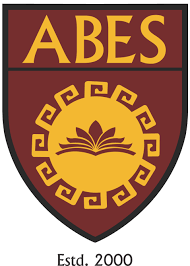
**OPERATING SYSTEM**

**LAB**

**(KCS451)**



HEY HOW N

**SUBMITTED BY**

**ROHIT KUMAR**

**1900320100131**

**SUBMITTED TO: NIDHI SINGH**

**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

ABES ENGINEERING COLLEGE

GHAZIABAD

INDEX

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Expt./ Job No.** | **Experiment / Job Details** | **Pre-Lab**  **Work** | **Implementation** | **Output** | **Viva** | **Total** |
| 1 | Program to Implement FCFS |  |  |  |  |  |
| 2 | Program to Implement SJF |  |  |  |  |  |
| 3 | Program to Implement Priority Scheduling |  |  |  |  |  |
| 4 | Program to Implement Round Robin |  |  |  |  |  |
| 5 | Program to Implement SRTF |  |  |  |  |  |
| 6 | Program to Implement Bankers Algorithms |  |  |  |  |  |
| 7 | Program to Implement Memory Allocation Scheme |  |  |  |  |  |
| 8 | Program to Implement Page Replacement Algorithms |  |  |  |  |  |
| 9 | Program to Implement Disk Scheduling Algorithms |  |  |  |  |  |

**Dated:** 14/04/2021

**Experiment No. – 01 (FCFS)**

**Objective:**

Calculate average waiting time and average turnaround time using FCFS scheduling algorithm using following input.

Design generic code that will run both input

**Program:**

// Program to Implement First Come First Serve Algorithm

// Rohit Kumar | 1900320100131

#include<stdio.h>

void main()

{

// STDIN

int n;

printf("No of Process : ");

scanf("%d",&n);

int AT[n],BT[n],CT[n],WT[n],TAT[n];

printf("Enter the AT & BT of Process\n");

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

{

printf("P[%d]\t",i);

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

}

// PROCESSING

float avgWT = 0.0;

float avgTAT = 0.0;

// Calculation for 1st Process

CT[0] = AT[0] + BT[0];

TAT[0] = CT[0] - AT[0];

WT[0] = TAT[0] - BT[0];

// Loop for Calculation

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

{

if(CT[i-1]>AT[i])// if CT of 1st Process is more then than Arrival of Second Process

CT[i] = CT[i-1] + BT[i];

else // if CT of 1st Process is less then than Arrival of Second Process

CT[i] = AT[i] + BT[i];

TAT[i] = CT[i] - AT[i];

WT[i] = TAT[i] - BT[i];

avgTAT += TAT[i];

avgWT += WT[i];

}

avgWT = avgWT/n;

avgTAT = avgTAT/n;

// STDOUT

printf("\n\tAT\tBT\tWT\tTAT\n");

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

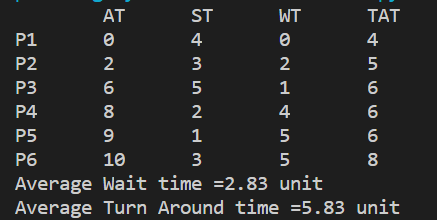
printf("P[%d]\t%d\t%d\t%d\t%d\n",i,AT[i],BT[i],WT[i],TAT[i]);

printf("\nAverage WT : %f Units\n",avgWT);

printf("Average TAT : %f Units",avgTAT);

}

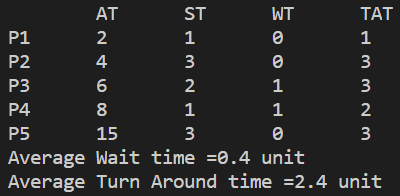
**Input/Output:**

**Q1:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process** | **Arrival Time** | **Burst Time** | **Waiting Time** | **TAT** |
| P1 | 0 | 4 | 0 | 4 |
| P2 | 2 | 3 | 2 | 5 |
| P3 | 6 | 5 | 1 | 6 |
| P4 | 8 | 2 | 4 | 6 |
| P5 | 9 | 1 | 5 | 6 |
| P6 | 10 | 3 | 5 | 8 |

Average waiting time = **2.8 Units.**

Average turnaround time = **5.83 Units.**

**Q2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Process** | **Arrival Time** | **Burst Time** | **Waiting Time** | **TAT** |
| P1 | 6 | 1 | 0 | 1 |
| P2 | 4 | 3 | 0 | 3 |
| P3 | 2 | 2 | 1 | 3 |
| P4 | 8 | 1 | 1 | 2 |
| P5 | 15 | 3 | 0 | 3 |

Average waiting time = **0.4 Units.**

Average turnaround time **= 2.4 Units.**

**Result:**

We have Successfully implemented FCFS.

