Logo

Description automatically generated

JAMIA MILLIA ISLAMIA

OS Lab File 2022-23

Submitted by: Submitted to:

Name: **BUSHRA SHAHZAD Mr. Jawahar Lal** Course: **B. TECH Dr. Shahzad Alam**

Branch: **Computer Engineering Dr. Waseem Ahmed**

Semester: **IV**

Roll No.: **21BCS046**

Subject: **OS Lab**

INDEX

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S. No | Program Name | Date | Remarks | Signature |
| 1. | Write a menu driven program in to implement priority queue scheduling algorithm using linked list. | 23rd Jan,2023 |  |  |
| 2. | Write a program to implement the shortest job first non-pre-emptive scheduling algorithm and find TAT, WT, CT, and RT. | 6th Feb,2023 |  |  |
| 3. | Write a program to implement the shortest job first pre-emptive scheduling algorithm and find TAT, WT, CT, and RT. | 13st Feb,2023 |  |  |
| 4. | Write a program to implement the round robin scheduling algorithm and find TAT, WT, CT, and RT. | 13th March,2023 |  |  |
| 5. | WAP to implement memory allocation methods, first fit, best fit, and worst fit. | 27th March,2023 |  |  |
| 6. | WAP to implement disk scheduling algos- FCFS, SSTF, SCAN, CSCAN, LOOK, CLOOK. | 1st May,2023 |  |  |

PRoGRAM 1 - Priority scheduling

CODE:

#include <stdio.h>

#include <stdlib.h>

struct node

{

char process\_name[4];

int process\_priority;

int process\_burst\_time;

struct node \*next;

};

struct node \*head = NULL;

void insert\_process()

{

struct node \*temp, \*ptr;

int time\_cal = 0;

temp = (struct node \*)malloc(sizeof(struct node));

printf("Enter the process name:");

getchar();

gets(temp->process\_name);

printf("Enter process priority : ");

scanf("%d", &temp->process\_priority);

printf("Enter process burst time : ");

scanf("%d", &temp->process\_burst\_time);

time\_cal = temp->process\_burst\_time;

if (head == NULL || temp->process\_priority < head->process\_priority)

{

temp->next = head;

head = temp;

}

else

{

ptr = head;

while (ptr->next != NULL && temp->process\_priority > ptr->process\_priority)

{

ptr = ptr->next;

}

temp->next = ptr->next;

ptr->next = temp;

}

}

void execute\_process()

{

struct node \*temp;

temp = head;

int time\_total = 0;

printf("Process Name\tPriority\tBT\tStart Time\tEnd Time\n");

while (temp != NULL)

{

printf("%s\t\t", temp->process\_name);

printf("%d\t\t", temp->process\_priority);

printf("%d\t", temp->process\_burst\_time);

printf("%d\t\t", time\_total);

time\_total += temp->process\_burst\_time;

printf("%d\n", time\_total);

temp = temp->next;

}

}

int main()

{

int choice;

printf("Bushra Shahzad-21BCS046");

while (1)

{

printf("\nEnter\n1. Insert a process.\n2. Execute all processes.\n3. Exit!\n");

scanf("%d", &choice);

switch (choice)

{

case 1:

insert\_process();

execute\_process();

break;

case 2:

execute\_process();

head = NULL;

printf("\nProcesses Executed!\n");

break;

case 3:

exit(0);

break;

default:

break;

