

FCFS

```
#include <stdio.h>
struct Process{
int id,at,bt,ct,tat,wt;
};
void main()
{
int n;
float tot_tat=0,tot_wt=0;
float avg_tat,avg_wt;
printf("Enter the no.of process:");
scanf("%d",&n);
struct Process p[n],temp;
printf("Enter the Arrival time and Burst time of the process:");
for (int i=0;i<n;i++)
{
p[i].id=i+1;
printf("P%d(AT,BT):",i+1);
scanf("%d%d",&p[i].at,&p[i].bt);
}
for(int i=0;i<n-1;i++)
{
for(int j=0;j<n-i-1;j++)
{
if(p[j].at>p[j+1].at)
{
temp=p[j];
p[j]=p[j+1];
p[j+1]=temp;
}
}
}
p[0].ct=p[0].at+p[0].bt;
p[0].tat=p[0].ct-p[0].at;
p[0].wt=p[0].tat-p[0].bt;
tot_tat=p[0].tat;
tot_wt=p[0].wt;
for(int i=1;i<n;i++)
{
if(p[i].at>p[i-1].ct)
{
p[i].ct=p[i].at+p[i].bt;
}
else{
p[i].ct=p[i-1].ct+p[i].bt;
}
p[i].tat=p[i].ct-p[i].at;
p[i].wt=p[i].tat-p[i].bt;
tot_tat=tot_tat+p[i].tat;
tot_wt=tot_wt+p[i].wt;
}
avg_tat=tot_tat/n;
avg_wt=tot_wt/n;
printf("PID\tAT\tBT\tCT\tTAT\tWT\n");
for(int i=0;i<n;i++)
{
printf("P%d\t%d\t%d\t%d\t%d\t%d\n",p[i].id,p[i].at,p[i].bt,p[i].ct,p[i].tat,p[i].wt);
```

```
}  
printf("Average TAT=%f\nAverage WT=%f",avg_tat,avg_wt);  
}
```

Output

Enter the no.of process: 4

Enter the Arrival time and Burst time of the process:

P1(AT,BT): 0 5

P2(AT,BT): 1 3

P3(AT,BT): 2 8

P4(AT,BT): 3 6

PID	AT	BT	CT	TAT	WT
-----	----	----	----	-----	----

P1	0	5	5	5	0
----	---	---	---	---	---

P2	1	3	8	7	4
----	---	---	---	---	---

P3	2	8	16	14	6
----	---	---	----	----	---

P4	3	6	22	19	13
----	---	---	----	----	----

Average TAT = 11.250000

Average WT = 5.750000

LRU

```
#include <stdio.h>
int main()
{
int reference [50],frame[50],fsize,i,j,flag=0,c=0,n,fault=0,k,recent[50],temp,flag1=0;
float miss,hit;
printf("Enter the number of references:");
scanf("%d",&n);
printf("Enter the references:");
for(i=0;i<n;i++)
scanf("%d",&reference[i]);
printf("Enter the frame size:");
scanf("%d",&fsize);
for(i=0;i<fsize;i++)
{
fault++;
frame[i]=reference[i];
recent[i]=reference[i];
c++;
}
recent[c]=0;
for(i=fsize;i<n;i++)
{
for(k=0;k<c;k++)
{
if(reference[i]==recent[k])
{
flag1=1;
break;
}
}
if(flag1==1)
{
temp=recent[k];
for(j=k;j<c;j++)
recent[j]=recent[j+1];
recent[c-1]=temp;
}
else
{
recent[c]=reference[i];
c++;
recent[c]=0;
}
flag1=0;
for(j=0;j<fsize;j++)
{
if(frame[j]==reference[i])
flag=1;
}
if(flag!=1)
{
fault++;
for(k=0;k<c;k++)
{
for(j=0;j<fsize;j++)
{
```

```

if(recent[k]==frame[j])
{
frame[j]=reference[i];
goto end;
}
}
}
}
end:
flag=0;
}
printf("Total number of fault=%d\n",fault);
miss=((float)fault/n)*100;
hit=((float)(n-fault)/n)*100;
printf("Total number of references = %d\n",n);
printf("Miss ratio=%.2f\n",miss);
printf("No.of Hits=%d\n",(n-fault));
printf("Hit ratio=%.2f\n",hit);
return 0;
}

