Name - Parshwa Shah
Roll Number - AU1940263

Question 1 (Multi-threading)

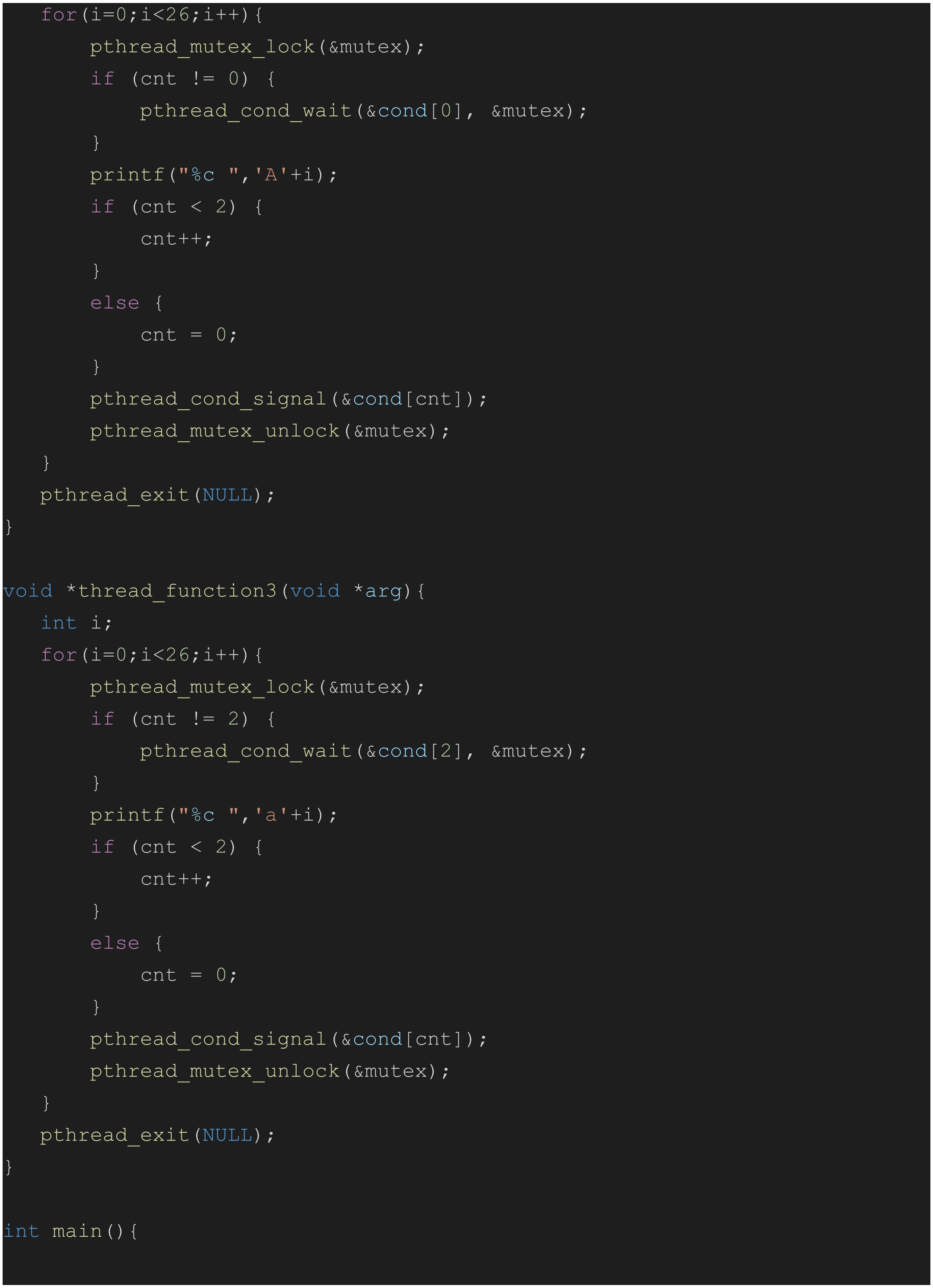
```
// Auther: Parshwa Shah
// Roll Number: AU1940263
// Question 1 using Multi-threading

#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t* cond = NULL;
volatile int cnt = 0;

void *thread_function(void *arg){
    int i;
    for(i=0;i<26;i++){
        pthread_mutex_lock(&mutex);
        if (cnt != 1) {
            pthread_cond_wait(&cond[1], &mutex);
        }
        printf("%d ",1+i);
        if (cnt < 2) {
            cnt++;
        }
        else {
            cnt = 0;
        }
        pthread_cond_signal(&cond[cnt]);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}

void *thread_function2(void *arg){
    int i;
```



```
    for(i=0;i<26;i++){
        pthread_mutex_lock(&mutex);
        if (cnt != 0) {
            pthread_cond_wait(&cond[0], &mutex);
        }
        printf("%c ",'A'+i);
        if (cnt < 2) {
            cnt++;
        }
        else {
            cnt = 0;
        }
        pthread_cond_signal(&cond[cnt]);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}

void *thread_function3(void *arg){
    int i;
    for(i=0;i<26;i++){
        pthread_mutex_lock(&mutex);
        if (cnt != 2) {
            pthread_cond_wait(&cond[2], &mutex);
        }
        printf("%c ",'a'+i);
        if (cnt < 2) {
            cnt++;
        }
        else {
            cnt = 0;
        }
        pthread_cond_signal(&cond[cnt]);
        pthread_mutex_unlock(&mutex);
    }
    pthread_exit(NULL);
}

int main(){
    pthread_t thread1, thread2, thread3;
    cond = (pthread_cond_t*)malloc(3 * sizeof(pthread_cond_t));
    pthread_cond_init(&cond[0], NULL);
    pthread_cond_init(&cond[1], NULL);
    pthread_cond_init(&cond[2], NULL);
    pthread_create(&thread1,NULL,thread_function2,NULL);
    pthread_create(&thread2,NULL,thread_function,NULL);
    pthread_create(&thread3,NULL,thread_function3,NULL);
    pthread_join(thread1,NULL);
    pthread_join(thread2,NULL);
    pthread_join(thread3,NULL);
    return 0;
}
```

Question 2 (Buddy’s algorithm)

```
#include<bits/stdc++.h>
using namespace std;

// Global size variable to know total size of the memory
int totalSize;

// List to manage the list of all free nodes
vector<pair<int, int>> availableNodes[100000];

// List to manage allocated Nodes
map<int, int> allocatedNodes;

int allocate(int requirement)
{
    // Calculating 2's power for the required size
    int requiredBlocks = ceil(log(requirement) / log(2));

    if (availableNodes[requiredBlocks].size() <= 0)
    {
        int i;
        for(i = requiredBlocks + 1; i < totalSize; i++)
        {
            if (availableNodes[i].size() != 0)
                break;
        }

        if (i == totalSize)
        {
            cout << "Can't allocate memory\n";
        }

        else
        {
            pair<int, int> assignee;
            assignee = availableNodes[i][0];

            availableNodes[i].erase(availableNodes[i].begin());
            i--;

            for(;i >= requiredBlocks; i--)
            {

                pair<int, int> firstPart, secondPart;
                firstPart = make_pair(assignee.first, assignee.first +
(assignee.second - assignee.first) / 2);
                secondPart = make_pair(assignee.first + (assignee.second -
assignee.first + 1) / 2, assignee.second);

                availableNodes[i].push_back(firstPart);

                availableNodes[i].push_back(secondPart);
                assignee = availableNodes[i][0];

                availableNodes[i].erase(availableNodes[i].begin());
            }

            cout << "Memory from " << assignee.first << " to " <<
assignee.second << " allocate" << "\n";

            allocatedNodes[assignee.first] = assignee.second -
assignee.first + 1;
```

```
            return assignee.first;
        }
    }
    else
    {
        pair<int, int> assignee = availableNodes[requiredBlocks][0];


availableNodes[requiredBlocks].erase(availableNodes[requiredBlocks].begin(
));

        cout << "Memory from " << assignee.first << " to " <<
assignee.second << " allocated" << "\n";

        allocatedNodes[assignee.first] = assignee.second - assignee.first +
1;

        return assignee.first;
    }
}

void deallocate(int target)
{

    if(allocatedNodes.find(target) == allocatedNodes.end())
    {
        cout << "Can't find the target\n";
        return;
    }

    int n = ceil(log(allocatedNodes[target]) / log(2));
    int i;
    availableNodes[n].push_back(make_pair(target, target + pow(2, n) - 1));
    cout << "Memory block from " << target << " to " << target + pow(2, n)
- 1 << " freed\n";

    int closeBlockNumber = target / allocatedNodes[target];
    int closeBlockAddress;

    if (closeBlockNumber % 2)
```

```
        closeBlockAddress = target - pow(2, n);
    else
        closeBlockAddress = target + pow(2, n);

    for(i = 0; i < availableNodes[n].size(); i++)
    {

        if (availableNodes[n][i].first == closeBlockAddress)
        {

            if (closeBlockNumber % 2)
            {
                availableNodes[n + 1].push_back(make_pair(closeBlockAddress, closeBlockAddress + 2 * (pow(2, n))));
                cout << "Connecting blocks from " << closeBlockAddress << " and " << target << " was done" << "\n";
            }
            else
            {
                availableNodes[n + 1].push_back(make_pair(target, target + 2 * (pow(2, n) - 1)));