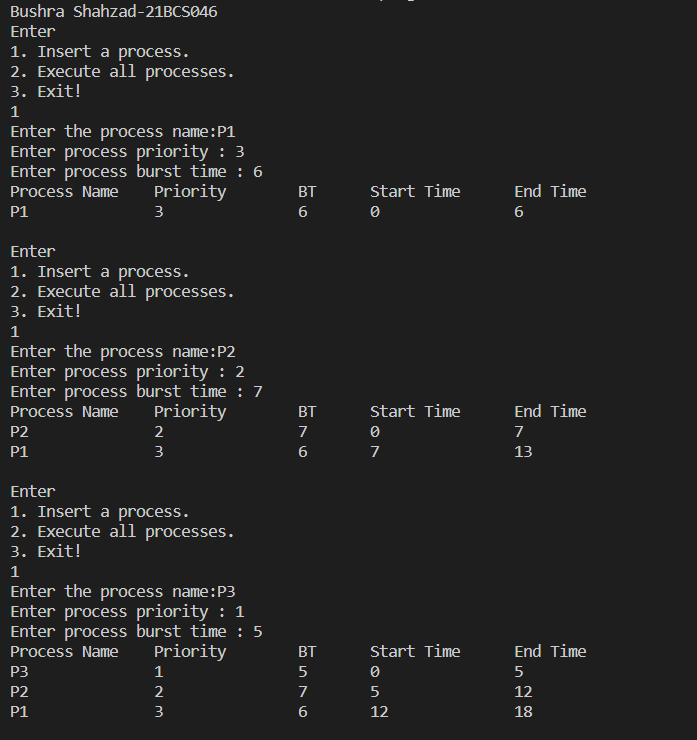
}

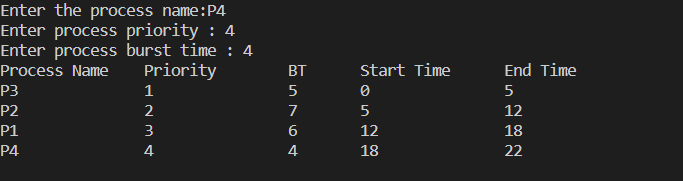
}

return 0;

}

OUTPUT:





PRoGRAM 2 – Shortest job First (NON PRE-EMPTIVE)

CODE:

#include <stdio.h>

#include <limits.h>

#include <stdlib.h>

struct node

{

int id;

int arrival\_time;

int burst\_time;

int completion\_time;

int waiting\_time;

int turn\_around\_time;

int st;

} p[100];

int main()

{

printf("\nBushra Shahzad-21BCS046\n");

int n, t, current\_time = 0;

int proecess\_executed[20] = {0};

printf("Enter the number of processes: ");

scanf("%d", &n);

t = n;

float atat = 0, awt = 0, act = 0;

for (int i = 0; i < n; i++)

{

printf("P%d : ", i + 1);

printf("\nBurst time : ");

scanf("%d", &p[i].burst\_time);

printf("Arrival time : ");

scanf("%d", &p[i].arrival\_time);

p[i].id = i + 1;

}

while (t)

{

int min\_index = -1;

int min\_burst\_time = INT\_MAX;

for (int i = 0; i < n; i++)

{

if (p[i].arrival\_time <= current\_time && proecess\_executed[i] == 0)

{

if (p[i].burst\_time < min\_burst\_time)

{

min\_burst\_time = p[i].burst\_time;

min\_index = i;

}

if (p[i].burst\_time == min\_burst\_time)

{

if (p[i].arrival\_time < p[min\_index].arrival\_time)

{

min\_burst\_time = p[i].burst\_time;

min\_index = i;

}

}

}

}

if (min\_index == -1)

{

current\_time++;

}

else

{

p[min\_index].st = current\_time;

p[min\_index].completion\_time = p[min\_index].st + p[min\_index].burst\_time;

p[min\_index].turn\_around\_time = p[min\_index].completion\_time - p[min\_index].arrival\_time;

p[min\_index].waiting\_time = p[min\_index].turn\_around\_time - p[min\_index].burst\_time;

atat += p[min\_index].turn\_around\_time;

awt += p[min\_index].waiting\_time;

act += p[min\_index].completion\_time;

proecess\_executed[min\_index] = 1;

current\_time = p[min\_index].completion\_time;

t--;

}

}

printf("\nProcess No.\tBT\tAT\tCT\tTAT\tWT\n");

for (int i = 0; i < n; i++)

printf("P%d\t\t%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].burst\_time, p[i].arrival\_time, p[i].completion\_time, p[i].turn\_around\_time, p[i].waiting\_time);

printf("\n");

printf("\nAverage Turn Around Time = %0.2f", atat / n);

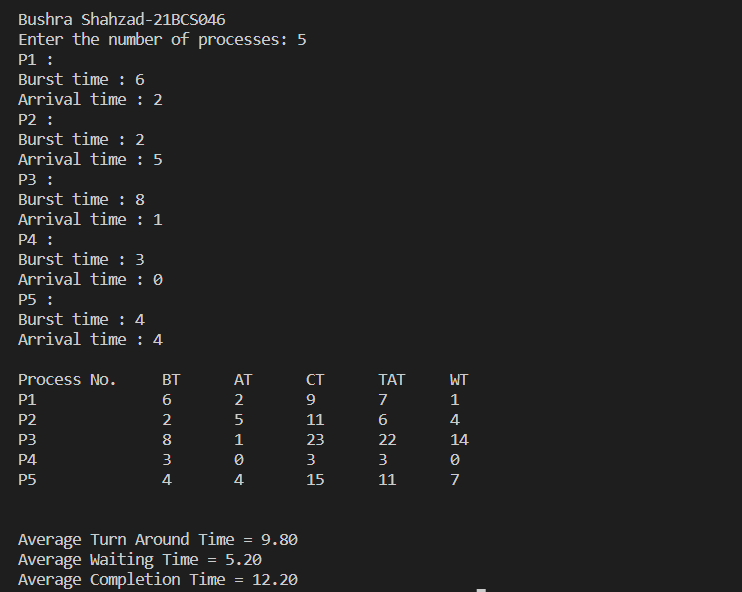
printf("\nAverage Waiting Time = %0.2f ", awt / n);

printf("\nAverage Completion Time = %0.2f", act / n);

return 0;

}

OUTPUT:



Program 3 - Shortest job First ( PRE-EMPTIVE)

CODE:

#include <iostream>

#include <algorithm>

using namespace std;

struct process

{

char process\_name;

int burst\_time;

int arrival\_time;

int response\_time = 0;

int completion\_time = 0;

int waiting\_time = 0;

} a[100], b[100], c[100];

bool btimeSort(process a, process b)

{

return a.burst\_time < b.burst\_time;

}

bool atimeSort(process a, process b)

{

return a.arrival\_time < b.arrival\_time;

}

int k = 0, f = 0, r = 0;

void display(int m, int current\_time)

{

int n = m, q;

sort(a, a + n, atimeSort);

int total\_time = 0, i;

int j, total\_array[n];

int alltime = 0;

bool counter = false;

for (i = 0; i < n; i++)

{

alltime += a[i].burst\_time;

}

alltime += a[0].arrival\_time;

for (i = 0; total\_time <= alltime;)

{

j = i;

while (a[j].arrival\_time <= total\_time && j != n)

{

b[r] = a[j];

j++;

r++;

}

if (r == f)

{

c[k].process\_name = 'i';

c[k].burst\_time = a[j].arrival\_time;

c[k].arrival\_time = total\_time;

total\_time += c[k].burst\_time;

k++;

continue;

}

i = j;

if (counter == true)

{

sort(b + f, b + r, btimeSort);

}

j = f;

if (b[j].burst\_time > current\_time)

{

c[k] = b[j];

c[k].burst\_time = current\_time;

k++;

b[j].burst\_time = b[j].burst\_time - current\_time;

total\_time += current\_time;

counter = true;

for (q = 0; q < n; q++)

{

if (b[j].process\_name != a[q].process\_name)

{

a[q].waiting\_time += current\_time;

}

}

}

else

{

c[k] = b[j];

k++;

f++;

total\_time += b[j].burst\_time;

counter = false;

for (q = 0; q < n; q++)

{

if (b[j].process\_name != a[q].process\_name)

{

a[q].waiting\_time += b[j].burst\_time;

}

}

}

if (f == r && i >= n)

break;

}

total\_array[i] = total\_time;

total\_time += a[i].burst\_time;

for (i = 0; i < k - 1; i++)

{

if (c[i].process\_name == c[i + 1].process\_name)

{

c[i].burst\_time += c[i + 1].burst\_time;

for (j = i + 1; j < k - 1; j++)

c[j] = c[j + 1];

k--;

i--;

}

}

int rtime = 0;

for (j = 0; j < n; j++)

{

rtime = 0;

for (i = 0; i < k; i++)

{

if (c[i].process\_name == a[j].process\_name)

{

a[j].response\_time = rtime;

break;

}

rtime += c[i].burst\_time;

}

}

rtime = 0;

for (i = 0; i < k; i++)

{

if (i != k)

rtime += c[i].burst\_time;

for (j = 0; j < n; j++)

{

if (a[j].process\_name == c[i].process\_name)

a[j].completion\_time = rtime;

}

}

rtime = 0;

for (i = 0; i < k + 1; i++)

{

// cout << rtime << "\t";

total\_array[i] = rtime;

rtime += c[i].burst\_time;

}

float averageWaitingTime = 0, averageResponseTime = 0, averageTAT = 0, averagect = 0;

cout << "\nGantt Chart\n ";

rtime = 0;

for (i = 0; i < k; i++)

{

if (i != k)

cout << "| " << 'P' << c[i].process\_name << " ";

rtime += c[i].burst\_time;

for (j = 0; j < n; j++)

{

if (a[j].process\_name == c[i].process\_name)

a[j].completion\_time = rtime;

}

}

printf("\n\n");

cout << "P.Name AT\tBT\tCT\tTAT\tWT\tRT\n";

for (i = 0; i < m && a[i].process\_name != 'i'; i++)

{

if (a[i].process\_name == '\0') break;

cout << 'P' << a[i].process\_name << "\t";

cout << a[i].arrival\_time << "\t";

cout << a[i].burst\_time << "\t";

cout << a[i].completion\_time << "\t";

cout << a[i].completion\_time - a[i].arrival\_time<< "\t";

averageTAT += a[i].completion\_time - a[i].arrival\_time;

cout << a[i].waiting\_time + a[i].completion\_time - rtime << "\t";

averageWaitingTime += a[i].waiting\_time + a[i].completion\_time - rtime;

cout << a[i].response\_time << "\t";

averageResponseTime += a[i].response\_time;

cout << "\n";

}

for (int i = 0; i < m; i++)

