FAIZAN CHOUDHARY

20BCS021

OS LAB Practical Exam

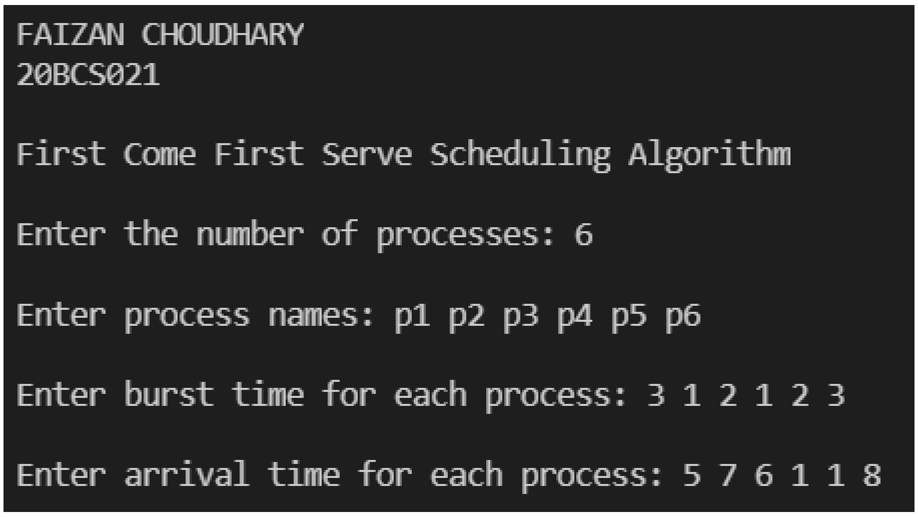
12th June 2022

1. Write a program to implement the First Come First Serve scheduling algorithm and find the average turnaround time, waiting time, completion time and response time for overall process. Also Print Gantt chart for it.

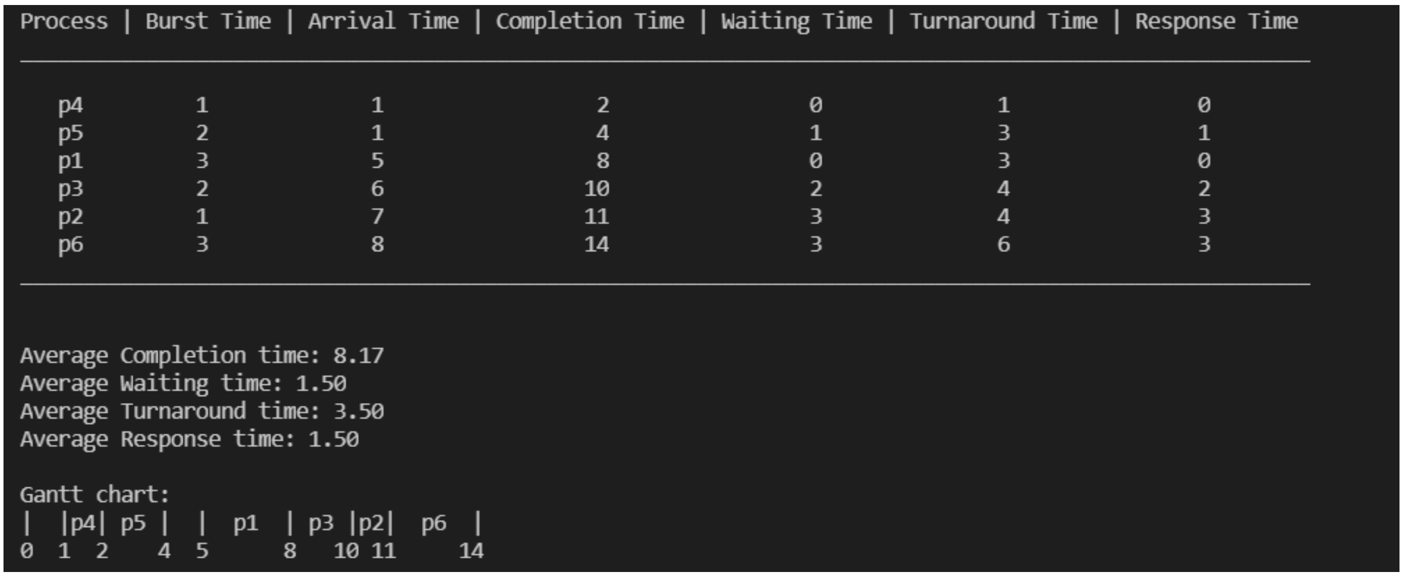
CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <string.h>
using namespace std;