**Dated:** 14/04/2021

**Experiment No. – 02 (SJF)**

**Objective:**

Calculate the average waiting time and average turnaround time using SJF (Non-Primitive Scheduling)

**Program:**

// Program to Implement Shortest Job First (SJF)

// Rohit Kumar | 1900320100131

#include<stdio.h>

void main()

{

// STDIN

int n;

printf("No of Process : ");

scanf("%d",&n);

int AT[n],BT[n],CT[n],WT[n],TAT[n];

printf("Enter the AT & BT of Process\n");

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

{

printf("P[%d]\t",i);

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

}

// PROCESSING

float AvgWT = 0.0;

float AvgTAT = 0.0;

int flag[n];

//Initialize Flag of Every process to Zero

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

flag[i]=0;

int time=AT[0],min,k;

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

{

min=100;

k=0;

//Loop to Find Less Burst Time Process by Checking Flag

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

{

if(flag[i]==0)

{

if(AT[i]<=time)

{

if(BT[i]<min)

{

min=BT[i];

k=i;

}

}

}

}

CT[k]=min+time;

time=time+min;

flag[k]=1;

WT[k]=CT[k]-(BT[k]+AT[k]);

TAT[k]=CT[k]-AT[k];

AvgTAT += TAT[k];

AvgWT += WT[k];

}

AvgWT = AvgWT/n;

AvgTAT = AvgTAT/n;

// STDOUT

printf("\n\tAT\tBT\tWT\tTAT\n");

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

{

printf("P[%d]\t%d\t%d\t%d\t%d\n",i,AT[i],BT[i],WT[i],TAT[i]);

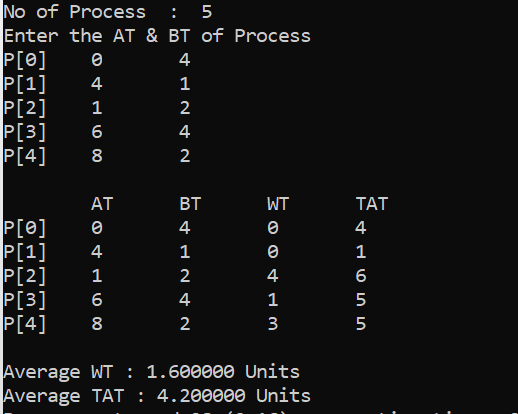
}

printf("\nAverage WT : %f Units\n",AvgWT);

printf("Average TAT : %f Units",AvgTAT);

}

**Input/Output:**



**Result:**

We have Successfully implemented SJF.

**Dated:** 04/05/2021

**Experiment No. – 03 (Priority)**

**Experiment No. 3(a)**

**Objective:**

Calculate avgWT and avgTAT using Priority Scheduling (Preemptive).

**Program:**

#include<stdio.h>

void swap(int \*a, int \*b)

{

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

void main()

{

int n;

float avgWT = 0.0;

float avgTAT = 0.0;

printf("No of Process : ");

scanf("%d",&n);

int AT[n],BT[n],PR[n],CT[n],WT[n],TAT[n],RBT[n];

printf("Enter the AT,BT and Priority of Process\n");

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

{

printf("P[%d]\t",i);

scanf("%d%d %d ",&AT[i],&BT[i],&PR[i]);

}

// Sort the Input

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

{

for(int j=0;j<n-1-i;j++)

{

if(\*(AT+j)>\*(AT+j+1))

{

swap((AT+j),(AT+j+1));

swap((BT+j),(BT+j+1));

swap((PR+j),(PR+j+1));

}

}

}

int sum=0;

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

{

RBT[i]=BT[i];

sum+=BT[i];

}

int time=AT[0],min,k;

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

{

min=100;

k=0;

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

{

if(RBT[j]!=0)

{

if(AT[j]<=time)

{

if(PR[j]<min)

{

min=PR[j];

k=j;

}

}

}

}

RBT[k]=RBT[k]-1;

time=time+1;