{

averagect += a[i].completion\_time;

}

cout << "Average Completion Time " << (float)averagect / (float)n << endl;

cout << "Average TA time: " << (float)averageTAT / (float)n << endl;

cout << "Average Waiting time: " << (float)averageWaitingTime / (float)n << endl;

cout << "Average Response time: " << (float)averageResponseTime / (float)n << endl;

}

int main()

{

int m, choice, i, current\_time;

cout << "\nBushra Shahzad-21BCS046\n";

cout << "Enter number of processes: "; cin >> m;

cout << "Enter process, arrival time, Burst Time!\n";

for (i = 0; i < m; i++)

{

cout << "Enter process id: ";

cin >> a[i].process\_name;

cout << "Enter arrival time: ";

cin >> a[i].arrival\_time;

cout << "Enter burst time: ";

cin >> a[i].burst\_time;

a[i].waiting\_time = -a[i].arrival\_time + 1;

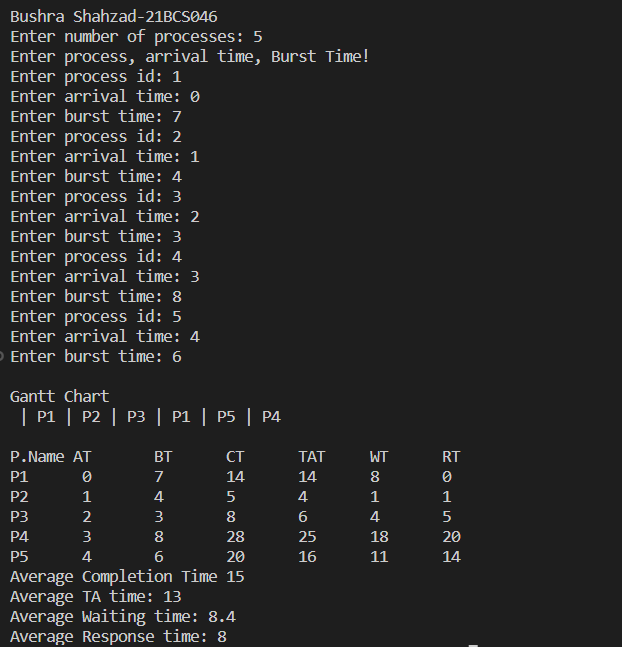
}

display(m, 1);

return 0;

}

OUTPUT:



Program 4- Round robin

CODE:

#include <iostream>

#include <algorithm>

#include <stdlib.h>

using namespace std;

struct node

{

char pname;

int burst\_time;

int arrival\_time;

int response\_time = 0;

int completion\_time = 0;

int waiting\_time = -1;

} a[100], b[100], c[100];

bool btimeSort(node a, node b)

{

return a.burst\_time < b.burst\_time;

}

bool atimeSort(node a, node b)

{

return a.arrival\_time < b.arrival\_time;

}

int k = 0, f = 0, r = 0;

void disp(int nop, int qt)

{

int n = nop, q;

sort(a, a + n, atimeSort);

int ttime = 0, i;

int j, tArray[n];

int alltime = 0;

bool moveLast = false;

for (i = 0; i < n; i++)

{

alltime += a[i].burst\_time;

}

alltime += a[0].arrival\_time;

for (i = 0; ttime <= alltime;)

{

j = i;

while (a[j].arrival\_time <= ttime && j != n)

{

b[r] = a[j];

j++;

r++;

}

if (r == f)

{

c[k].pname = 'i';

c[k].burst\_time = a[j].arrival\_time - ttime;

c[k].arrival\_time = ttime;

ttime += c[k].burst\_time;

k++;

continue;

}

i = j;

if (moveLast == true)

{

b[r] = b[f];

f++;

r++;

}

j = f;

if (b[j].burst\_time > qt)

{

c[k] = b[j];

c[k].burst\_time = qt;

k++;

b[j].burst\_time = b[j].burst\_time - qt;

ttime += qt;

moveLast = true;

for (q = 0; q < n; q++)

{

if (b[j].pname != a[q].pname)

{

a[q].waiting\_time += qt;

}

}

}

else

{

c[k] = b[j];

k++;

f++;

ttime += b[j].burst\_time;

moveLast = false;

for (q = 0; q < n; q++)

{

if (b[j].pname != a[q].pname)

{

a[q].waiting\_time += b[j].burst\_time;

}

}

}

if (f == r && i >= n)

break;

}

tArray[i] = ttime;

ttime += a[i].burst\_time;

int rtime = 0;

for (j = 0; j < n && j < 6; j++)

{

rtime = 0;

for (i = 0; i < k; i++)

{

if (c[i].pname == a[j].pname)

{

a[j].response\_time = rtime;

break;

}

rtime += c[i].burst\_time;

}

}

float averageWaitingTime = 0;

float averageResponseTime = 0;

float averageTAT = 0;

float averageCT=0;

printf("\nGantt Chart\n");

rtime = 0;

for (i = 0; i < k + 1 && i < 20; i++)

{

if (i != k)

printf("| %c ", c[i].pname);

rtime += c[i].burst\_time;

for (j = 0; j < 6; j++)

{

if (a[j].pname == c[i].pname)

a[j].completion\_time = rtime;

}

}

printf("\n");

rtime = 0;

for (i = 0; i < k + 1 && i < 20; i++)

{

printf(" %d ", rtime);

tArray[i] = rtime;

rtime += c[i].burst\_time;

}

printf("\n");

printf("\n");

printf("P.Name AT\tBT\tCT\tTAT\tWT\tRT\n");

for (i = 0; i < 6 && i < nop && a[i].pname != 'i'; i++)

{

if (a[i].pname == '\0')

break;

printf("%d\t", a[i].pname);

printf("%d\t", a[i].arrival\_time);

printf("%d\t", a[i].burst\_time);

printf("%d\t", a[i].completion\_time);

averageCT+=a[i].completion\_time;

printf("%d\t", a[i].waiting\_time + a[i].completion\_time - rtime + a[i].burst\_time);

averageTAT += a[i].waiting\_time + a[i].completion\_time - rtime + a[i].burst\_time;

printf("%d\t", a[i].waiting\_time + a[i].completion\_time - rtime);

averageWaitingTime += a[i].waiting\_time + a[i].completion\_time - rtime;

printf("%d", a[i].response\_time);

averageResponseTime += a[i].response\_time;

printf("\n");

}

cout << "\nAverage Completion time: " << (float)averageCT / (float)n << endl;

cout << "\nAverage TurnAround time: " << (float)averageTAT / (float)n << endl;

cout << "Average Waiting time: " << (float)averageWaitingTime / (float)n << endl;

cout << "Average Response time: " << (float)averageResponseTime / (float)n << endl;

}

int main()

{

int n, qt;

printf("\nBushra Shahzad-21BCS046\n");

printf("Enter number of processes: ");

scanf("%d", &n);

printf("Enter time quantum: ");

scanf("%d", &qt);

printf("\nEnter the process\n");

for (int i = 0; i < n; i++)

{

printf("Enter process id: ");

cin >> a[i].pname;

printf("Enter arrival time: ");

scanf("%d", &a[i].arrival\_time);

printf("Enter burst time: ");

scanf("%d", &a[i].burst\_time);

a[i].waiting\_time = -a[i].arrival\_time;

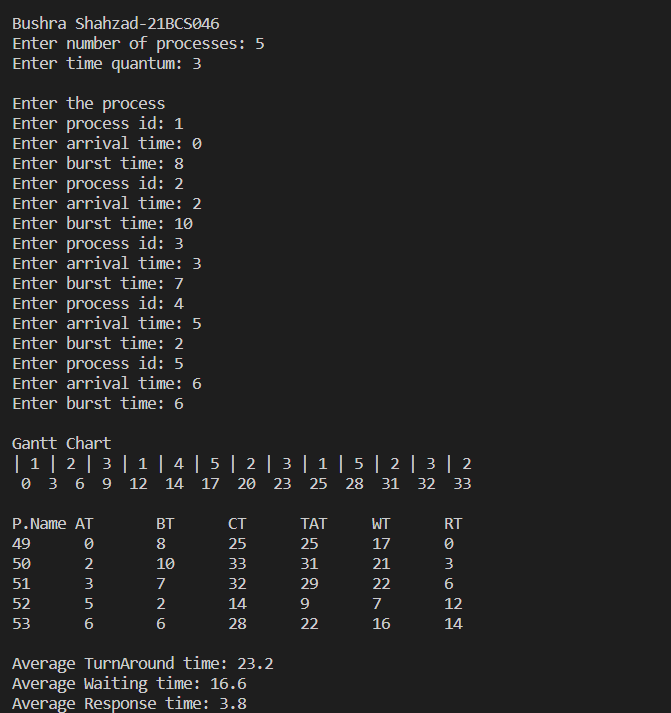
}

disp(n, qt);

return 0;

}

OUTPUT:



Program 5 – MEMORY ALLOCATION

CODE:

#include <stdio.h>

#include <stdlib.h>

struct mem

{

int size;

int proc;

int procSize;

} a[50];

struct node

{

int pro;

int size;

struct node \*next;

} \*head = NULL, \*tail = NULL;

void display(int intFrag, int k)

{

int extFrag = 0;

printf("\nP-id\tPro-Size\tLeft-Memory\n");

for (int i = 0; i < k; i++)

{

if (a[i].proc == 0)

{

extFrag += a[i].size;

printf("%d\t 0\t\t%d", a[i].proc, a[i].size);

}

else

{

printf("%d\t%d\t\t%d", a[i].proc, a[i].procSize, a[i].size);

}

printf("\n");

}

printf("\nInternal fragmentation : %d", intFrag);

printf("\nExternal fragmentation : %d", extFrag);

}

void main()

{

printf("\nBushra Shahzad-21BCS046\n");

int c = 0, ch, p = 0;

int intFrag = 0;

struct node \*temp;

printf("\nEnter the number of memory blocks : \n");

int k;

scanf("%d", &k);

printf("\nEnter the size of each memory block : \n");

for (int i = 0; i < k; i++)

{

scanf("%d", &a[i].size);

a[i].proc = 0;

}

while (1)

{

printf("\n1. Enter process.\n2. Best fit.\n3. Worst fit.\n4. First fit.\n5. Change memory blocks. \n6. Quit!\n");

scanf("%d", &ch);

switch (ch)

{

case 1:

{

struct node \*newNode = (struct node \*)malloc(sizeof(struct node));

printf("\nEnter the size of the process : \n");

scanf("%d", &newNode->size);

p++;

newNode->pro = p;

newNode->next = NULL;

if (head == NULL)

{

head = newNode;

tail = newNode;

}

else

{

tail->next = newNode;

tail = newNode;

}

break;

}

case 2:

{

// best fit

temp = head;

int diff;

int p;

printf("\nBest Fit : \n");

while (temp != NULL)

{

p = -1;

diff = -1;

for (int i = 0; i < k; i++)

{

if (a[i].proc == 0)

{

if (temp->size <= a[i].size)

{

if (a[i].size - temp->size <= diff || diff == -1)

{

diff = a[i].size - temp->size;

p = i;

}

}

}

}

if (p == -1)

{

printf("Process id %d of size %d could not be allocated!", temp->pro, temp->size);

temp = temp->next;

}

else

{

a[p].proc = temp->pro;

a[p].procSize = temp->size;

intFrag += a[p].size - a[p].procSize;

a[p].size = a[p].size - a[p].procSize;

temp = temp->next;

}

}

head = NULL;

tail = NULL;

display(intFrag, k);

break;

}

case 3:

{

// worst fit

temp = head;

int diff;

int p;

int i;

printf("\nWorst Fit : \n");

while (temp != NULL)

{

p = -1;

diff = -1;

for (i = 0; i < k; i++)

{

if (a[i].proc == 0)

{

if (temp->size <= a[i].size)

{

if (a[i].size - temp->size >= diff)

{

diff = a[i].size - temp->size;

p = i;

}

}

}

}

if (p == -1)

{

printf("Process id %d of size %d could not be allocated!", temp->pro, temp->size);

temp = temp->next;

}

else

{

a[p].proc = temp->pro;

a[p].procSize = temp->size;

intFrag += a[p].size - a[p].procSize;

a[p].size = a[p].size - a[p].procSize;

temp = temp->next;

}

}

head = NULL;

tail = NULL;

display(intFrag, k);

break;

}

case 4:

{

// first fit

int p;

temp = head;

printf("\nFirst Fit : \n");

while (temp != NULL)

{

p = -1;

for (int i = 0; i < k; i++)

{

if (a[i].proc == 0)

{

if (temp->size <= a[i].size)

{

a[i].proc = temp->pro;

a[i].procSize = temp->size;

p = 1;

intFrag += a[i].size - a[i].procSize;

a[i].size = a[i].size - a[i].procSize;

break;

}

}

}

if (p == -1)

{

printf("Process id %d of size %d could not be allocated!", temp->pro, temp->size);

temp = temp->next;

}

else

{

temp = temp->next;

}

}

head = NULL;

tail = NULL;

display(intFrag, k);

break;

}

case 5:

{

printf("\nEnter the number of memory blocks : \n");

scanf("%d", &k);

printf("\nEnter the size of each memory block : \n");

for (int i = 0; i < k; i++)

{

scanf("%d", &a[i].size);

a[i].proc = 0;

}

}

case 6:

{

exit(0);

}

default:

{

printf("Invalid Input");

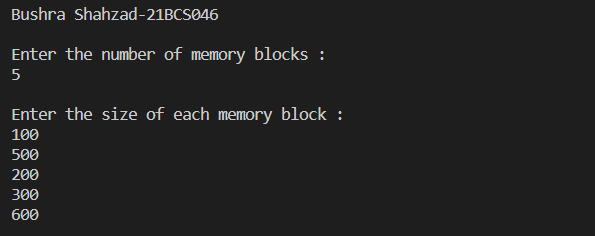
}

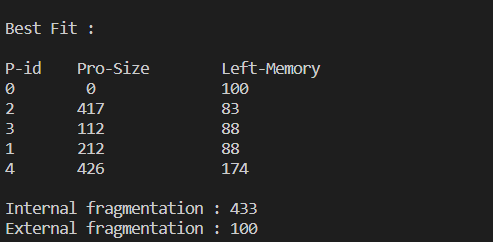
}

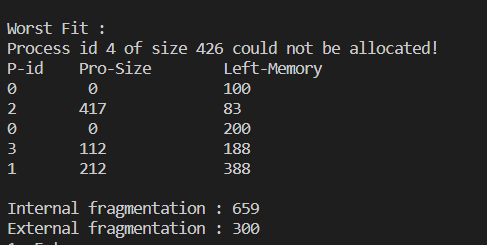
}

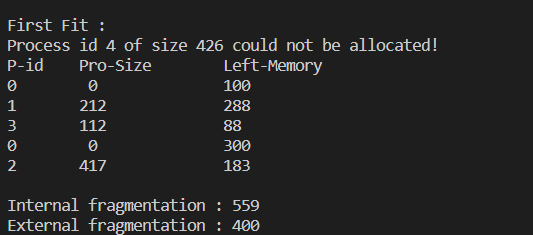
}

OUTPUT:









Program 6 – DISK SCHEDULING

CODE:

#include <stdio.h>

#include <stdlib.h>

void fcfs(int \*a, int n)

{

int curr = 53, headmov = 0;

printf("\nSeek sequence :\n");

for (int i = 0; i <= n; i++)

{

printf("%d -> ", curr);

if (i != n)

headmov += abs(\*(a + i) - curr);

curr = \*(a + i);

}

printf("\nTotal head movements : %d", headmov);

}

void sstf(int \*a, int n)

{

int temp[n], i, headmov = 0, j, k, curr = 53, b;

char dir = 'L';

for (i = 0; i < n; i++)

{

temp[i] = \*(a + i);

}

// Sorting

for (i = 0; i < n; ++i)

{

for (j = i + 1; j < n; ++j)

{

if (temp[i] > temp[j])

{

b = temp[i];

temp[i] = temp[j];

temp[j] = b;

}

}

}

for (i = 0; i < n; i++)

{

if (temp[i] > 53)

{

j = i;

k = i - 1;

break;

}

else if (temp[i] == 53)

{

j = i + 1;

k = i - 1;

break;

}

}

printf("\nSeek Sequence:\n53 -> ");

while (k >= 0 || j < n)

{

if (k >= 0 && j < n)

{

if (abs(curr - temp[j]) > abs(curr - temp[k]))

{

headmov += abs(curr - temp[k]);

curr = temp[k];

k = k - 1;

dir = 'L';

printf("%d -> ", curr);

}

else if (abs(curr - temp[j]) < abs(curr - temp[k]))

{

headmov += abs(curr - temp[j]);

curr = temp[j];

j = j + 1;

dir = 'R';

printf("%d -> ", curr);

}

else

{

if (dir == 'L')

{

headmov += abs(curr - temp[k]);

curr = temp[k];

k = k - 1;

dir = 'L';

printf("%d -> ", curr);

}

else

{

headmov += abs(curr - temp[k]);

curr = temp[k];

k = k - 1;

dir = 'L';

printf("%d -> ", curr);

}

}

}

else if (k < 0 && j < n)

{

headmov += abs(curr - temp[j]);

curr = temp[j];

j = j + 1;

dir = 'R';

printf("%d -> ", curr);

}

else if (k >= 0 && j >= n)

{

headmov += abs(curr - temp[k]);

curr = temp[k];

k = k - 1;

dir = 'L';

printf("%d -> ", curr);

}

}

printf("\nTotal head movements : = %d", headmov);

}

void scan(int \*a, int n)

{

int temp[n], i, headmov = 0, j, k, curr = 53, b;

char dir = 'L';

for (i = 0; i < n; i++)

{

temp[i] = \*(a + i);

}

// sorting

for (i = 0; i < n; ++i)

{

for (j = i + 1; j < n; ++j)

{

if (temp[i] > temp[j])

{

b = temp[i];

temp[i] = temp[j];

temp[j] = b;

}

}

}

for (i = 0; i < n; i++)

{

if (temp[i] > 53)

{

j = i;

k = i - 1;

break;

}

else if (temp[i] == 53)

{

j = i + 1;

k = i - 1;

break;

}

}

printf("\nSeek Sequence:\n53 -> ");

for (i = k; i >= 0; i--)

{

headmov += abs(curr - temp[i]);

curr = temp[i];

printf("%d -> ", curr);

}

if (curr > 0)

{

headmov += curr;

curr = 0;

printf("%d -> ", curr);

}

for (i = j; i < n; i++)

{

headmov += abs(curr - temp[i]);

curr = temp[i];

printf("%d -> ", curr);

}

printf("\nTotal head movements : = %d", headmov);

}

void cscan(int \*a, int n)

{

int temp[n], i, headmov = 0, j, k, curr = 53, b;

char dir = 'L';

for (i = 0; i < n; i++)

{

temp[i] = \*(a + i);

printf(" %d", temp[i]);

}

// Sorting

for (i = 0; i < n; ++i)

{

for (j = i + 1; j < n; ++j)

{

if (temp[i] > temp[j])

{

b = temp[i];

temp[i] = temp[j];

temp[j] = b;

}

}

}

for (i = 0; i < n; i++)

{

if (temp[i] > 53)

{

j = i;

k = i - 1;

break;

}

else if (temp[i] == 53)

{

j = i + 1;

k = i - 1;

break;

}

}

printf("\nSeek Sequence:\n53 -> ");

for (i = k; i >= 0; i--)

{

headmov += abs(curr - temp[i]);

curr = temp[i];

printf("%d -> ", curr);

}

if (curr > 0)

{

headmov += curr;

curr = 0;

printf("%d -> ", curr);

headmov += 255;

curr = 255;

printf("%d -> ", curr);

}

for (i = n - 1; i >= j; i--)

{

headmov += abs(curr - temp[i]);

curr = temp[i];

printf("%d -> ", curr);

}

printf("\nTotal head movements : %d", headmov);

}

void look(int \*a, int n)

{

int temp[n], i, headmov = 0, j, k, curr = 53, b;

char dir = 'L';

for (i = 0; i < n; i++)

{

temp[i] = \*(a + i);

printf(" %d", temp[i]);

}

// Sorting

for (i = 0; i < n; ++i)

{

for (j = i + 1; j < n; ++j)

{

if (temp[i] > temp[j])

{

b = temp[i];

temp[i] = temp[j];

temp[j] = b;

}

}

}

for (i = 0; i < n; i++)

{

if (temp[i] > 53)

{

j = i;

k = i - 1;

break;

}

else if (temp[i] == 53)

{

j = i + 1;

k = i - 1;

break;

}

}

printf("\nSeek Sequence:\n53 -> ");

for (i = k; i >= 0; i--)

{

headmov += abs(curr - temp[i]);

curr = temp[i];

printf("%d -> ", curr);

}

for (i = j; i < n; i++)

{

headmov += abs(curr - temp[i]);

curr = temp[i];

printf("%d -> ", curr);

}

printf("\nTotal head movements : = %d", headmov);

}

void clook(int \*a, int n)

{

int temp[n], i, headmov = 0, j, k, curr = 53, b;

char dir = 'L';

for (i = 0; i < n; i++)

{

temp[i] = \*(a + i);

printf(" %d", temp[i]);

}

// Sorting

for (i = 0; i < n; ++i)

{

for (j = i + 1; j < n; ++j)

{

if (temp[i] > temp[j])

{

b = temp[i];

temp[i] = temp[j];

temp[j] = b;

}

}

}

for (i = 0; i < n; i++)

{

if (temp[i] > 53)

{

j = i;

k = i - 1;

break;

}

else if (temp[i] == 53)

{

j = i + 1;

k = i - 1;

break;

}

}

printf("\nSeek Sequence:\n53 -> ");

for (i = k; i >= 0; i--)

{

headmov += abs(curr - temp[i]);

curr = temp[i];

printf("%d -> ", curr);

}

headmov += abs(curr - temp[n - 1]);

curr = temp[n - 1];

printf("%d -> ", curr);

for (i = n - 2; i >= j; i--)

{

headmov += abs(curr - temp[i]);

curr = temp[i];

printf("%d -> ", curr);

}

printf("\nTotal head movements : = %d", headmov);

}

int main()

{

printf("\nBushra Shahzad-21BCS046\n");

int choice, n;

int array[1000];

while (1)

{

printf("\n\n0.Input track numbers between 0 to 256!");

printf("\n1. FCFS");

printf(" 2. SSTF");

printf(" 3. SCAN");

printf(" 4. CSCAN");

printf(" 5. LOOK");

printf(" 6. CLOOK");

printf(" 7. QUIT!\n");

scanf("%d", &choice);

switch (choice)

{

case 0:

printf("Enter no of processes:\n");

scanf("%d", &n);

printf("Enter track nos.\n");

for (int i = 0; i < n; i++)

{

scanf("%d", &array[i]);

}

break;

case 1:

fcfs(array, n);

break;

case 2:

sstf(array, n);

break;

case 3:

scan(array, n);

break;

case 4:

cscan(array, n);

break;

case 5:

look(array, n);

break;

case 6:

clook(array, n);

break;

case 7:

exit(0);

default:

printf("\nInvalid input!\n");

break;

}

}

return 0;

}

OUTPUT:

