Name: Purvam Sheth
Enrolment No. AU1940151
EndSem-OS

Ahmedabad University

PART-C

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

#define MaxItems 5 // Maximum items a producer can produce or a consumer can consume
#define BufferSize 5 // Size of the buffer

sem_t empty;
sem_t full;
int in = 0;
int out = 0;
int buffer[BufferSize];
pthread_mutex_t mutex;

void *producer(void *pno)
{
   int item;
   for(int i = 0; i < MaxItems; i++) {
      item = rand(); // Produce an random item
      sem_wait(&empty);
      pthread_mutex_lock(&mutex);
      buffer[in] = item;
      printf("Producer %d: Insert Item %d at %d\n", *((int *)pno),buffer[in],in);
      in = (in+1)%BufferSize;
      pthread_mutex_unlock(&mutex);
      sem_post(&full);
   }
}
void *consumer(void *cno)
{
   for(int i = 0; i < MaxItems; i++) {
      sem_wait(&full);
      pthread_mutex_lock(&mutex);
      int item = buffer[out];
```

```
            printf("Consumer %d: Remove Item %d from %d\n",*((int *)cno),item, out);
            out = (out+1)%BufferSize;
            pthread_mutex_unlock(&mutex);
            sem_post(&empty);
        }
    }

    int main()
    {

        pthread_t pro[5],con[5];
        pthread_mutex_init(&mutex, NULL);
        sem_init(&empty,0,BufferSize);
        sem_init(&full,0,0);

        int a[5] = {1,2,3,4,5}; //Just used for numbering the producer and consumer

        for(int i = 0; i < 5; i++) {
            pthread_create(&pro[i], NULL, (void *)producer, (void *)&a[i]);
        }
        for(int i = 0; i < 5; i++) {
            pthread_create(&con[i], NULL, (void *)consumer, (void *)&a[i]);
        }

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

        pthread_mutex_destroy(&mutex);
        sem_destroy(&empty);
        sem_destroy(&full);

        return 0;

    }
```

PART-B

```
#include <stdio.h>

int main()

{

   // initlialize the variable name

   int i, NOP, sum = 0, count = 0, y, quant, wt = 0, tat = 0, at[10], bt[10], temp[10];
```

```
float avg_wt, avg_tat;
printf(" Total number of process in the system: ");
scanf("%d", &NOP);
y = NOP; // Assign the number of process to variable y

// Use for loop to enter the details of the process like Arrival time and the Burst Time
for (i = 0; i < NOP; i++)
{
   printf("\n Enter the Arrival and Burst time of the Process[%d]\n", i + 1);
   printf(" Arrival time is: \t"); // Accept arrival time
   scanf("%d", &at[i]);
   printf(" \nBurst time is: \t"); // Accept the Burst time
   scanf("%d", &bt[i]);
   temp[i] = bt[i]; // store the burst time in temp array
}
// Accept the Time qunat
printf("Enter the Time Quantum for the process: \t");
scanf("%d", &quant);
// Display the process No, burst time, Turn Around Time and the waiting time
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);
    getch();
}
```

Code-2.b

```
#include<bits/stdc++.h>
using namespace std;
int size;
vector<pair<int, int>> arr[100000];
map<int, int> mp;
void Buddy(int s)
{
	int n = ceil(log(s) / log(2));
	size = n + 1;
	for(int i = 0; i <= n; i++)
		arr[i].clear();
	arr[n].push_back(make_pair(0, s - 1));
}
void allocate(int s)
{
	int x = ceil(log(s) / log(2));
	if (arr[x].size() > 0)
	{
		pair<int, int> temp = arr[x][0];
		arr[x].erase(arr[x].begin());
		cout << "Memory from " << temp.first
			<< " to " << temp.second
			<< " allocated" << "\n";
		mp[temp.first] = temp.second - temp.first + 1;
	}
	else
	{
		int i;
		for(i = x + 1; i < size; i++)
		{
```

```
            if (arr[i].size() != 0)
                break;
        }
        if (i == size)
        {
            cout << "Sorry, failed to allocate memory\n";
        }
        else
        {
            pair<int, int> temp;
            temp = arr[i][0];

            arr[i].erase(arr[i].begin());
            i--;
            for(;i >= x; i--)
            {
                pair<int, int> pair1, pair2;
                pair1 = make_pair(temp.first, temp.first + (temp.second -
temp.first) / 2);
                pair2 = make_pair(temp.first + (temp.second - temp.first + 1) /
2, temp.second);
                arr[i].push_back(pair1);
                arr[i].push_back(pair2);
                temp = arr[i][0];
                arr[i].erase(arr[i].begin());
            }
            cout << "Memory from " << temp.first
                << " to " << temp.second
                << " allocate" << "\n";
            mp[temp.first] = temp.second -
                                    temp.first + 1;
```

```
                }
        }
}
void deallocate(int id)
{
        if(mp.find(id) == mp.end())
        {
                cout << "Sorry, invalid free request\n";
                return;
        }

        int n = ceil(log(mp[id]) / log(2));
        int i, buddyNumber, buddyAddress;
        arr[n].push_back(make_pair(id, id + pow(2, n) - 1));
        cout << "Memory block from " << id
                << " to "<< id + pow(2, n) - 1
                << " freed\n";
        buddyNumber = id / mp[id];
        if (buddyNumber % 2 != 0)
                buddyAddress = id - pow(2, n);
        else
                buddyAddress = id + pow(2, n);

        for(i = 0; i < arr[n].size(); i++)
        {
                if (arr[n][i].first == buddyAddress)
                {
                        if (buddyNumber % 2 == 0)
                        {
                                arr[n + 1].push_back(make_pair(id,
```

```
                        id + 2 * (pow(2, n) - 1)));
                        cout << "Coalescing of blocks starting at "
                                << id << " and " << buddyAddress
                                << " was done" << "\n";
                    }
                    else
                    {
                        arr[n + 1].push_back(make_pair(
                                buddyAddress, buddyAddress +
                                2 * (pow(2, n))));
                        cout << "Coalescing of blocks starting at "
                                << buddyAddress << " and "
                                << id << " was done" << "\n";
                    }
                    arr[n].erase(arr[n].begin() + i);
                    arr[n].erase(arr[n].begin() +
                    arr[n].size() - 1);
                    break;
                }
            }
            mp.erase(id);
}
// Driver code
int main()
{
        Buddy(128);
        allocate(16);
        allocate(16);
        allocate(16);
        allocate(16);
```

```
        deallocate(0);
        deallocate(9);
        deallocate(32);
        deallocate(16);
        return 0;
}
```

PART-A

```
#include<stdio.h>
#include<pthread.h>
#include<time.h>
pthread_t tid[3];
pthread_mutex_t mutex;
unsigned int rc;
//prototypes for callback functions
int upperCase = 65;
int lowerCase = 97;
int number = 1;
int flag = 0;
void* PrintCapitalAlpabets(void*);
void* PrintLowerNumber(void*);
void* PrintNumber(void*);
void main(void)
{
   pthread_create(&tid[0],0,&PrintCapitalAlpabets,0);
   pthread_create(&tid[1],0,&PrintNumber,0);
   pthread_create(&tid[2],0,&PrintLowerNumber,0);
  //sleep(3);
   pthread_join(tid[0],NULL);
   pthread_join(tid[1],NULL);
```

```
    pthread_join(tid[2],NULL);
}
void* PrintCapitalAlpabets(void *ptr)
{
     rc = pthread_mutex_lock(&mutex);
     while(upperCase<=90)
     {
     if(flag%3==0){
          printf("%c ",upperCase);
                    upperCase++;
                    flag++;
     }
     else{
          rc=pthread_mutex_unlock(&mutex);//if number is odd, do not print, release mutex
     }
     }
}
void* PrintLowerNumber(void* ptr1)
{
    rc = pthread_mutex_lock(&mutex);
    while(lowerCase<=122)
    {
      if(flag%3==2){
      printf("%c ",lowerCase);
                    lowerCase++;
                    flag++;
      }
      else{
          rc=pthread_mutex_unlock(&mutex);//if number is odd, do not print, release mutex
      }
```

```
    }
}
void* PrintNumber(void* ptr1)
{
    rc = pthread_mutex_lock(&mutex);
   while(number <= 26)
    {
    if(flag%3==1){
     printf("%d ",number);
                    number++;
                    flag++;
    }
    else{
            rc=pthread_mutex_unlock(&mutex);//if number is odd, do not print, release mutex
          }
    }
}
```