struct node
{
    char n[10];
    int burst;
    int arrival;
    int completion;
    int waiting;
    int turnaround;
    int response;
    struct node *next;
};
struct node *front=NULL, *p, *ptr, *temp;

bool isEmpty () {
    if (front==NULL)
        return true;
    else
        return false;
}

void insertProcess (char *pr, int bt, int at) {
    ptr = (struct node *) malloc (sizeof(struct node));
    if (ptr == NULL) {
        cout<<"\nMemory could not be allocated!\n";
        return;
    }

    strcpy(ptr->n, pr);
    ptr->burst = bt;
    ptr->arrival = at;
    ptr->next=NULL;

    if (front == NULL || at < (front->arrival)) {
        ptr->next = front;
        front=ptr;
    }
    else {
        p=front;
        while (p->next != NULL && p->next->arrival <= at)
            p=p->next;
        ptr->next = p->next;
        p->next = ptr;
    }
}

void FCFS () {
    int time = 0;
    p = front;
    while (p != NULL) {
        if (time < p->arrival) {
            while (time != p->arrival)
                time++;
        }
        p->response = time - p->arrival;
        time += p->burst;
        p->completion = time;   // completion occurs after burst time ends
        p->turnaround = p->completion - p->arrival;     // tat = ct - at = wt + bt
        p->waiting = p->turnaround - p->burst;          // wt = tat - bt
        p = p->next;
    }
}

void display () {
    double tot_ct = 0, tot_wt =0, tot_tat = 0, tot_rt =0;
    int count = 0;
    p = front;
    cout<<"\n\nProcess | Burst Time | Arrival Time | Completion Time | Waiting Time |
Turnaround Time | Response Time\n";
    cout<<"_______________________________________________________________________________
________________________\n\n";
    while (p != NULL) {
        printf("   %s        %2d            %2d                  %2d                 %2d
      %2d              %2d\n", p->n, p->burst, p->arrival, p->completion, p->waiting, p-
>turnaround, p->response);

        tot_ct += p->completion;
        tot_wt += p->waiting;
        tot_tat += p->turnaround;
        tot_rt += p->response;

        count++;
        p = p->next;
    }
    cout<<"_______________________________________________________________________________
________________________\n\n";

    printf("\nAverage Completion time: %.2f",tot_ct / (float) count);
    printf("\nAverage Waiting time: %.2f", tot_wt / (float) count);
    printf("\nAverage Turnaround time: %.2f",tot_tat / (float) count);
    printf("\nAverage Response time: %.2f\n",tot_rt / (float) count);
}

void displayGantt () {
    int time = 0;
    p = front;
    cout<<"\nGantt chart: \n";
    // for printing structure
    while (p != NULL) {
        cout<<"|";
        if (time < p->arrival) {
            while (time != p->arrival) {
                time++;
            }
            time += p->burst;
            cout<<"  |";
        }
        else {
            time += p->arrival;
            if (front->arrival == 0)
                time += p->burst;
        }
        for (int i=0; i<(p->burst-1); i++)
            cout<<" ";
        cout<<p->n;
        for (int i=0; i<(p->burst-1); i++)
            cout<<" ";
        p = p->next;
    }
    cout<<"|"<<endl;
    p = front;
    time = 0;
    // for printing time below each process
    if (time < p->arrival && p->arrival != 0) {
        cout<<time;
        while (time != p->arrival) {
            time++;
        }
        time += p->burst;
        cout<<"  ";
    }
    cout<<p->arrival;
    while (p != NULL) {
        if (time < p->arrival) {
            while (time != p->arrival) {
                time++;
            }
            if (time < 9)
                cout<<"  "<<time;
            else
                cout<<" "<<time;
            time += p->burst;
        }
        else {
            time += p->arrival;
            if (front->arrival == 0)
                time += p->burst;
        }
        for (int i=0; i< 2*(p->burst)-1; i++)
            cout<<" ";
        if (p->completion < 9)
            cout<<" "<<p->completion;
        else
            cout<<p->completion;
        p = p->next;
    }
    cout<<endl<<endl;
}

void del () {
    p = front;
    front=front->next;
    delete p;
}

int main () {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nFirst Come First Serve Scheduling Algorithm\n";
    int n;

    cout<<"\nEnter the number of processes: ";
    cin>>n;
    char k[n][10];
    int bt[n], at[n];                   // burst time and arrival time

    cout<<"\nEnter process names: ";
    for (int i=0; i<n; i++)
        cin>>k[i];
    cout<<"\nEnter burst time for each process: ";
    for (int i=0; i<n; i++)
        cin>>bt[i];
    cout<<"\nEnter arrival time for each process: ";
    for (int i=0; i<n; i++)
        cin>>at[i];

    for (int i=0; i<n; i++)
        insertProcess(k[i],bt[i],at[i]);

    FCFS ();            // logic for calculating various times
    display ();         // displaying calculated values of time
    displayGantt ();    // to display Gantt chart
    del ();             // releasing memory

    return 0;
}
```

OUTPUT:



```
FAIZAN CHOUDHARY
20BCS021

First Come First Serve Scheduling Algorithm

Enter the number of processes: 6

Enter process names: p1 p2 p3 p4 p5 p6

Enter burst time for each process: 3 1 2 1 2 3

Enter arrival time for each process: 5 7 6 1 1 8
```



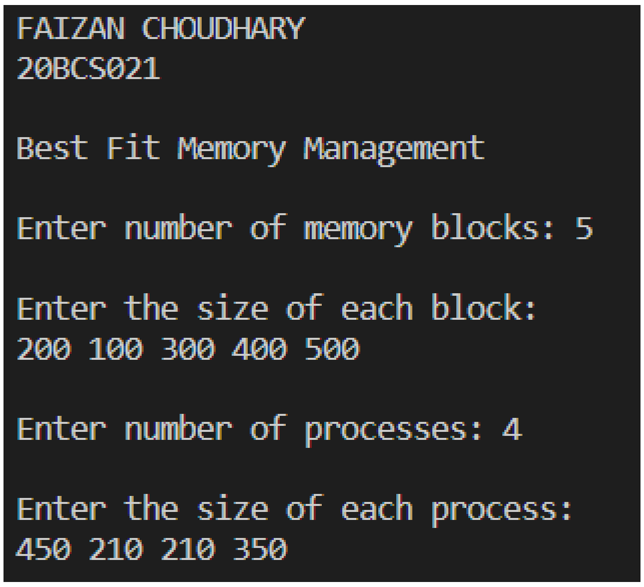
```
Process | Burst Time | Arrival Time | Completion Time | Waiting Time | Turnaround Time | Response Time
____________________________________________________________________________________________________

   p4        1            1               2               0               1               0
   p5        2            1               4               1               3               1
   p1        3            5               8               0               3               0
   p3        2            6              10               2               4               2
   p2        1            7              11               3               4               3
   p6        3            8              14               3               6               3
____________________________________________________________________________________________________


Average Completion time: 8.17
Average Waiting time: 1.50
Average Turnaround time: 3.50
Average Response time: 1.50

Gantt chart:
|  |p4| p5 |  |  p1  | p3 |p2|  p6  |
0  1  2    4  5      8   10 11      14
```

3. Write a program to implement the Best fit memory management algorithm. Program should take input total no. of memory block ,their sizes , process name and process size. Output of program should give the details about memory allocated to process with fragmentation detail.

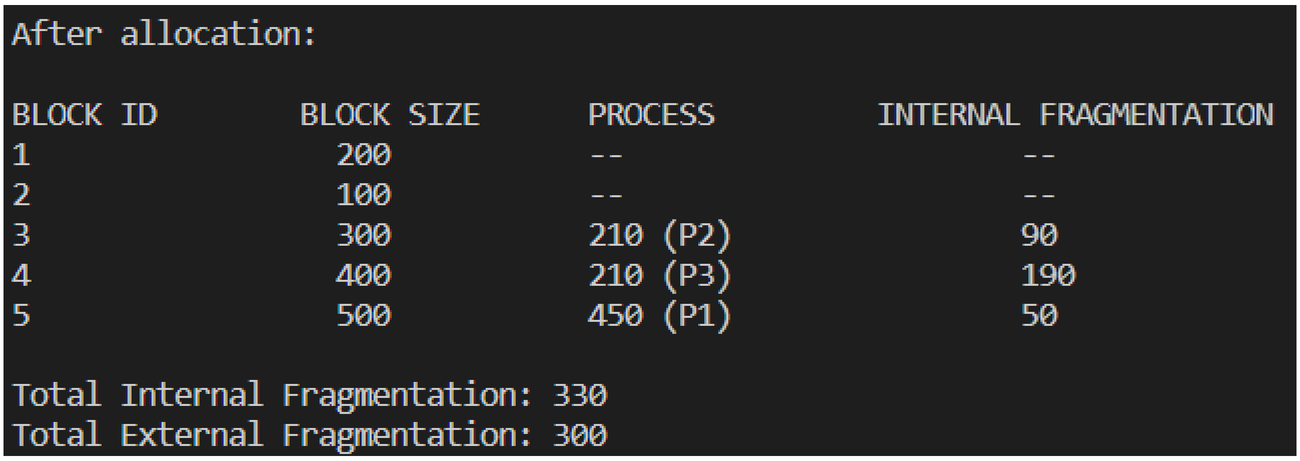
CODE: (code pasted in this format for readability)