                cout << "Connecting blocks from " << target << " and " ;
                cout << closeBlockAddress << " was done" << "\n";
            }
            availableNodes[n].erase(availableNodes[n].begin() + i);
            availableNodes[n].erase(availableNodes[n].begin() + availableNodes[n].size() - 1);
            break;
        }
    }

    allocatedNodes.erase(target);
}

int main()
{

    int n = 7;
    totalSize = n + 1;
    for(int i = 0; i <= n; i++) availableNodes[i].clear();
    availableNodes[n].push_back(make_pair(0, 127));

    int id1 = allocate(8);
    int id2 = allocate(16);
    int id3 = allocate(32);
    int id4 = allocate(64);
    deallocate(id1);
    deallocate(id2);
    deallocate(id4+1);
    deallocate(id3);

    return 0;
}
```

Question 3 (Bonus)

```
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>

#define MaxBoats 5
#define MaxStreams 5

sem_t readyForProducer;
sem_t ReadyForConsumer;

int streams[MaxStreams];
pthread_mutex_t lock;

int boatTowardsCons = 0;
int boatUsedbyProds = 0;

void *producer_code(void *pno)
{
   int boat;
   for(int i = 0; i < MaxBoats; i++) {
       boat = rand();
       sem_wait(&readyForProducer);
       pthread_mutex_lock(&lock);
       streams[boatTowardsCons] = boat;
       printf("Producer %d: Insert Item %d at %d\n", *((int
*)pno),streams[boatTowardsCons],boatTowardsCons);
       boatTowardsCons = (boatTowardsCons+1)%MaxStreams;
       pthread_mutex_unlock(&lock);
       sem_post(&ReadyForConsumer);
   }
}
void *consumer_code(void *cno)
{
   for(int i = 0; i < MaxBoats; i++) {
       sem_wait(&ReadyForConsumer);
       pthread_mutex_lock(&lock);
       int boat = streams[boatUsedbyProds];
       printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),boat,
boatUsedbyProds);
       boatUsedbyProds = (boatUsedbyProds+1)%MaxStreams;
       pthread_mutex_unlock(&lock);
       sem_post(&readyForProducer);
   }
}

int main()
{

   pthread_t prods[5],cons[5];
   pthread_mutex_init(&lock, NULL);
   sem_init(&readyForProducer,0,MaxStreams);
   sem_init(&ReadyForConsumer,0,0);

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

   for(int i = 0; i < 5; i++) {
       pthread_create(&prods[i], NULL, (void *)producer_code, (void
*)&arr[i]);
```

```
    }
    for(int i = 0; i < 5; i++) {
        pthread_create(&cons[i], NULL, (void *)consumer_code, (void
*)&arr[i]);
    }

    for(int i = 0; i < 5; i++) {
        pthread_join(prods[i], NULL);
    }
    for(int i = 0; i < 5; i++) {
        pthread_join(cons[i], NULL);
    }

    pthread_mutex_destroy(&lock);
    sem_destroy(&readyForProducer);
    sem_destroy(&ReadyForConsumer);

    return 0;

}
```