Lab 2

Ahmed Nazir, nazira1, 400188027

Kevin Zhang, zhank15, 400268742

Madrigal Weersink, weersinm, 400244666

Stephen Oh, ohs9, 400189617

Undergraduate Department of Computing and Software, McMaster University

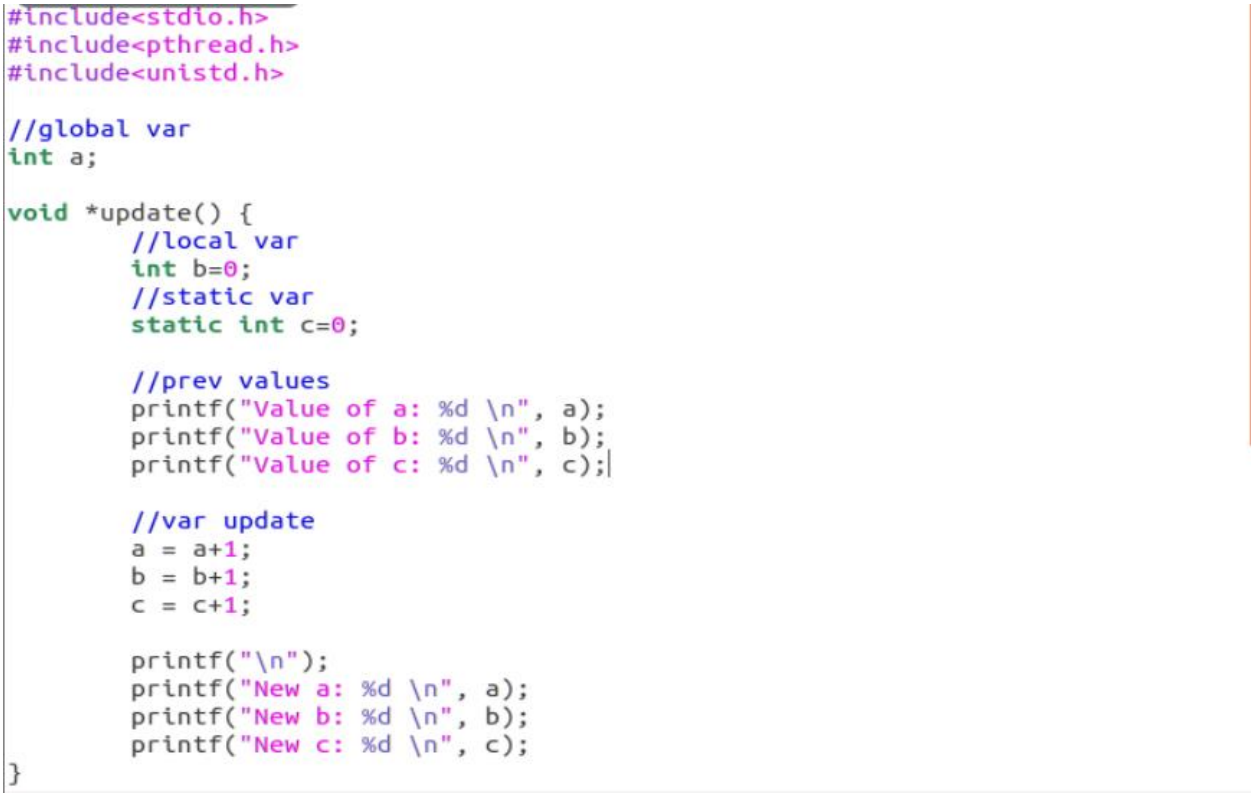
Operating Systems, 3SH3

Dr. Anwar Mirza

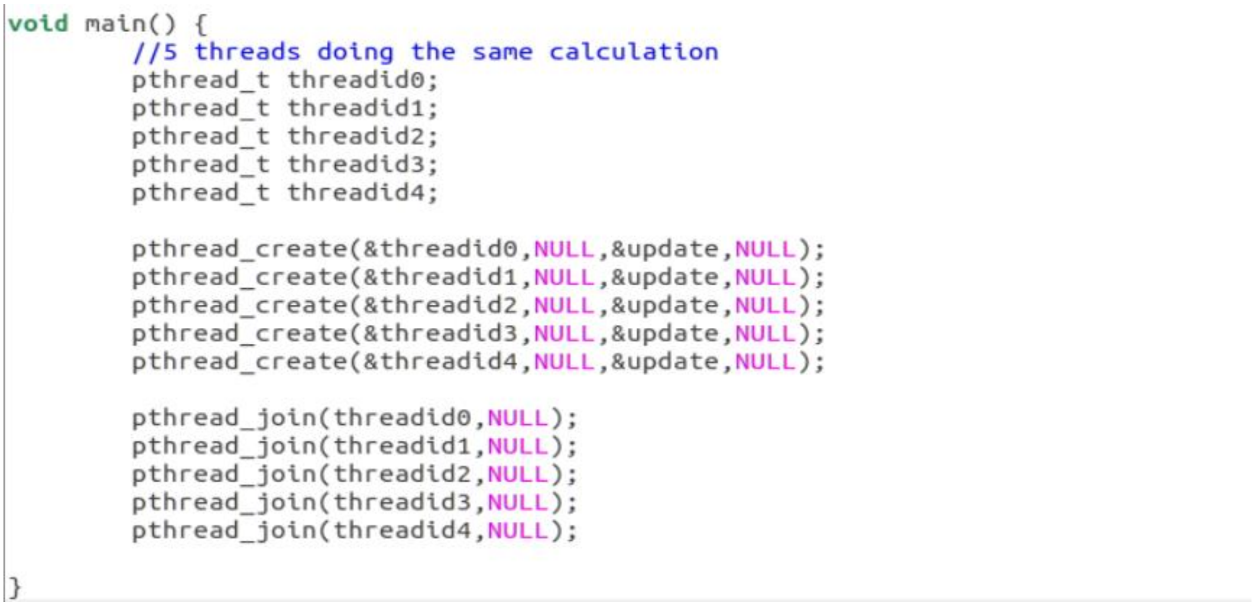
March 5, 2022

Q1

Code:

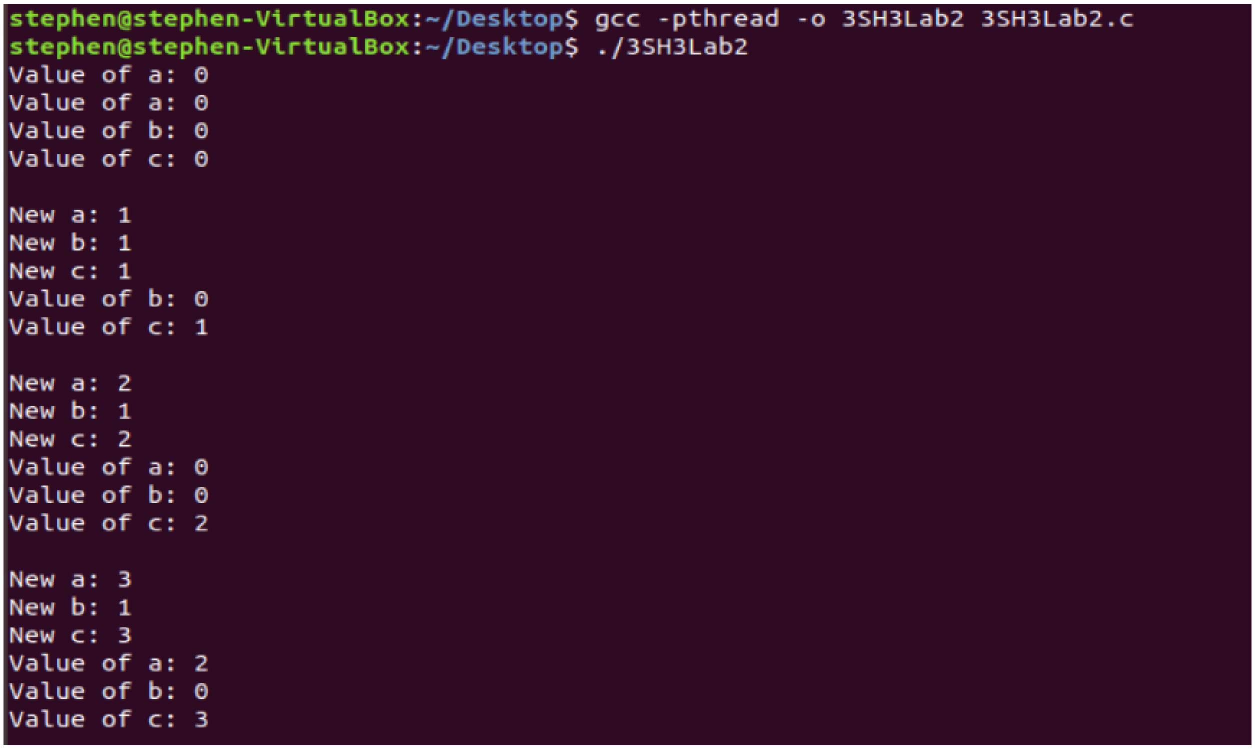


```c
#include<stdio.h>
#include<pthread.h>
#include<unistd.h>

//global var
int a;

void *update() {
        //local var
        int b=0;
        //static var
        static int c=0;

        //prev values
        printf("Value of a: %d \n", a);
        printf("Value of b: %d \n", b);
        printf("Value of c: %d \n", c);

        //var update
        a = a+1;
        b = b+1;
        c = c+1;

        printf("\n");
        printf("New a: %d \n", a);
        printf("New b: %d \n", b);
        printf("New c: %d \n", c);
}
```

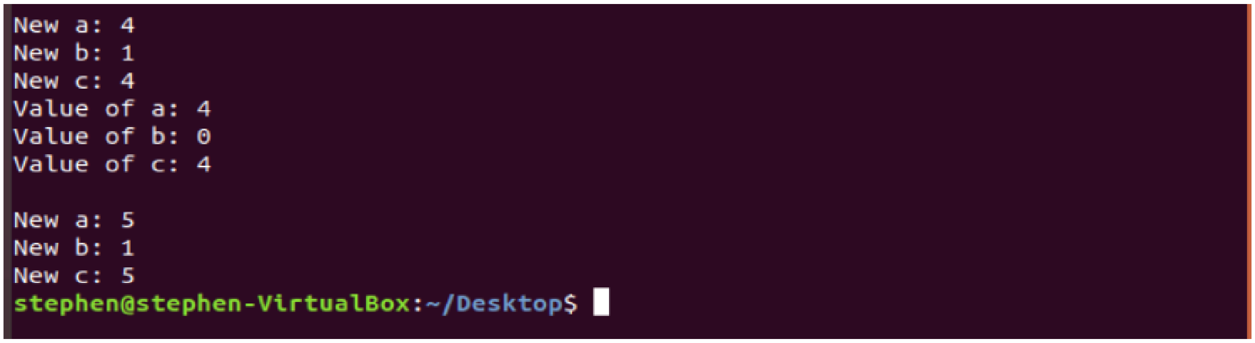


```c
void main() {
        //5 threads doing the same calculation
        pthread_t threadid0;
        pthread_t threadid1;
        pthread_t threadid2;
        pthread_t threadid3;
        pthread_t threadid4;

        pthread_create(&threadid0,NULL,&update,NULL);
        pthread_create(&threadid1,NULL,&update,NULL);
        pthread_create(&threadid2,NULL,&update,NULL);
        pthread_create(&threadid3,NULL,&update,NULL);
        pthread_create(&threadid4,NULL,&update,NULL);

        pthread_join(threadid0,NULL);
        pthread_join(threadid1,NULL);
        pthread_join(threadid2,NULL);
        pthread_join(threadid3,NULL);
        pthread_join(threadid4,NULL);

}
```

Output:



```
stephen@stephen-VirtualBox:~/Desktop$ gcc -pthread -o 3SH3Lab2 3SH3Lab2.c
stephen@stephen-VirtualBox:~/Desktop$ ./3SH3Lab2
Value of a: 0
Value of a: 0
Value of b: 0
Value of c: 0

New a: 1
New b: 1
New c: 1
Value of b: 0
Value of c: 1

New a: 2
New b: 1
New c: 2
Value of a: 0
Value of b: 0
Value of c: 2

New a: 3
New b: 1
New c: 3
Value of a: 2
Value of b: 0
Value of c: 3
```



```
New a: 4
New b: 1
New c: 4
Value of a: 4
Value of b: 0
Value of c: 4

New a: 5
New b: 1
New c: 5
stephen@stephen-VirtualBox:~/Desktop$ █
```

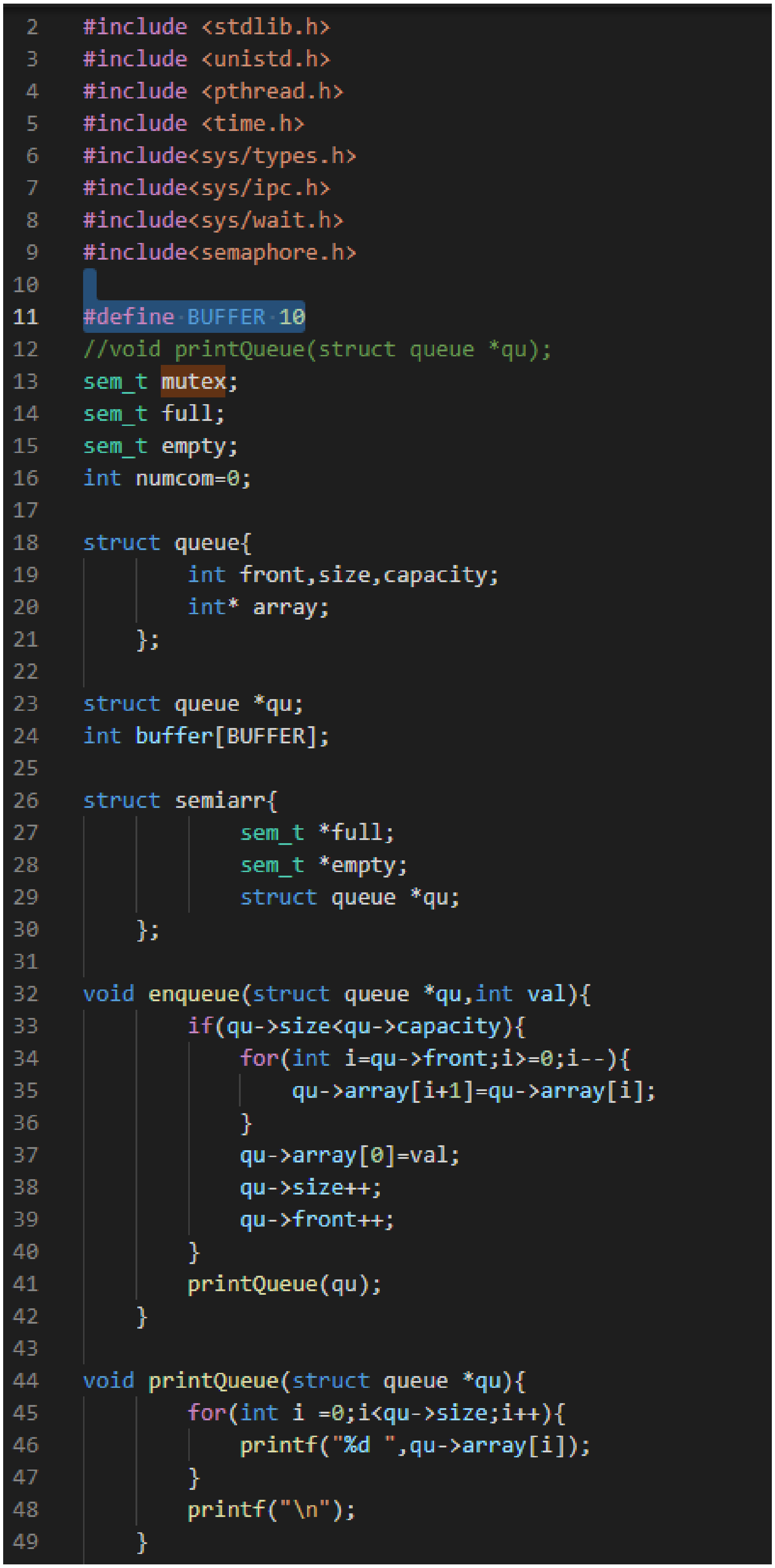
Variable a was declared as a global variable, variable b was declared as a local variable, and variable c was declared as a static variable. All variables held integer values.

Upon execution of the program, all variables were incremented by 1 by each of the five threads.

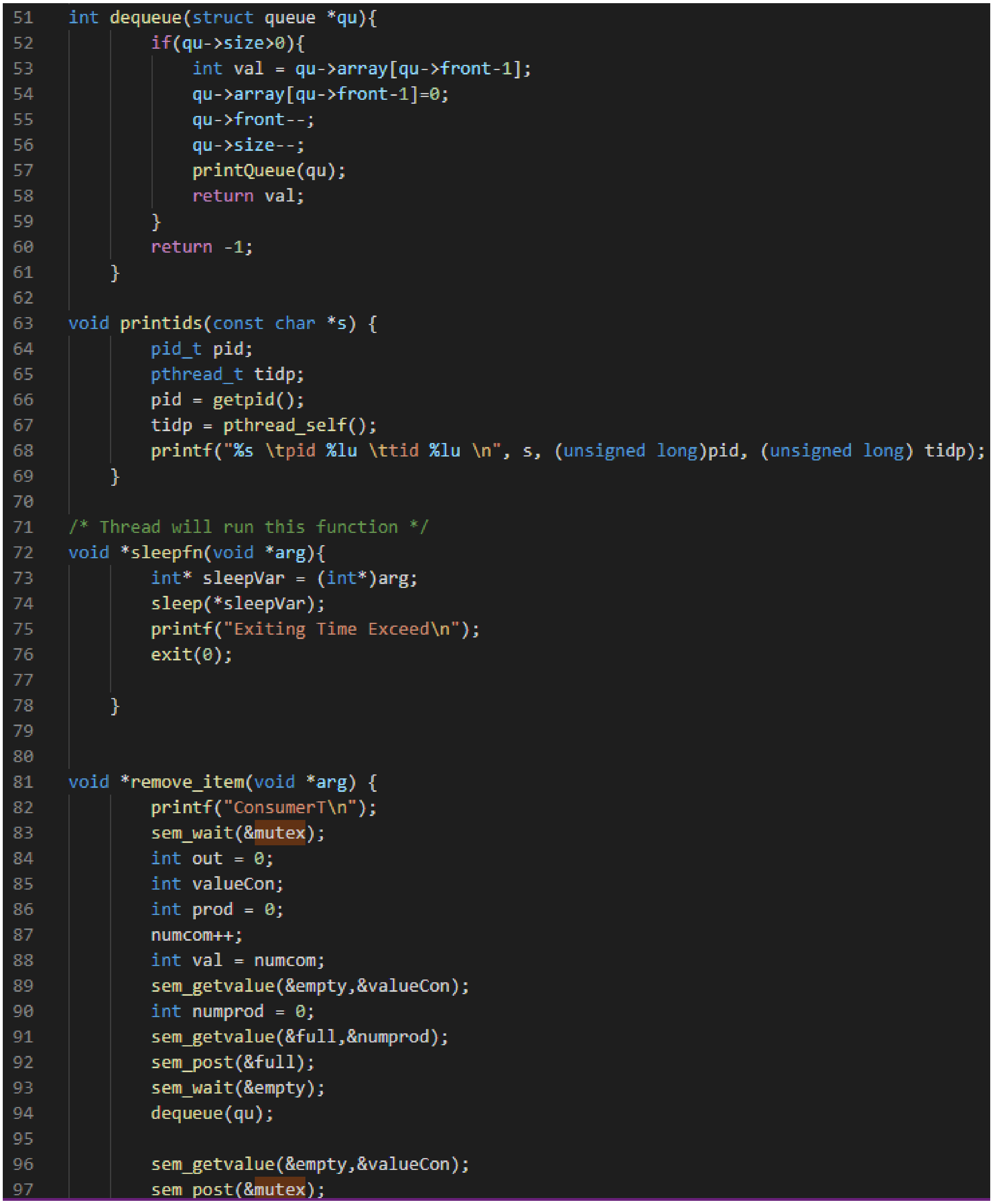
The final values for each of the variable's a, b, and c were 5, 1, 5 respectively. The reason why the variable sums were not uniform upon execution by each thread was because of the variable types. For both the global and static variables, the new sums were retained and updated upon each thread execution of the update function. For the local variable sum b however, the new sum was not retained because of the local variable type and as a result, had a different final value than variables a and c.

2.

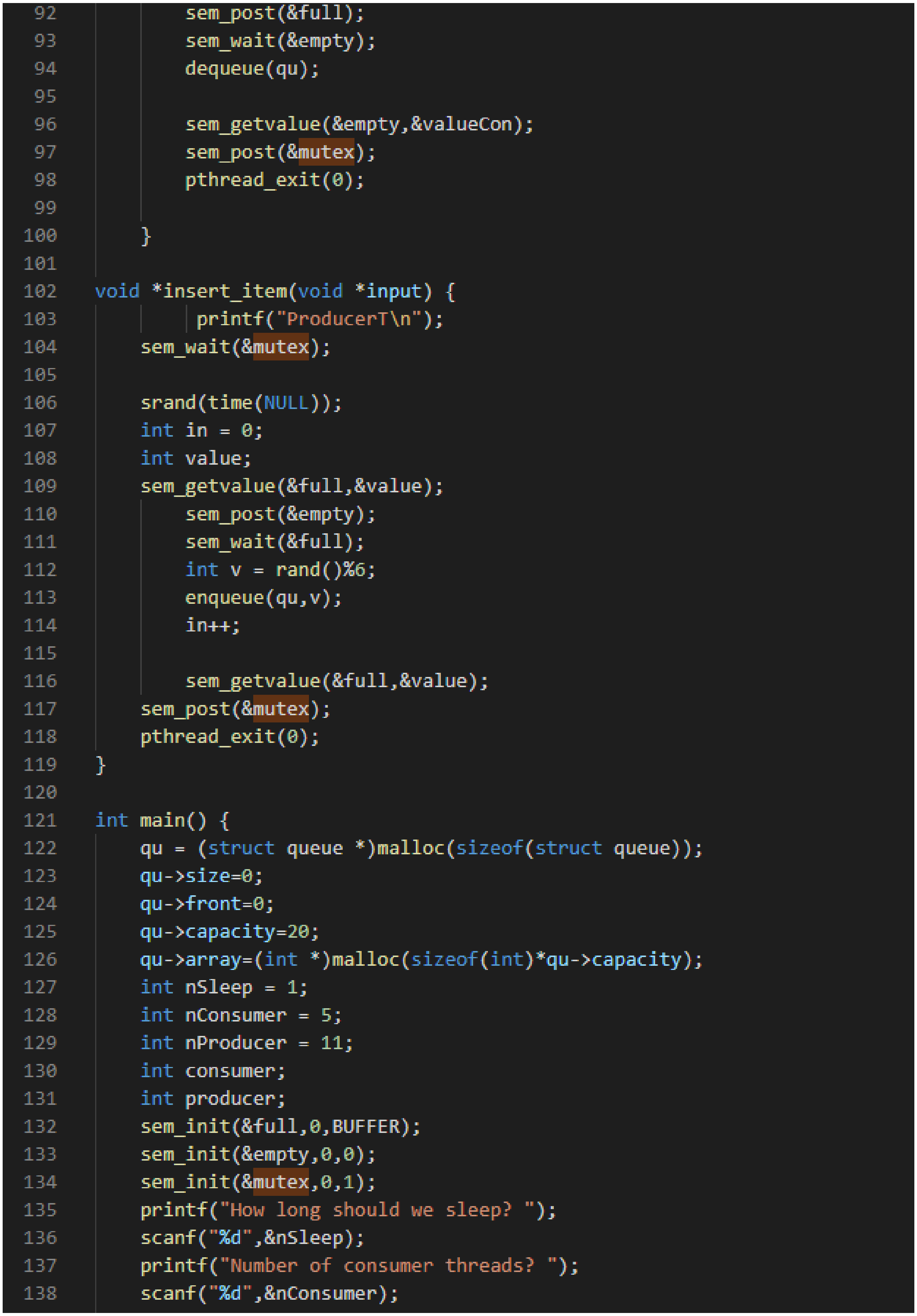
Code:



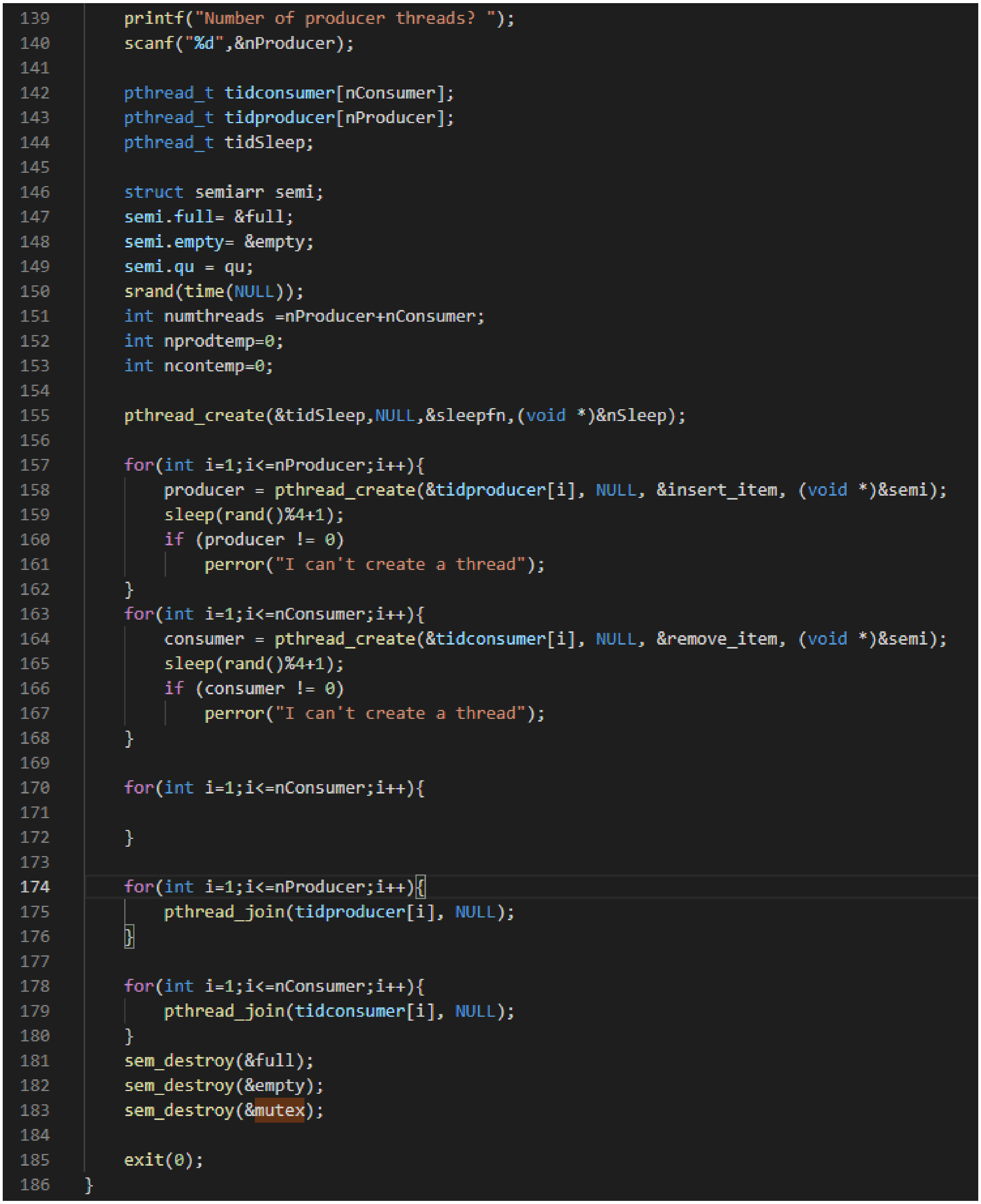
```c
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/wait.h>
#include<semaphore.h>

#define BUFFER 10
//void printQueue(struct queue *qu);
sem_t mutex;
sem_t full;
sem_t empty;
int numcom=0;

struct queue{
        int front,size,capacity;
        int* array;
    };

struct queue *qu;
int buffer[BUFFER];

struct semiarr{
            sem_t *full;
            sem_t *empty;
            struct queue *qu;
    };

void enqueue(struct queue *qu,int val){
        if(qu->size<qu->capacity){
            for(int i=qu->front;i>=0;i--){
                qu->array[i+1]=qu->array[i];
            }
            qu->array[0]=val;
            qu->size++;
            qu->front++;
        }
        printQueue(qu);
    }

void printQueue(struct queue *qu){
        for(int i =0;i<qu->size;i++){
            printf("%d ",qu->array[i]);
        }
        printf("\n");
    }
```



```c
int dequeue(struct queue *qu){
        if(qu->size>0){
            int val = qu->array[qu->front-1];
            qu->array[qu->front-1]=0;
            qu->front--;
            qu->size--;
            printQueue(qu);
            return val;
        }
        return -1;
    }

void printids(const char *s) {
        pid_t pid;
        pthread_t tidp;
        pid = getpid();
        tidp = pthread_self();
        printf("%s \tpid %lu \ttid %lu \n", s, (unsigned long)pid, (unsigned long) tidp);
    }

/* Thread will run this function */
void *sleepfn(void *arg){
        int* sleepVar = (int*)arg;
        sleep(*sleepVar);
        printf("Exiting Time Exceed\n");
        exit(0);

    }


void *remove_item(void *arg) {
        printf("ConsumerT\n");
        sem_wait(&mutex);
        int out = 0;
        int valueCon;
        int prod = 0;
        numcom++;
        int val = numcom;
        sem_getvalue(&empty,&valueCon);
        int numprod = 0;
        sem_getvalue(&full,&numprod);
        sem_post(&full);
        sem_wait(&empty);
        dequeue(qu);

        sem_getvalue(&empty,&valueCon);
        sem_post(&mutex);
```

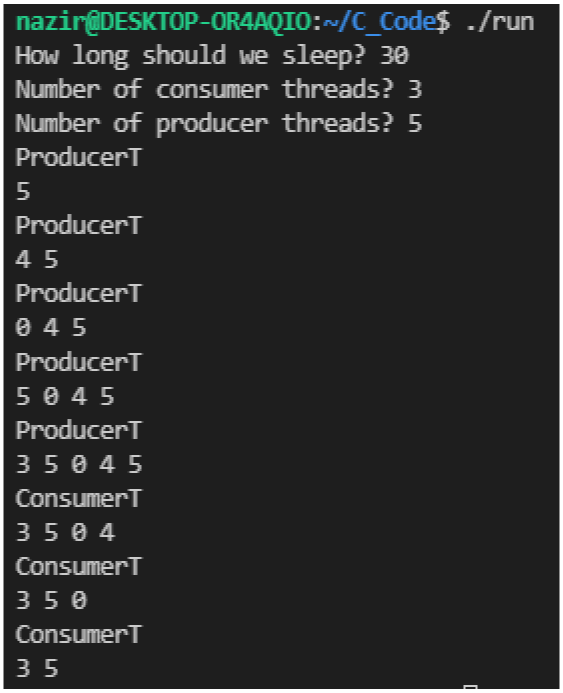


```c
        sem_post(&full);
        sem_wait(&empty);
        dequeue(qu);

        sem_getvalue(&empty,&valueCon);
        sem_post(&mutex);
        pthread_exit(0);

    }

void *insert_item(void *input) {
         printf("ProducerT\n");
    sem_wait(&mutex);

    srand(time(NULL));
    int in = 0;
    int value;
    sem_getvalue(&full,&value);
        sem_post(&empty);
        sem_wait(&full);
        int v = rand()%6;
        enqueue(qu,v);
        in++;

        sem_getvalue(&full,&value);
    sem_post(&mutex);
    pthread_exit(0);
}

int main() {
    qu = (struct queue *)malloc(sizeof(struct queue));
    qu->size=0;
    qu->front=0;
    qu->capacity=20;
    qu->array=(int *)malloc(sizeof(int)*qu->capacity);
    int nSleep = 1;
    int nConsumer = 5;
    int nProducer = 11;
    int consumer;
    int producer;
    sem_init(&full,0,BUFFER);
    sem_init(&empty,0,0);
    sem_init(&mutex,0,1);
    printf("How long should we sleep? ");
    scanf("%d",&nSleep);
    printf("Number of consumer threads? ");
    scanf("%d",&nConsumer);
```

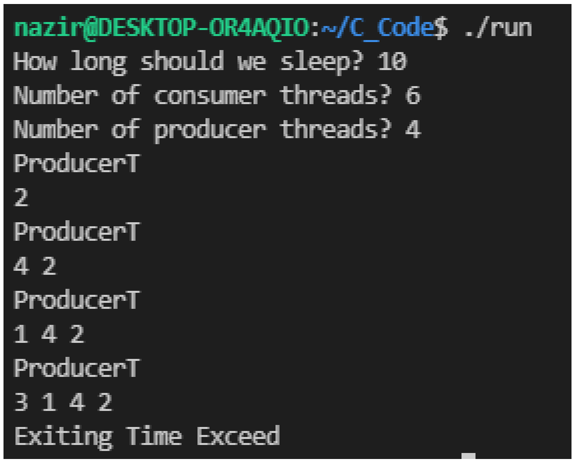


```
    printf("Number of producer threads? ");
    scanf("%d",&nProducer);

    pthread_t tidconsumer[nConsumer];
    pthread_t tidproducer[nProducer];
    pthread_t tidSleep;

    struct semiarr semi;
    semi.full= &full;
    semi.empty= &empty;
    semi.qu = qu;
    srand(time(NULL));
    int numthreads =nProducer+nConsumer;
    int nprodtemp=0;
    int ncontemp=0;

    pthread_create(&tidSleep,NULL,&sleepfn,(void *)&nSleep);

    for(int i=1;i<=nProducer;i++){
        producer = pthread_create(&tidproducer[i], NULL, &insert_item, (void *)&semi);
        sleep(rand()%4+1);
        if (producer != 0)
            perror("I can't create a thread");
    }
    for(int i=1;i<=nConsumer;i++){
        consumer = pthread_create(&tidconsumer[i], NULL, &remove_item, (void *)&semi);
        sleep(rand()%4+1);
        if (consumer != 0)
            perror("I can't create a thread");
    }

    for(int i=1;i<=nConsumer;i++){

    }

    for(int i=1;i<=nProducer;i++){
        pthread_join(tidproducer[i], NULL);
    }

    for(int i=1;i<=nConsumer;i++){
        pthread_join(tidconsumer[i], NULL);
    }
    sem_destroy(&full);
    sem_destroy(&empty);
    sem_destroy(&mutex);

    exit(0);
}
```

Output:



```
nazir@DESKTOP-OR4AQIO:~/C_Code$ ./run
How long should we sleep? 30
Number of consumer threads? 3
Number of producer threads? 5
ProducerT
5
ProducerT
4 5
ProducerT
0 4 5
ProducerT
5 0 4 5
ProducerT
3 5 0 4 5
ConsumerT
3 5 0 4
ConsumerT
3 5 0
ConsumerT
3 5
```



```
nazir@DESKTOP-OR4AQIO:~/C_Code$ ./run
How long should we sleep? 10
Number of consumer threads? 6
Number of producer threads? 4
ProducerT
2
ProducerT
4 2
ProducerT
1 4 2
ProducerT
3 1 4 2
Exiting Time Exceed
```