```

Output

Enter the number of references: 12
Enter the references: 1 3 0 3 5 6 3 0 3 1 2 1
Enter the frame size: 3

Total number of fault = 7
Total number of references = 12
Miss ratio = 58.33
No. of Hits = 5
Hit ratio = 41.67

SJF

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
struct Process{
    int id,at,bt,ct,tat,wt,completed;
};
int findshortest(struct Process p[],int n,int currentTime){
    int index=-1,bt=INT_MAX;
    for(int i=0;i<n;i++){
        {
            if(p[i].at<=currentTime&& p[i].completed!=0){
                if(p[i].bt<bt) {
                    bt=p[i].bt;
                    index=i;
                }
            }
        }
    }
    return index;
}
void sjf(struct Process p[],int n)
{
    int completed=0,currentTime=0;
    float tot_tat=0,tot_wt=0;
    while(completed<n)
    {
        int ind=findshortest(p,n,currentTime);
        if(ind!=-1)
        {
            currentTime++;
        }
        else{
            currentTime=p[ind].bt+currentTime;
            completed++;
            p[ind].completed=0;
            p[ind].ct=currentTime;
            p[ind].tat=p[ind].ct-p[ind].at;
            p[ind].wt=p[ind].tat-p[ind].bt;
            tot_tat+=p[ind].tat;
            tot_wt+=p[ind].wt;
        }
    }
    printf("PID\tAT\tBT\tCT\tTAT\tWT\n");
    for(int i=0;i<n;i++){
        {
            printf("P%d\t%d\t%d\t%d\t%d\t%d\n",p[i].id,p[i].at,p[i].bt,p[i].ct,p[i].tat,p[i].wt);
        }
    }
    printf("Average TAT=%f\n",tot_tat/n);
    printf("Average WT=%f\n",tot_wt/n);
}
void main()
{
    int n;
    printf("Enter the number of process:");
    scanf("%d",&n);
    struct Process p[n];
    printf("Enter the Arrival time(AT) and Burst Time(BT) \n");
```

```

for(int i=0;i<n;i++)
{
p[i].id=i+1;
printf("P%d(AT,BT):",p[i].id);
scanf("%d%d",&p[i].at,&p[i].bt);
p[i].completed=-1;
}
sjf(p,n);
}

```

Output

Enter the number of processes: 4

Enter the Arrival Time (AT) and Burst Time (BT)

P1(AT, BT): 0 6

P2(AT, BT): 2 8

P3(AT, BT): 4 7

P4(AT, BT): 5 3

PID	AT	BT	CT	TAT	WT
P1	0	6	6	6	0
P4	5	3	9	4	1
P3	4	7	16	12	5
P2	2	8	24	22	14

Average TAT = 11.00

Average WT = 5.00

BANKERS

```
#include<stdio.h>
struct pro
{
int all[10],max[10],need[10];
int flag;
} p[10];
int i,j,pno,r,id,k=0,safe=0,exec,count=0;
int aval[10],seq[10];
void safeState()
{
while(count!=pno)
{
for(i=0;i<pno;i++)
if(p[i].flag)
{
exec=r;
for(j=0;j<r;j++)
if(p[i].need[j]>aval[j])
exec=0;
if(exec==r)
{
for(j=0;j<r;j++)
aval[j]+=p[i].all[j];
p[i].flag=0;
seq[k++]=i;
safe=1;
count++;
}}
if(!safe)
{
printf("System is in Unsafe State\n");
break;
}
}
if(safe)
{
printf("System is in safe State.The Safe sequence: ");
for(i=0;i<pno;i++)
printf("P[%d] ",seq[i]);
printf("\n");
}
}
int main()
{
printf("Enter no of process: ");
scanf("%d",&pno);
printf("Enter no of resources: ");
scanf("%d",&r);
printf("Enter available resources of each type:");
for(j=0;j<r;j++)
scanf("%d",&aval[j]);
printf("Enter process details: ");
for(i=0;i<pno;i++)
{
printf("\n Process %d\n",i);
```

```

printf("Allocation Matrix:\t");
for(j=0;j<r;j++)
scanf("%d",&p[i].all[j]);
printf("Maximum Matrix:\t\t");
for(j=0;j<r;j++)
scanf("%d",&p[i].max[j]);
p[i].flag=1;
for(j=0;j<r;j++)
p[i].need[j]=p[i].max[j]-p[i].all[j];
}

printf("\nProcess details\n");
printf("PID\t\tALL\t\tMax\t\tNeed\n");
for(i=0;i<pno;i++)
{
printf("%d\t\t",i);
for(j=0;j<r;j++)
printf("%d ",p[i].all[j]);
printf("\t\t");
for(j=0;j<r;j++)
printf("%d ",p[i].max[j]);
printf("\t\t");
for(j=0;j<r;j++)
printf("%d ",p[i].need[j]);
printf("\n");
}
safeState();
return 0;
}

```

Output

Enter no of process: 5
Enter no of resources: 3
Enter available resources of each type: 3 3 2
Enter process details:

Process 0
Allocation Matrix: 0 1 0
Maximum Matrix: 7 5 3

Process 1
Allocation Matrix: 2 0 0
Maximum Matrix: 3 2 2

Process 2
Allocation Matrix: 3 0 2
Maximum Matrix: 9 0 2

Process 3
Allocation Matrix: 2 1 1
Maximum Matrix: 4 2 2

Process 4
Allocation Matrix: 0 0 2
Maximum Matrix: 5 3 3

Process details

PID	ALL	Max	Need
0	0 1 0	7 5 3	7 4 3
1	2 0 0	3 2 2	1 2 2
2	3 0 2	9 0 2	6 0 0
3	2 1 1	4 2 2	2 1 1
4	0 0 2	5 3 3	5 3 1

System is in safe state. The Safe sequence: P[1] P[3] P[4] P[2] P[0]

BESTFIT

```
#include <stdio.h>
#define max 25
int i,j,k=0,nb,nf,temp=0,lowest=999,flag=0;
void bestfit(int b[],int f[])
{
    for(i=1;i<=nf;i++)
    {
        for(j=1;j<=nb;j++)
        {
            temp=b[j]-f[i];
            if(temp>=0)
            {
                if(lowest>temp)
                {
                    k=j;
                    lowest=temp;
                }
            }
        }
        if(lowest!=999)
            printf("\nFile Size %d is put in %d partition\n",f[i],b[k]);
        else
            printf("\nFile Size %d must wait",f[i]);
        b[k]=lowest;
        lowest=999;
    }
}

