_WRITERS 2

pthread\_mutex\_tmutex = PTHREAD\_MUTEX\_INITIALIZER;

sem\_trw\_mutex;

sem\_treaders\_mutex;

intreaders\_count = 0;

void \*reader(void \*arg) {

while (1) {

sem\_wait(&readers\_mutex);

pthread\_mutex\_lock(&mutex);

readers\_count++;

if (readers\_count == 1) {

sem\_wait(&rw\_mutex);

}

pthread\_mutex\_unlock(&mutex);

sem\_post(&readers\_mutex);

printf("Reader %d is reading.\n", \*(int\*)arg);

pthread\_mutex\_lock(&mutex);

readers\_count--;

if (readers\_count == 0) {

sem\_post(&rw\_mutex);

}

pthread\_mutex\_unlock(&mutex);

usleep(rand() % 1000000);

}

return NULL;

}

void \*writer(void \*arg) {

while (1) {

sem\_wait(&rw\_mutex);

printf("Writer %d is writing.\n", \*(int\*)arg);

sem\_post(&rw\_mutex);

usleep(rand() % 1000000);

}

return NULL;

}

int main() {

pthread\_t readers[NUM\_READERS], writers[NUM\_WRITERS];

intreader\_ids[NUM\_READERS], writer\_ids[NUM\_WRITERS];

sem\_init(&rw\_mutex, 0, 1);

sem\_init(&readers\_mutex, 0, NUM\_READERS);

for (inti = 0; i< NUM\_READERS; i++) {

reader\_ids[i] = i + 1;

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

}

for (inti = 0; i< NUM\_WRITERS; i++) {

writer\_ids[i] = i + 1;

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

}

for (inti = 0; i< NUM\_READERS; i++) {

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

}

for (inti = 0; i< NUM\_WRITERS; i++) {

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

}

sem\_destroy(&rw\_mutex);

sem\_destroy(&readers\_mutex);

return 0;

}