```
#include <iostream>
#include <limits.h>
using namespace std;
int n, no;
// array to store process indices for each block index
int allocation_block[100] = {-1};
int totIntFrag=0, totExtFrag=0;
// temp array to store size of blocks for display
int temp[100];
// array to store internal fragmentation of each block
int intFrag[100] = {0};
// array to store the occupancy status of each block
bool occupied_block[100] = {false};
// counter to keep track of allocated processes
int counter=0;


void display (int *s_b, int *s_p) {
    cout<<"\nAfter allocation:\n";
    cout<<"\nBLOCK ID\tBLOCK SIZE\tPROCESS\t\tINTERNAL FRAGMENTATION\n";
    for (int i=0; i<n; i++) {
        cout<<i+1<<"\t\t  "<<temp[i]<<"\t\t";
        // if block is actually allocated a process
        if (occupied_block[i] == false || allocation_block[i] == -1)
            cout<<"--\t\t\t--";
        else if (allocation_block[i] != -1) {
            cout<<s_p[allocation_block[i]]<<" (P"<<allocation_block[i] + 1<<")\t\t";
            cout<<intFrag[i];
        }
        cout<<endl;
    }
    cout<<"\nTotal Internal Fragmentation: "<<totIntFrag;
    cout<<"\nTotal External Fragmentation: "<<totExtFrag<<endl<<endl;

}

void bestFit (int *s_b, int *s_p) {
    for (int i=0; i<n; i++)
        temp[i] = s_b[i];

    for (int i=0; i<no; i++) {
        // to store the index of the best fit
        int idx = -1;
        for (int j=0; j<n; j++) {
            if (s_b[j] >= s_p[i] && (idx == -1 || s_b[idx] > s_b[j]) && occupied_block[j]
== false)
                idx = j;
        }

        // for a successful best fit
        if (idx != -1) {
            counter++;
            allocation_block[idx] = i;
            occupied_block[idx] = true;
            intFrag[idx] = s_b[idx] - s_p[i];
            s_b[idx] -= s_p[i];
        }
    }

    for (int i=0; i<n; i++) {
        // cout<<allocation_block[i]<<endl;
        if (occupied_block[i] == true)
            totIntFrag += intFrag[i];
        if (occupied_block[i] == false && counter < no)
            totExtFrag += s_b[i];
    }
}

int main() {
    cout<<"\nFAIZAN CHOUDHARY\n20BCS021\n";
    cout<<"\nBest Fit Memory Management\n";

    cout<<"\nEnter number of memory blocks: ";
    cin>>n;

    int size_blocks[100];
    cout<<"\nEnter the size of each block:\n";
    for (int i=0; i<n; i++)
        cin>>size_blocks[i];

    cout<<"\nEnter number of processes: ";
    cin>>no;

    int size_processes[100];
    cout<<"\nEnter the size of each process:\n";
    for (int i=0; i<no; i++)
        cin>>size_processes[i];

    bestFit (size_blocks, size_processes);
    display (size_blocks, size_processes);
    return 0;
}
```

OUTPUT:



```
FAIZAN CHOUDHARY
20BCS021

Best Fit Memory Management

Enter number of memory blocks: 5

Enter the size of each block:
200 100 300 400 500

Enter number of processes: 4

Enter the size of each process:
450 210 210 350
```



```
After allocation:

BLOCK ID        BLOCK SIZE      PROCESS         INTERNAL FRAGMENTATION
1                 200           --                      --
2                 100           --                      --
3                 300           210 (P2)                90
4                 400           210 (P3)                190
5                 500           450 (P1)                50

Total Internal Fragmentation: 330
Total External Fragmentation: 300
```