int main()
{
    int b[max],f[max];
    printf("\nMemory Management Scheme-Best Fit");
    printf("\nEnter the number of blocks:");
    scanf("%d",&nb);
    printf("\nEnter the number of files:");
    scanf("%d",&nf);
    printf("\nEnter the size of the blocks:\n");
    for(i=1;i<=nb;i++)
    {
        printf("Block %d:",i);
        scanf("%d",&b[i]);
    }
    printf("Enter the size of the files:\n");
    for(i=1;i<=nf;i++)
    {
        printf("File %d:",i);
        scanf("%d",&f[i]);
    }
    bestfit(b,f);
    return 0;
}
```

Output

Memory Management Scheme-Best Fit

Enter the number of blocks: 5

Enter the number of files: 4

Enter the size of the blocks:

Block 1: 100

Block 2: 500

Block 3: 200

Block 4: 300

Block 5: 600

Enter the size of the files:

File 1: 212

File 2: 417

File 3: 112

File 4: 426

File Size 212 is put in 3 partition

File Size 417 is put in 4 partition

File Size 112 is put in 1 partition

File Size 426 is put in 5 partition

NON PREEMPTIVE PRIORITY

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>

struct Process
{
    int id, at, bt, ct, tat, wt, pt;
    int completed; // To track completion
};

int findHighestPriority(struct Process p[], int n, int currentTime)
{
    int index = -1, highestPriority = INT_MAX;

    for (int i = 0; i < n; i++)
    {
        if (p[i].at <= currentTime && !p[i].completed) // Process must have arrived and not be completed
        {
            if (p[i].pt < highestPriority || (p[i].pt == highestPriority && p[i].at < p[index].at))
            {
                highestPriority = p[i].pt;
                index = i;
            }
        }
    }
    return index;
}

void NonPreemptivePriority(struct Process p[], int n)
{
    int completedCount = 0, currentTime = 0;
    float totalTAT = 0, totalWT = 0;

    while (completedCount < n)
    {
        int ind = findHighestPriority(p, n, currentTime);

        if (ind == -1) // If no process is available, move time forward
        {
            currentTime++;
        }
        else
        {
            // Process execution
            p[ind].completed = 1;
            completedCount++;
            currentTime += p[ind].bt;
            p[ind].ct = currentTime;
            p[ind].tat = p[ind].ct - p[ind].at;
            p[ind].wt = p[ind].tat - p[ind].bt;
            totalTAT += p[ind].tat;
            totalWT += p[ind].wt;
        }
    }
}
```

```

printf("PID\tAT\tBT\tP\tCT\tTAT\tWT\n");
for (int i = 0; i < n; i++)
{
    printf("P%d\t%d\t%d\t%d\t%d\t%d\t%d\n", p[i].id, p[i].at, p[i].bt, p[i].pt, p[i].ct, p[i].tat, p[i].wt);
}

printf("Average TAT = %.2f\n", totalTAT / n);
printf("Average WT = %.2f\n", totalWT / n);
}

int main()
{
    int n;
    printf("Enter the number of processes: ");
    scanf("%d", &n);

    struct Process p[n];

    printf("Enter Arrival Time (AT), Burst Time (BT), and Priority (P) for each process:\n");
    for (int i = 0; i < n; i++)
    {
        p[i].id = i + 1;
        p[i].completed = 0; // Initially, no process is completed
        printf("P%d (AT, BT, P): ", p[i].id);
        scanf("%d %d %d", &p[i].at, &p[i].bt, &p[i].pt);
    }

    NonPreemptivePriority(p, n);
    return 0;
}

```

Output

Enter the number of processes: 4

Enter Arrival Time (AT), Burst Time (BT), and Priority (P) for each process:

P1 (AT, BT, P): 0 5 2

P2 (AT, BT, P): 1 3 1

P3 (AT, BT, P): 2 8 4

P4 (AT, BT, P): 3 6 3

PID	AT	BT	P	CT	TAT	WT
P1	0	5	2	5	5	0
P2	1	3	1	8	7	4
P4	3	6	3	14	11	5
P3	2	8	4	22	20	12

Average TAT = 10.75

Average WT = 5.25

IPC

WRITER

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    key_t key=ftok("shmfile",65);
    int shmid=shmget(key,1024,0666 | IPC_CREAT);
    char *data=(char *)shmat(shmid,NULL,0);
    printf("Enter a string:");
    fgets(data,1024,stdin);
    return 0;
}
```

READER

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    key_t key=ftok("shmfile",65);
    int shmid=shmget(key,1024,0666);
    char *data=(char *)shmat(shmid,NULL,0);
    printf("Data from writer: %s",data);
    return 0;
}
```

Output

Enter a string: Hello from Shared Memory!

Data from writer: Hello from Shared Memory!

SEMAPHORE

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int mutex=1,full=0,empty=3,x=0;
int signal(int s)
{
    return (++s);
}
int wait(int s)
{
    return (--s);
}
void producer()
{
    empty=wait(empty);
    mutex=wait(mutex);
    x++;
    printf("\n Producer Produces the item %d",x);
    mutex=signal(mutex);
    full=signal(full);
}
void consume()
{
    full=wait(full);
    mutex=wait(mutex);
    printf("\n Consumer consumes the item %d",x);
    x--;
    mutex=signal(mutex);
    empty=signal(empty);
}
void main()
{
    int n;
    while(1)
    {
        printf("\n1.Producer\n2.Consumer\n3.Exit\n");
        printf("\nEnter your choice: \n");
        scanf("%d",&n);
        switch(n)
        {
            case 1:
                if((mutex==1)&&(empty!=0))
                    producer();
                else
                    printf("\nBuffer is Full\n");
                break;
            case 2:
                if((mutex==1)&&(full!=0))
                    consume();
                else
                    printf("\nBuffer is Empty\n");
                break;
            case 3:
```

```
exit(0);  
break;  
}  
}  
}
```

Output

1. Producer
2. Consumer
3. Exit

Enter your choice:

1

Producer Produces the item 1

Enter your choice:

1

Producer Produces the item 2

Enter your choice:

1

Producer Produces the item 3

Enter your choice:

1

Buffer is Full

Enter your choice:

2

Consumer consumes the item 3

Enter your choice:

2

Consumer consumes the item 2

Enter your choice:

2

Consumer consumes the item 1

Enter your choice:

2

Buffer is Empty

Enter your choice:

3

Srtf

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
struct Process{
    int id,at,bt,ct,tat,wt,rt;
};
int findshortest(struct Process p[],int n,int currentTime){
    int index=-1,rt=INT_MAX;
    for(int i=0;i<n;i++) {
        if(p[i].at<=currentTime&& p[i].rt!=0) {
            if(p[i].rt<rt) {
                rt=p[i].rt;
                index=i;
            }
        }
    }
    return index;
}
void Srtf(struct Process p[],int n){
    int completed=0,currentTime=0;
    float tot_tat=0,tot_wt=0;
    while(completed<n){
        int ind=findshortest(p,n,currentTime);
        if(ind== -1){
            currentTime++;
        }
        else{
            p[ind].rt--;
            currentTime++;
            if(p[ind].rt==0){
                completed++;
                p[ind].ct=currentTime;
                p[ind].tat=p[ind].ct-p[ind].at;
                p[ind].wt=p[ind].tat-p[ind].bt;
                tot_tat+=p[ind].tat;
                tot_wt+=p[ind].wt;
            }
        }
    }
    printf("PID\tAT\tBT\tCT\tTAT\tWT\n");
    for(int i=0;i<n;i++){
        printf("P%d\t%d\t%d\t%d\t%d\t%d\n",p[i].id,p[i].at,p[i].bt,p[i].ct,p[i].tat,p[i].wt);
    }
    printf("Average TAT=%f\n",tot_tat/n);
    printf("Average WT=%f\n",tot_wt/n);
}
void main(){
    int n;
    printf("Enter the number of process:");
```

```

scanf("%d",&n);
struct Process p[n];
printf("Enter the Arrival time(AT) and Burst Time(BT) \n");
for(int i=0;i<n;i++){
p[i].id=i+1;
printf("P%d(AT,BT):",p[i].id);
scanf("%d%d",&p[i].at,&p[i].bt);
p[i].rt=p[i].bt;
}

```

Output

Enter the number of processes: 4
Enter the Arrival time (AT) and Burst Time (BT) for each process:

P1 (AT, BT): 0 4

P2 (AT, BT): 1 3

P3 (AT, BT): 2 5

P4 (AT, BT): 3 2

PID	AT	BT	CT	TAT	WT
P1	0	4	7	7	3
P2	1	3	4	3	0
P3	2	5	12	10	5
P4	3	2	9	6	4

Average TAT = 6.50

Average WT = 3.00