CS205 OS Lab Sesion

Name: Kothapalli Hemanth Simha

Reg.No: 9201141

Date: 08th November 2021

Q1) Write a program with N threads. Thread i must print number i in a continuous loop. Without any synchronization between the threads, the threads will print their numbers in any order. Now, add synchronization to your code such that the numbers are printed in the order 1, 2, ..., N , 1, 2, ...,N , and so on. You may want to start with N =2 and then move on to larger values of N .

With 3 threads:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<pthread.h>
pthread_cond_t c1 = PTHREAD_COND_INITIALIZER;
pthread_cond_t c2 = PTHREAD_COND_INITIALIZER;
pthread_cond_t c3 = PTHREAD_COND_INITIALIZER;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
int global = 1;
void *sample(void *arg){
        int i = (int)*(int*) arg;
        for (int c=0; c < 3; c++){
                pthread_mutex_lock(&m);
                printf("%d ", i);
                if (global == 3){
                        global = 1;
                        pthread_cond_signal(&c1);
                        if (c<2) pthread_cond_wait(&c3, &m);
                }
                else if (global == 1){
```

```
                        global = 2;
                        pthread_cond_signal(&c2);
                        if (c<2) pthread_cond_wait(&c1, &m);
                }
                else if (global == 2){
                        global = 3;
                        pthread_cond_signal(&c3);
                        if (c<2) pthread_cond_wait(&c2, &m);
                }
                pthread_mutex_unlock(&m);
        }
        return NULL;
}
int main(int argc, char* argv[]){
        pthread_t t1, t2, t3;
        int n1 = 1;
        int n2 = 2;
        int n3 = 3;
        pthread_create(&t1, NULL, sample, (void *)&n1);
        sleep(1);
        pthread_create(&t2, NULL, sample, (void *)&n2);
        pthread_create(&t3, NULL, sample, (void *)&n3);
        pthread_join(t1, NULL);
        pthread_join(t2, NULL);
        pthread_join(t3, NULL);
        printf("\n");
        return 0;
}
```

Output:

```
hemanth@hemanth:~$ cd 08-11-2021/
hemanth@hemanth:~/08-11-2021$ gcc Ex1.c -l pthread
hemanth@hemanth:~/08-11-2021$ ./a.out
1 2 3 1 2 3 1 2 3
hemanth@hemanth:~/08-11-2021$
```

With N threads:

```
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
#include<pthread.h>
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t *co = NULL;
int n, global = 0;
void *sample(void *arg){
        int k = (int)*(int*) arg;
        for (int c=0; c < 5; c++){
                pthread_mutex_lock(&m);
                if (k!= global)
                        pthread_cond_wait(&co[k], &m);
                printf("%d ", k);
                if (global >= n-1)
                        global = 0;
                else
                        global++;
                pthread_cond_signal(&co[global]);
                pthread_mutex_unlock(&m);
        }
        return NULL;
}
i
```

```
nt main(int argc, char* argv[]){
        printf("Enter the number of Threads: ");
        scanf("%d", &n);
        pthread_t *t = (pthread_t *)malloc(sizeof(pthread_t)*n);
        co = (pthread_cond_t *)malloc(sizeof(pthread_cond_t)*n);
        for (int i=0; i<n; i++){
                pthread_cond_init(&co[i], NULL);
        }
        int arr[n];
        for (int i=0; i<n; i++){
                arr[i] = i;
                pthread_create(&t[i], NULL, sample, &arr[i]);
        }
        for (int j=0; j<n; j++){
                pthread_join(t[j], NULL);
        }
        free(co);
        free(t);
        printf("\n");
        return 0;
}
```

Output:

