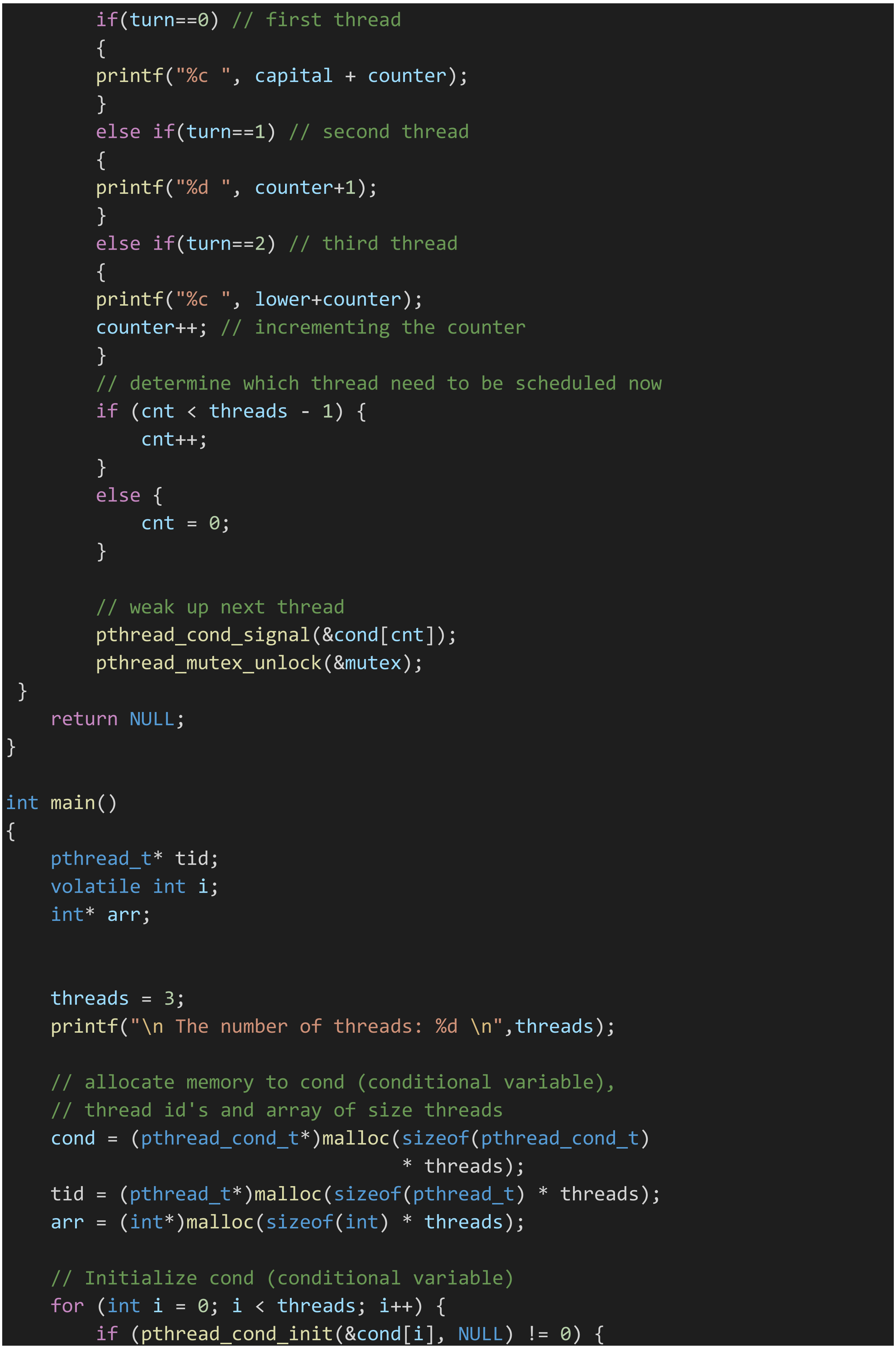
OS LAB – End Semester Code

Name- Raj Chauhan   Roll no – AU1940215

--------------------------------------Question-1 --------------------------------------

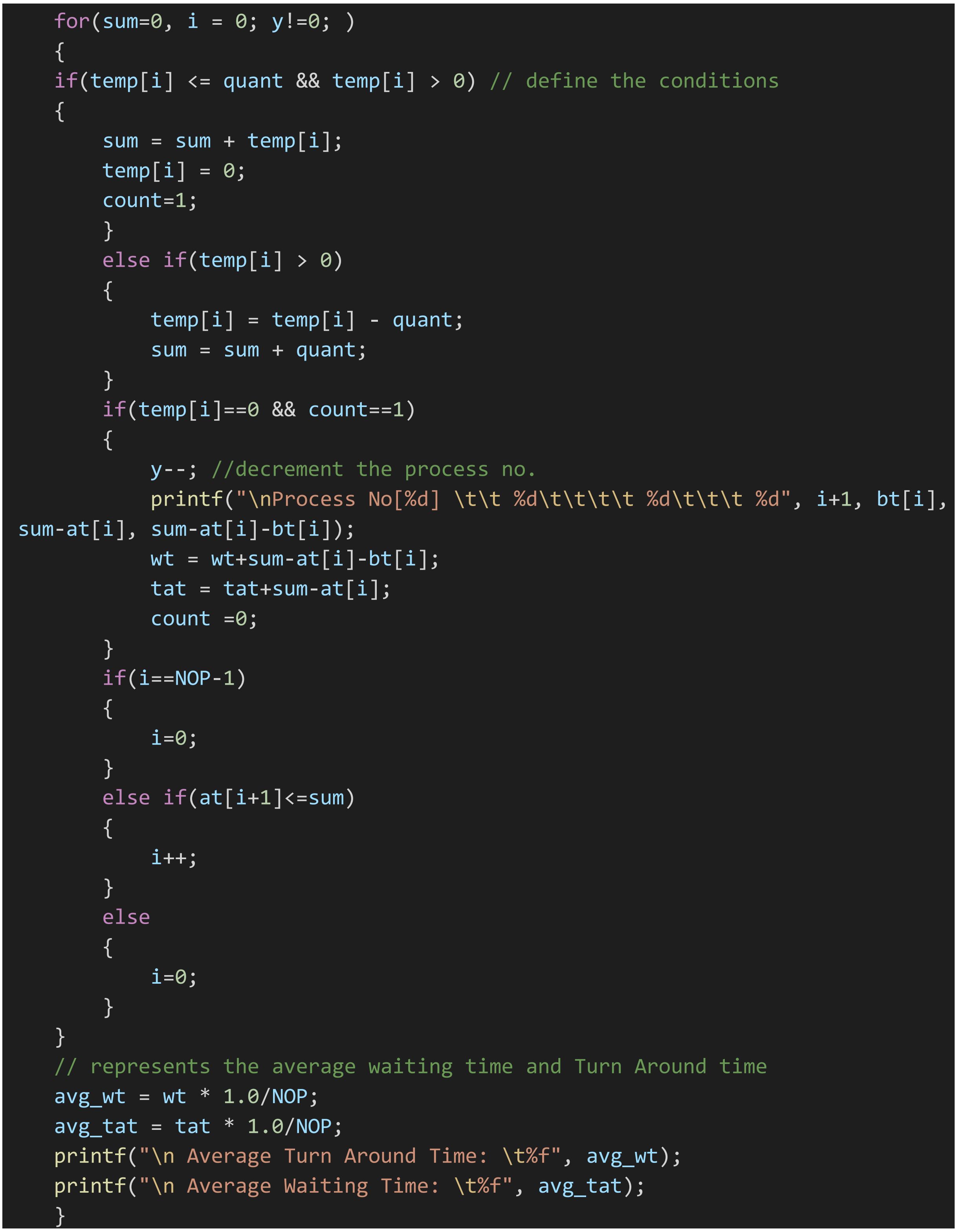
```
//Name - Raj Chauhan
//Question-1 using multithreading in C
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t* cond = NULL;

int threads;
volatile int cnt = 0;
int counter = 0;
char capital = 65;
char lower = 97;

// function to synchronize threads
void* foo(void* arg)
{
    // turn is a basically to identify a thread
    int turn = *(int*)arg;

  while(counter<25)
  {
        pthread_mutex_lock(&mutex);

        // cnt is used to determine which thread should
        // enter into critical section(printf() statement)
        if (turn != cnt) {

            // put all thread except one thread in waiting
            // state
            pthread_cond_wait(&cond[turn], &mutex);
        }

//if the turn if of first thread then we will display capital letter.
//in the second thread we will display the number
//We will display lower letter in the third thread
```



```
        if(turn==0) // first thread
        {
        printf("%c ", capital + counter);
        }
        else if(turn==1) // second thread
        {
        printf("%d ", counter+1);
        }
        else if(turn==2) // third thread
        {
        printf("%c ", lower+counter);
        counter++; // incrementing the counter
        }
        // determine which thread need to be scheduled now
        if (cnt < threads - 1) {
            cnt++;
        }
        else {
            cnt = 0;
        }

        // weak up next thread
        pthread_cond_signal(&cond[cnt]);
        pthread_mutex_unlock(&mutex);
 }
    return NULL;
}

int main()
{
    pthread_t* tid;
    volatile int i;
    int* arr;


    threads = 3;
    printf("\n The number of threads: %d \n",threads);

    // allocate memory to cond (conditional variable),
    // thread id's and array of size threads
    cond = (pthread_cond_t*)malloc(sizeof(pthread_cond_t)
                                   * threads);
    tid = (pthread_t*)malloc(sizeof(pthread_t) * threads);
    arr = (int*)malloc(sizeof(int) * threads);

    // Initialize cond (conditional variable)
    for (int i = 0; i < threads; i++) {
        if (pthread_cond_init(&cond[i], NULL) != 0) {
            perror("pthread_cond_init() error");
            exit(1);
        }
    }

    // create threads
    for (i = 0; i < threads; i++) {
        arr[i] = i;
        pthread_create(&tid[i], NULL, foo, (void*)&arr[i]);
    }

    // waiting for thread
    for (i = 0; i < threads; i++) {
        pthread_join(tid[i], NULL);
    }

    return 0;
}
```

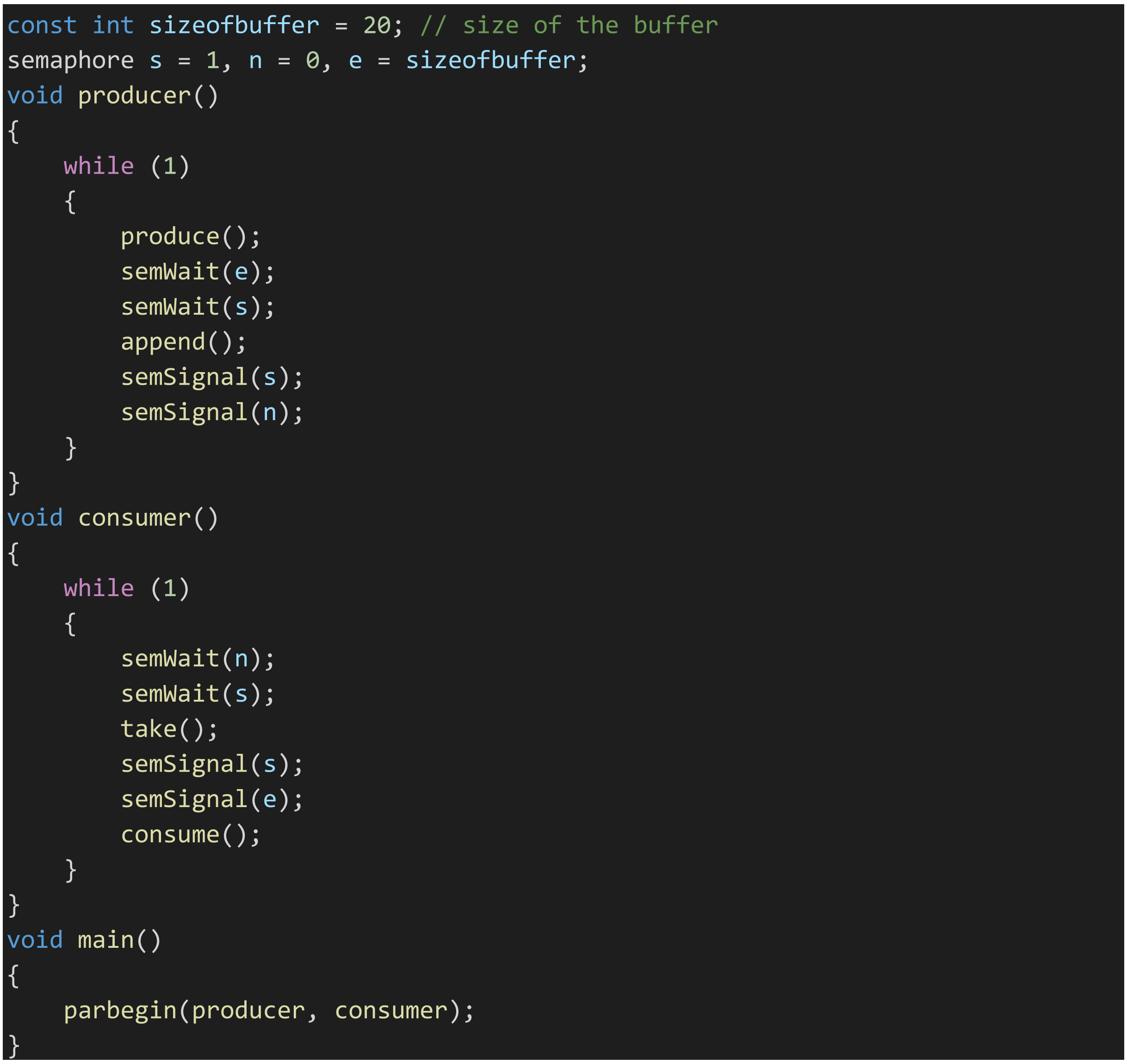
--------------------------------------Question-2--------------------------------------

```
//Name - Raj Chauhan
//Question-2
#include<stdio.h>

    void main()
    {
        // initlialize the variable name
        int i, NOP, sum=0,count=0, y, quant, wt=0, tat=0,temp[5];
        float avg_wt, avg_tat;
        y = 5; // Assign the number of process to variable y
        NOP = y;
        printf(" Total number of process in the system: %d",y);


    int at[5] = {0,4,10,15,17};

    int bt[5] = {7,18,27,30,36};
    // store the burst time and arrival time in temp array

    // Accept the Time qunat
    quant=10;
    printf("Enter the Time Quantum for the process: %d \t",quant);

    // Display the process No, burst time, Turn Around Time and the waiting ti
me
    printf("\n Process No \t\t Burst Time \t\t TAT \t\t Waiting Time ");
```



```
    for(sum=0, i = 0; y!=0; )
    {
    if(temp[i] <= quant && temp[i] > 0) // define the conditions
    {
        sum = sum + temp[i];
        temp[i] = 0;
        count=1;
        }
        else if(temp[i] > 0)
        {
            temp[i] = temp[i] - quant;
            sum = sum + quant;
        }
        if(temp[i]==0 && count==1)
        {
            y--; //decrement the process no.
            printf("\nProcess No[%d] \t\t %d\t\t\t\t %d\t\t\t %d", i+1, bt[i],
sum-at[i], sum-at[i]-bt[i]);
            wt = wt+sum-at[i]-bt[i];
            tat = tat+sum-at[i];
            count =0;
        }
        if(i==NOP-1)
        {
            i=0;
        }
        else if(at[i+1]<=sum)
        {
            i++;
        }
        else
        {
            i=0;
        }
    }
    // represents the average waiting time and Turn Around time
    avg_wt = wt * 1.0/NOP;
    avg_tat = tat * 1.0/NOP;
    printf("\n Average Turn Around Time: \t%f", avg_wt);
    printf("\n Average Waiting Time: \t%f", avg_tat);
    }
```

-------------------------------------Question-3-------------------------------------



```
const int sizeofbuffer = 20; // size of the buffer
semaphore s = 1, n = 0, e = sizeofbuffer;
void producer()
{
    while (1)
    {
        produce();
        semWait(e);
        semWait(s);
        append();
        semSignal(s);
        semSignal(n);
    }
}
void consumer()
{
    while (1)
    {
        semWait(n);
        semWait(s);
        take();
        semSignal(s);
        semSignal(e);
        consume();
    }
}
void main()
{
    parbegin(producer, consumer);
}
```