CT[k]=time;

}

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

{

WT[i]=CT[i]-AT[i]-BT[i];

TAT[i]=CT[i]-AT[i];

avgTAT += TAT[i];

avgWT += WT[i];

}

avgWT = avgWT/n;

avgTAT = avgTAT/n;

printf("\n \tAT\tBT\tPR\tWT\tTAT\n");

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

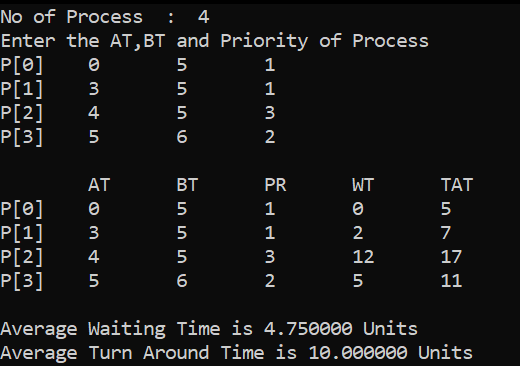
printf("P[%d]\t%d\t%d\t%d\t%d\t%d\n",i,AT[i],BT[i],PR[i],WT[i],TAT[i]);

printf("\nAverage Waiting Time is %f Units\n",avgWT);

printf("Average Turn Around Time is %f Units",avgTAT);

}

**Input/Output:**



**Result:**

We have verified **Priority Scheduling (Preemptive)** Successfully.

**Experiment No. 3(b)**

**Objective:**

Calculate avgWT and avgTAT using Priority Scheduling (Non - Preemptive).

**Program:**

#include<stdio.h>

void swap(int \*a, int \*b)

{

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

void main()

{

int n;

printf("No of Process : ");

scanf("%d",&n);

int AT[n],BT[n],PR[n],CT[n],WT[n],TAT[n],RBT[n];

printf("Enter the AT,BT and Priority of Process\n");

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

{

printf("P[%d]\t",i);

scanf("%d%d%d",&AT[i],&BT[i],&PR[i]);

}

float avgWT = 0.0;

float avgTAT = 0.0;

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

{

for(int j=0;j<n-1-i;j++)

{

if(\*(AT+j)>\*(AT+j+1))

{

swap((AT+j),(AT+j+1));

swap((BT+j),(BT+j+1));

swap((PR+j),(PR+j+1));

}

}

}

int flag[n];

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

{

flag[i]=0;

}

int time=AT[0],min,k;

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

{

min=100;

k=0;

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

{

if(flag[i]==0)

{

if(AT[i]<=time)

{

if(PR[i]<min)

{

min=PR[i];

k=i;

}

}

}

}

CT[k]=BT[k]+time;

time=time+BT[k];

flag[k]=1;

WT[k]=CT[k]-(BT[k]+AT[k]);

TAT[k]=CT[k]-AT[k];

avgTAT += TAT[k];

avgWT += WT[k];

}

avgWT = avgWT/n;

avgTAT = avgTAT/n;

printf("\n \tAT\tBT\tPR\tWT\tTAT\n");

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

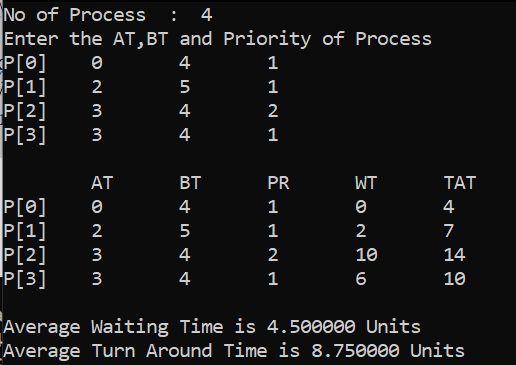
printf("P[%d]\t%d\t%d\t%d\t%d\t%d\n",I,AT[i],BT[i],PR[i],WT[i],TAT[i]);

printf("\nAverage Waiting Time is %f Units\n",avgWT);

printf("Average Turn Around Time is %f Units",avgTAT);

}

**Input/Output:**



**Result:**

We have verified **Priority Scheduling (Non - Preemptive)** Successfully.

**Dated:** 04/05/2021

**Experiment No. – 04 (Round Robin)**

**Objective:**

Calculate avgWT and avgTAT using Round Robin

**Program:**

#include<stdio.h>

void main()

{

int n,q;

printf("No of Process : ");

scanf("%d",&n);

int at[n],bt[n],ct[n],wt[n],tat[n],rbt[n];

printf("Enter the AT & BT of Process\n");

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

{

printf("P[%d]\t",i);

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

}

printf("\nEnter Quantum : ");

scanf("%d",&q);

float AvgWT = 0.0;

float AvgTAT = 0.0;

int sum=0;

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

{

rbt[i]=bt[i];

sum=sum+bt[i];

}

int time=0;

while(time<sum)

{

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

{

if(rbt[i]!=0)

{

if(rbt[i]>=q)

{

rbt[i]=rbt[i]-q;

time=time+q;

if(rbt[i]==0)

ct[i]=time;

}

else

{

time=time+rbt[i];

rbt[i]=0;

ct[i]=time;

}

}

}

}

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

{

wt[i]=ct[i]-bt[i];

tat[i]=ct[i];

AvgTAT += tat[i];

AvgWT += wt[i];

}

AvgWT = AvgWT/n;

AvgTAT = AvgTAT/n;

printf("\n \tAT\tBT\tWT\tTAT\n");

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

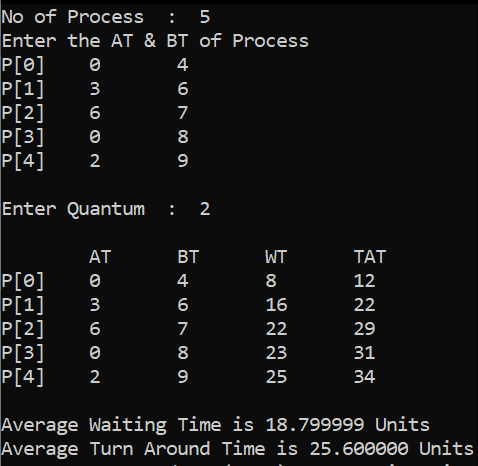
printf("P[%d]\t%d\t%d\t%d\t%d\n",i,at[i],bt[i],wt[i],tat[i]);

printf("\nAverage Waiting Time is %f Units\n",AvgWT);

printf("Average Turn Around Time is %f Units",AvgTAT);

}

**Input/Output :**



**Result:**

We have verified **Round Robin** Successfully.

**Dated:** XX/XX/2021

**Experiment No. – 05 (SRTF)**

**Objective:**

Calculate avgWT and avgTAT using SRTF

**Program:**

#include<stdio.h>

void swap(int \*a, int \*b)

{

int temp;

temp = \*a;

\*a = \*b;

\*b = temp;

}

void main()

{

int n,q;

printf("No of Process : ");

scanf("%d",&n);

int at[n],bt[n],ct[n],wt[n],tat[n],rbt[n];

printf("Enter the AT & BT of Process\n");

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

{

printf("P[%d]\t",i);

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

}

float AvgWT = 0.0;

float AvgTAT = 0.0;

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

{

for(int j=0;j<n-1-i;j++)

{

if(\*(at+j)>\*(at+j+1))

{

swap((at+j),(at+j+1));

swap((bt+j),(bt+j+1));

}

}

}

int sum=0;

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

{

rbt[i]=bt[i];

sum+=bt[i];

}

int time=at[0],min,k;

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

{

min=100;

k=0;

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

{

if(rbt[j]!=0)

{

if(at[j]<=time)

{

if(rbt[j]<min)

{

min=rbt[j];

k=j;

}

}

}

}

rbt[k]=rbt[k]-1;

time=time+1;

ct[k]=time;

}

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

{

wt[i]=ct[i]-at[i]-bt[i];

tat[i]=ct[i]-at[i];

AvgTAT += tat[i];

AvgWT += wt[i];

}

printf("\n \tAT\tBT\tWT\tTAT\n");

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

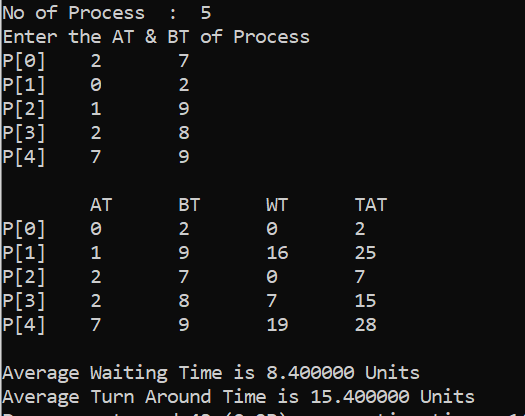
printf("P[%d]\t%d\t%d\t%d\t%d\n",i,at[i],bt[i],wt[i],tat[i]);

printf("\nAverage Waiting Time is %f Units \n",AvgWT/n);

printf("Average Turn Around Time is %f Units",AvgTAT/n);

}

**Input/Output:**



**Result:**

We have verified **SRTF** Successfully.

**Dated:** XX/XX/2021

**Experiment No. 06 (Bankers Algorithms)**

**Objective:** Implementation of Bankers Algorithm

**Program:**

# Program to Implement Bankers Algorithms

# Rohit Kumar | 1900320100131

def calNeed(max, allocation):

    need = []

    for i in range(5):

        l = []

        for j in range(3):

            l.append(max[i][j]-allocation[i][j])

        need.append(l)

    return need

def safe\_seq(availabe, max, allocatin, need):

    n = len(allocatin)

    work = availabe.copy()

    finish = [False]\*n

    ans = []

    k = 0

    while k < len(allocatin):

        for i in range(len(allocatin)):

            if finish[i] == False:

                if need[i][0] <= work[0] and need[i][1] <= work[1] and need[i][2] <= work[2]:

                    ans.append(f"P{i}")

                    for j in range(3):

                        allocatin[i][j] += need[i][j]

                        work[j] -= need[i][j]

                        need[i][j] = max[i][j]-allocatin[i][j]

                    for j in range(3):

                        work[j] += allocatin[i][j]

                    finish[i] = True

        k += 1

    return ans

if \_\_name\_\_ == "\_\_main\_\_":

    availabe = [3, 3, 2]

    max = [[7, 5, 3], [3, 2, 2], [9, 0, 2], [2, 2, 2], [4, 3, 3]]

    allocation = [[0, 1, 0], [2, 0, 0], [3, 0, 2], [2, 1, 1], [0, 0, 2]]

    need = calNeed(max, allocation)

    safeSeq=[]

    print("SAFE SEQUENCE", \*safe\_seq(availabe, max, allocation, need))

    while True:

        p = int(input("ENTER PROCESS ID  :: "))

        req = list(map(int, input("ENTER REQUEST  :: ").split()))

        flag = False

        if req[0] <= need[p][0] and req[1] <= need[p][1] and req[2] <= need[p][2

          if req[0] <= availabe[0] and req[1] <= availabe[1] and req[2] <= availabe[2]]:

                for i in range(len(req)):

                    availabe[i] -= req[i]

                    allocation[p][i] += req[i]

                    need[p][i] -= req[i]

                safeSeq = safe\_seq(availabe, max, allocation, need)

                flag = True

        if len(safeSeq) == 5:

            print("SAFE SEQUENCE ", \*safeSeq)

        else:

            for i in range(len(req)):

                availabe[i] += req[i]

                allocation[p][i] -= req[i]

                need[p][i] += req[i]

        if flag == False:

            print("SAFE SEQUENCE NOT OBTAINED")

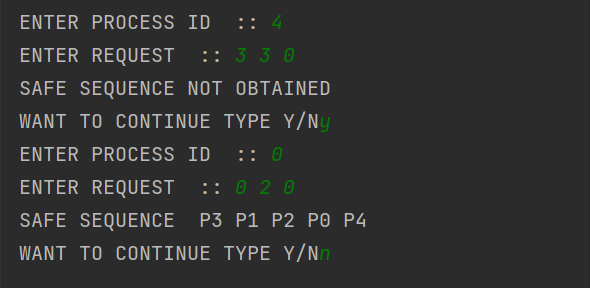
        con = input("WANT TO CONTINUE TYPE Y/N")

        if con == "n" or con == "N":

            break

**Input/Output:**





**Dated:** 15/06/2021

**Experiment No. – 07 (Memory Allocation)**

**Objective:**

Implement memory allocation for six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)?

And print the output in following manner for each process under every scheme:

**Program:**

#include <stdio.h>

#include <stdlib.h>

main()

{

int i, j, flag, x, y, min, alloc[100], k = 0, index, bestIdx, size1[100], size2[100];

int size[6] = {300, 600, 350, 200, 750, 125};

int process[5] = {115, 500, 358, 200, 375};

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

{

size1[i] = size[i];

size2[i] = size[i];

}

printf("-------------------------------------First Fit Allocation-------------------------------------\n\n");

for (j = 0; j < 5; j++)

{

flag = 0;

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

{

if (size[i] >= process[j])

{

flag = 1;

x = size[i];

size[i] = size[i] - process[j];

y = size[i];

break;

}

}

if (flag == 1)

{

printf("Process of size %d is allocated in the memory block of %d and new hole is created of size %d\n", process[j], x, y);

}

else

printf("Process of size %d is not allocated because sufficient contiguous memory hole is not available to load the process.\n", process[j]);

}

printf("\n-------------------------------------Best Fit Allocation-------------------------------------\n\n");

for (j = 0; j < 5; j++)

{

index = -1;

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

{

if (size1[i] >= process[j])

{

if (index == -1)

index = i;

else if (size1[index] > size1[i])

index = i;

}

}

if (index != -1)

{

x = size1[index];

size1[index] -= process[j];

y = size1[index];

printf("Process of size %d is allocated in the memory block of %d and new hole is created of size %d\n", process[j], x, y);

}

else

printf("Process of size %d is not allocated because sufficient contiguous memory hole is not available to load the process.\n", process[j]);

}

printf("\n-------------------------------------For worst fit allocation-------------------------------------\n\n");

for (j = 0; j < 5; j++)

{

index = -1;

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

{

if (size2[i] >= process[j])

{

if (index == -1)

index = i;

else if (size2[index] < size2[i])

index = i;

}

}

if (index != -1)

{

x = size2[index];

size2[index] -= process[j];

y = size2[index];

printf("Process of size %d is allocated in the memory block of %d and new hole is created of size %d\n", process[j], x, y);

}

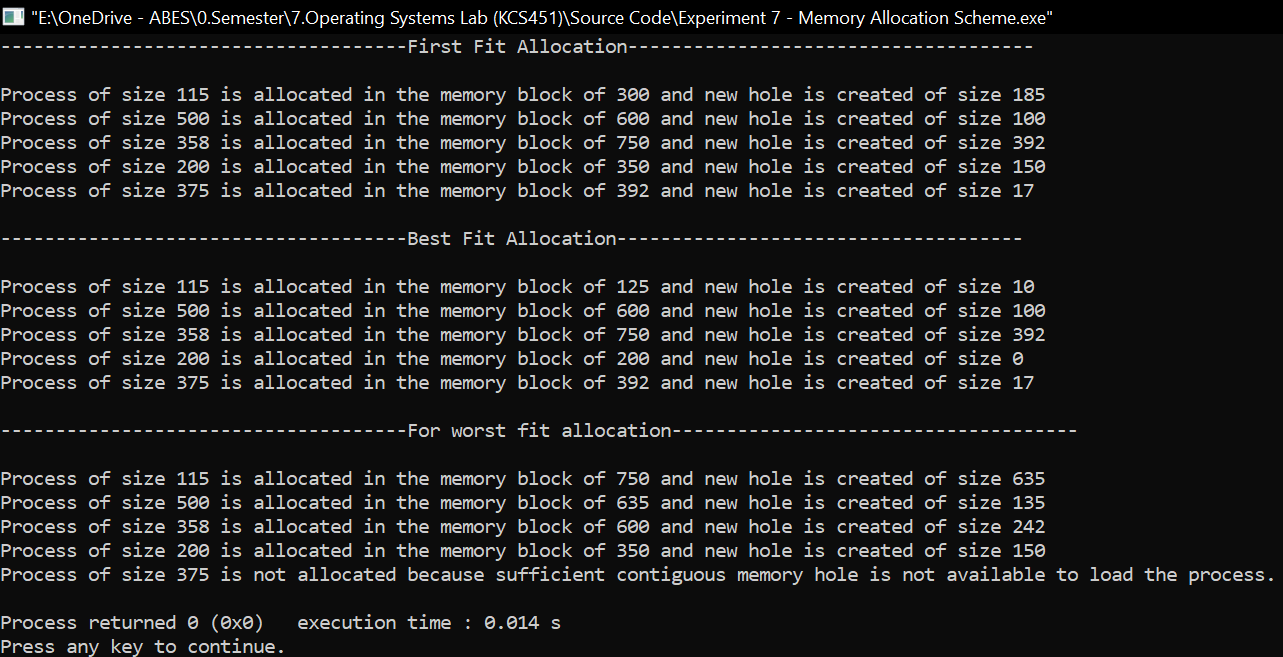
else

printf("Process of size %d is not allocated because sufficient contiguous memory hole is not available to load the process.\n", process[j]);

}

}

**Input/Output:**



**Dated:** 06/07/2021

**Experiment No. – 08 (Page Replacement Algorithms)**

**Objective:**

Consider the following page reference string:

7, 2, 3, 1, 2, 5, 3, 4, 6, 7, 7, 1, 0, 5, 4, 6, 2, 3, 0, 1.

Assuming demand paging with three frames, how many page faults would occur for the following replacement algorithms?

• LRU replacement

• FIFO replacement

• Optimal replacement

**Programs:**

#include<stdio.h>

int minimum(int arr[], int n)

{

int i, m = 0;

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

{

if(arr[i] < arr[m])

m = i;

}

return m;

}

int maximum(int arr[], int n)

{

int i, m = 0;

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

{

if(arr[i] > arr[m])

m = i;

}

return m;

}

void FIFO(int str[], int frames\_no, int total\_pages)

{

int i, j, flag, last\_in = -1, page\_fault = 0, frames[frames\_no];

for(i=0; i<total\_pages; i++)

{

flag = 0;

for(j=0; j<frames\_no; j++)

{

if(str[i]==frames[j])

flag = 1;

}

if(flag == 0)

{

last\_in = (last\_in+1)%frames\_no;

frames[last\_in] = str[i];

page\_fault++;

}

}

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(FIFO)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Final Frames :\n");

for(i=0; i<frames\_no; i++)

printf("%d\t",frames[i]);

printf("No of Page faults : %d\n",page\_fault);

}

void LRU(int str[], int frames\_no, int total\_pages)

{

int i, frames[frames\_no], page\_faults = 0, full = 0, age[frames\_no], j, flag, leastRecent;

for(i=0; i<total\_pages; i++)

{

if(full < frames\_no)

{

frames[full] = str[i];

age[full] = i;

full++;

page\_faults++;

}

else

{

flag = 0;

for(j=0; j<frames\_no; j++)

{

if(str[i]==frames[j])

{

flag = 1;

age[j] = i;

}

}

if(flag == 0)

{

leastRecent = minimum(age, frames\_no);

frames[leastRecent] = str[i];

age[leastRecent] = i;

page\_faults++;

}

}

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(LRU)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Final Frames :\n");

for(i=0; i<frames\_no; i++)

printf("%d\t",frames[i]);

printf("No of Page faults : %d\n",page\_faults);

}

void Optimal(int str[], int frames\_no, int total\_pages)

{

int i, frames[frames\_no], page\_faults = 0, full = 0, j, flag, freq[frames\_no], optimal, k;

for(i=0; i<total\_pages; i++)

{

if(full < frames\_no)

{

frames[full] = str[i];

full++;

page\_faults++;

}

else

{

flag = 0;

for(j=0; j<frames\_no; j++)

{

if(str[i]==frames[j])

{

flag = 1;

}

}

if(flag == 0)

{

for(j=0; j<frames\_no; j++)

{

int f = 0;

for(k=i; k<total\_pages; k++)

{

if(str[k] == frames[j])

{

freq[j] = k;

f = 1;

break;

}

}

if(f==0)

freq[j] = 100;

}

optimal = maximum(freq, frames\_no);

frames[optimal] = str[i];

page\_faults++;

}

}

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*(Optimal)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("Final Frames :\n");

for(i=0; i<frames\_no; i++)

printf("%d\t",frames[i]);

printf("No of Page faults : %d\n",page\_faults);

}

int main(void)

{

int page[] = {7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1};

int total\_pages = 20;

int frames\_no = 3;

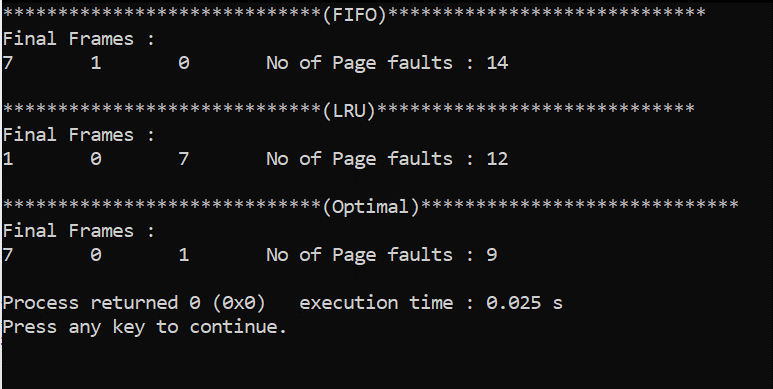
FIFO(page, frames\_no, total\_pages);

LRU(page, frames\_no, total\_pages);

Optimal(page, frames\_no, total\_pages);

}

**Output:**



**Dated:** 06/07/2021

**Experiment No. – 09 (Disk Scheduling Algorithms)**

**Objective:**

Consider total number of cylinders as 200. Head pointers processing the cylinder 53 currently after processing the request of cylinder no 5.

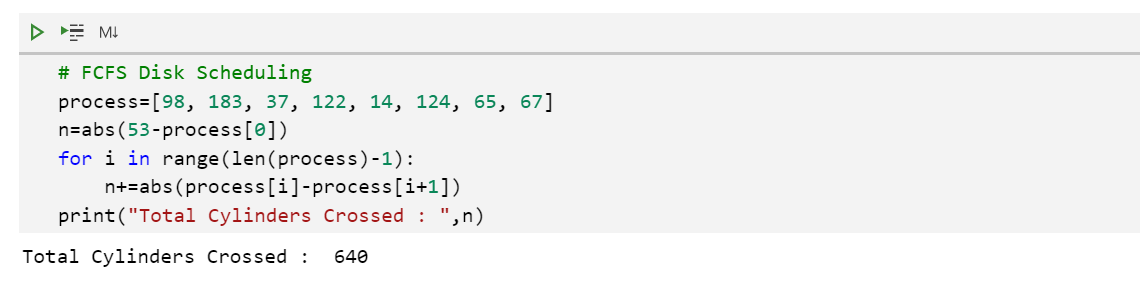
New request for processing are : 98, 183, 37, 122, 14, 124, 65, 67. Find the number of cylinder crossed while processing this request using following scheduling

* **FIFO**
* **SSTF**
* **SCAN**
* **C-SCAN**
* **LOOK**
* **C-LOOK**

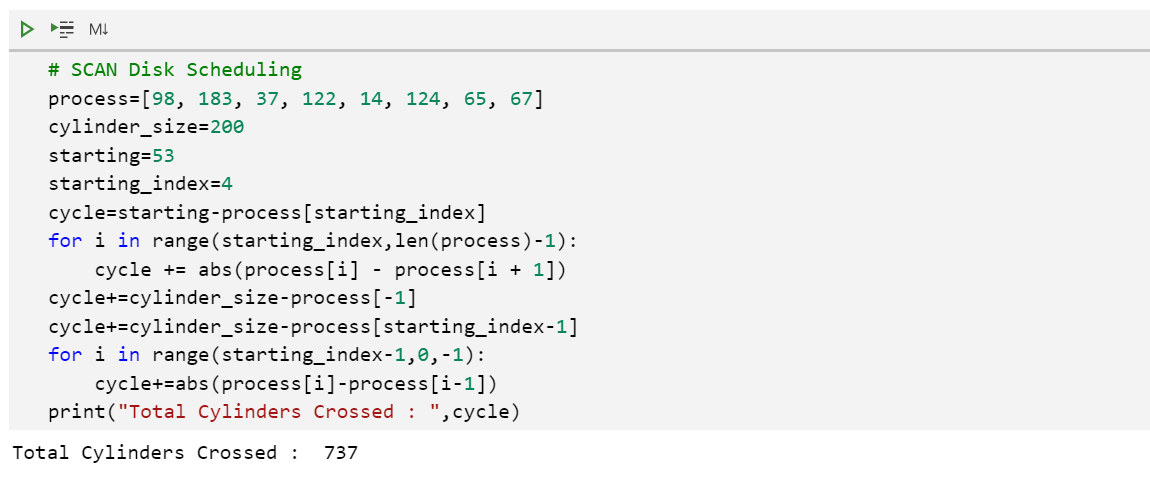
Note: Implement any three of them and upload the code along with output snapshot.

**Program & Output:**

**1.FCFS**



**2.SSTF**



**3.SCAN**