```
hemanth@hemanth:~/08-11-2021$ gcc ex1.c -l pthread
hemanth@hemanth:~/08-11-2021$ ./a.out
Enter the number of Threads: 9
0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8
hemanth@hemanth:~/08-11-2021$
```

Q2. Implement Producer-Consumer problem using Pthreads, mutex locks, condition variables. The program should take 4 command line arguments: how many numbers to "produce" (M), the maximum size of the buffer in which the produced numbers should be stored (N), the number of worker threads to consume these numbers (C), and the number of master threads to produce numbers (P). The producer will generate every integer from 0 to M −1 exactly once and consumed exactly once by the consumer threads. Your program should correctly synchronize the producer and consumer threads in such a way that every number is produced and consumed exactly once. Further, producers must not try to produce when the buffer is full, and consumers should not consume from an empty buffer. While you need to ensure that all C workers are involved in consuming the integers.

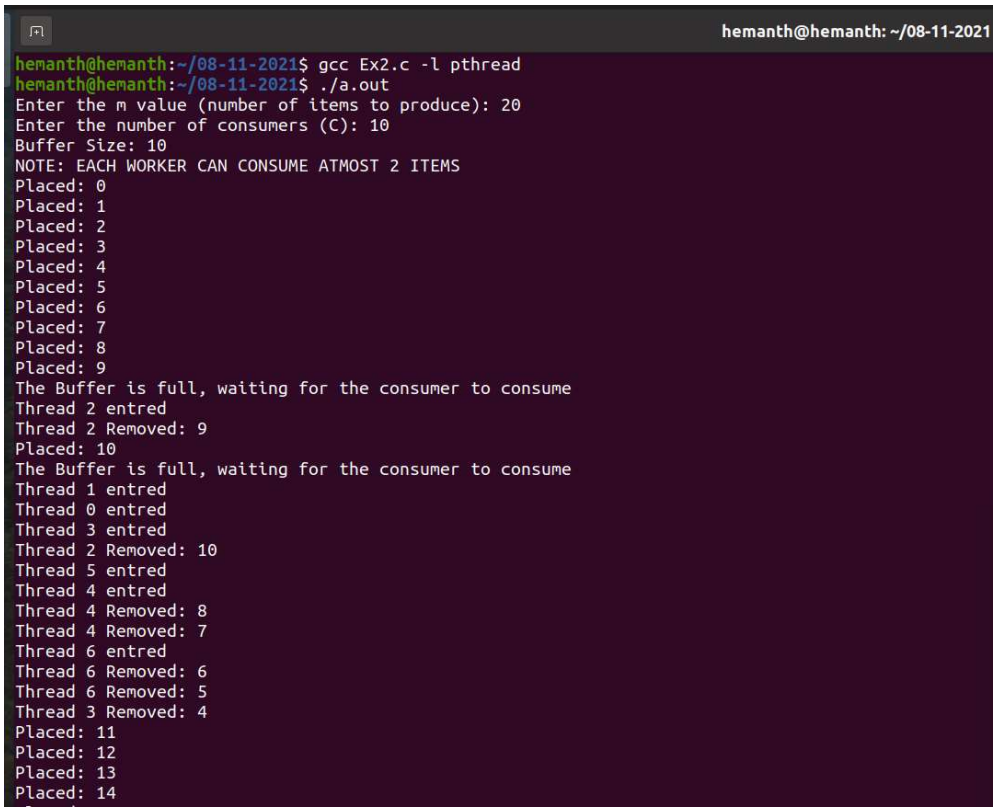
Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
#include<pthread.h>
#include<semaphore.h>
#define maxSize 10
int buffer[maxSize];
int count = 0;
pthread_cond_t full, empty;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
void put(int data){
        buffer[count] = data;
        count = count + 1;
}
int get(){
        count = count - 1;
        return buffer[count];
}
void *producer(void *arg){
        int loops = (int)*(int *)arg;
        for(int i = 0; i < loops; i++){
```

```
                pthread_mutex_lock(&m);
                while (count == maxSize){
                        printf("The Buffer is full, waiting for the consumer to consume\n");
                        pthread_cond_wait(&empty, &m);
                }
                put(i);
                printf("Placed: %d\n", i);
                pthread_cond_signal(&full);
                pthread_mutex_unlock(&m);
        }
}
void *consumer(void *arg){
        int id = (int)*(int *)arg;
        printf("Thread %d entred\n", id);
        for (int i=0; i<2; i++){
                pthread_mutex_lock(&m);
                int temp;
                while (count == 0){
                        printf("Waiting for producer to fill\n");
                        pthread_cond_wait(&full, &m);
                }
                temp = get();
                printf("Thread %d Removed: %d\n", id, temp);
                pthread_cond_signal(&empty);
                pthread_mutex_unlock(&m);
        }
}
```

```
int main(){
	pthread_t p, *t;
	pthread_cond_init(&full, NULL);
	pthread_cond_init(&empty, NULL);
	int m, c;
	printf("Enter the m value (number of items to produce): ");
	scanf("%d", &m);
	printf("Enter the number of consumers (C): ");
	scanf("%d", &c);
	printf("Buffer Size: %d\n", maxSize);
	printf("NOTE: EACH WORKER CAN CONSUME ATMOST 2 ITEMS \n");
	t = (pthread_t *)malloc(sizeof(pthread_t)*c);
	pthread_create(&p, NULL, producer, &m);
	int arr[c];
	for (int i=0; i<c; i++){
		arr[i] = i;
		pthread_create(&t[i], NULL, consumer, &arr[i]);
	}
	for (int j=0; j<c; j++){
		pthread_join(t[j], NULL);
	}
	free(t);
	return 0;
}
```

Output:



Q3. Implement Producer-Consumer problem using Pthreads, semaphores.

Code:

```
#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<pthread.h>

#include<semaphore.h>
```

```
#define maxSize 10
int buffer[maxSize];
int count = 0;
sem_t empty, full;
sem_t mutex;
void put(int data){
        buffer[count] = data;
        count = count + 1;
}
int get(){
        count = count - 1;
        return buffer[count];
}
void *producer(void *arg){
        sleep(1);
        int loops = (int)*(int *)arg;
        for(int i = 0; i < loops; i++){
                sem_wait(&empty);
                sem_wait(&mutex);
                put(i);
                printf("Placed: %d\n", i);
                sem_post(&mutex);
                sem_post(&full);
        }
        return NULL;
}
void *consumer(void *arg){
        int id = (int)*(int *)arg;
```

```
        printf("Thread %d entred\n", id);
        int temp;
        for (int i=0; i<5; i++){
                sem_wait(&full);
                sem_wait(&mutex);
                temp = get();
                printf("Worker %d Consumed: %d\n", id, temp);
                sem_post(&mutex);
                sem_post(&empty);
        }
        return NULL;
}
int main(){
        pthread_t p;
        pthread_t *c = NULL;
        sem_init(&empty, 0, maxSize);
        sem_init(&full, 0, 0);
        sem_init(&mutex, 0, 1);
        int m, d;
        printf("Enter the m value (number of items to produce): ");
        scanf("%d", &m);
        printf("Enter the number of consumers (C): ");
        scanf("%d", &d);
        printf("Buffer Size: %d\n", maxSize);
        c = (pthread_t*)malloc(sizeof(pthread_t)*d);
        int arr[d];
        pthread_create(&p, NULL, producer, &m);
```

```
        for (int i=0; i<d; i++){
                arr[i] = i;
                pthread_create(&c[i], NULL, consumer, &arr[i]);
        }
        for (int j=0; j<d; j++){
                pthread_join(c[j], NULL);
        }
        free(c);
        return 0;
}
```

Output: