**1. FCFS**

[Write a simulation program to implement FCFS CPU-scheduling algorithm. Accept the number of Processes as input. Also accept arrival time and CPU burst time for each process as input. The output should give the Gantt chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-15-2-iprogramx.html)

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct process\_info

{

char pname[20]; int at,bt,ct,bt1;

struct process\_info \*next;

}NODE;

int n;

NODE \*first,\*last;

void accept\_info()

{

NODE \*p;

int i;

printf("Enter no.of process:"); scanf("%d",&n);

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

{

p = (NODE\*)malloc(sizeof(NODE));

printf("Enter process name:");

scanf("%s",p->pname);

printf("Enter arrival time:"); scanf("%d",&p->at);

printf("Enter first CPU burst time:");

scanf("%d",&p->bt);

p->bt1 = p->bt;

p->next = NULL;

if(first==NULL) first=p; else last->next=p;

last = p;

}

}

void print\_output()

{

NODE \*p; float avg\_tat=0,avg\_wt=0;

printf("pname\tat\tbt\tct\ttat\twt\n");

p = first; while(p!=NULL)

{

int tat = p->ct-p->at;

int wt = tat-p->bt;

avg\_tat+=tat;

avg\_wt+=wt;

printf("%s\t%d\t%d\t%d\t%d\t%d\n",

p->pname,p->at,p->bt,p->ct,tat,wt);

p=p->next;

}

printf("Avg TAT=%f\tAvg WT=%f\n",

avg\_tat/n,avg\_wt/n);

}

void print\_input()

{

NODE \*p;

p = first;

printf("pname\tat\tbt\n");

while(p!=NULL)

{

printf("%s\t%d\t%d\n", p->pname,p->at,p->bt1);

p = p->next;

}

}

void sort()

{

NODE \*p,\*q;

int t;

char name[20];

p = first; while(p->next!=NULL)

{

q=p->next; while(q!=NULL)

{

if(p->at > q->at)

{

strcpy(name,p->pname); strcpy(p->pname,q->pname);

strcpy(q->pname,name);

t = p->at; p->at = q->at;

q->at = t;

t = p->bt; p->bt = q->bt;

q->bt = t;

t = p->ct; p->ct = q->ct;

q->ct = t;

t = p->bt1; p->bt1 = q->bt1;

q->bt1 = t;

}

q=q->next;

}

p=p->next;

}

}

int time;

NODE \* get\_fcfs()

{

NODE \*p;

p = first; while(p!=NULL)

{

if(p->at<=time && p->bt1!=0) return p;

p=p->next;

}

return NULL;

}

struct gantt\_chart

{

int start; char pname[30]; int end;

}s[100],s1[100];

int k;

void fcfs()

{

int prev=0,n1=0;

NODE \*p;

while(n1!=n)

{

p = get\_fcfs();

if(p==NULL)

{

time++; s[k].start = prev; strcpy(s[k].pname,"\*");

s[k].end = time;

prev = time;

k++;

}

else

{

time+=p->bt1; s[k].start = prev; strcpy(s[k].pname, p->pname); s[k].end = time;

prev = time;

k++;

p->ct = time;

p->bt1 = 0;

n1++;

}

print\_input();

sort();

}

}

void print\_gantt\_chart()

{ int i,j,m;

s1[0] = s[0];

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

{

if(strcmp(s[i].pname,s1[j].pname)==0)

s1[j].end = s[i].end; else s1[++j] = s[i];

}

printf("%d",s1[0].start); for(i=0;i<=j;i++)

{

m = (s1[i].end - s1[i].start);

for(k=0;k<m/2;k++)

printf("-");

printf("%s",s1[i].pname);

for(k=0;k<(m+1)/2;k++)

printf("-");

printf("%d",s1[i].end);

}

}

int main()

{

accept\_info(); sort();

fcfs(); print\_output();

print\_gantt\_chart();

return 0;

}

# 2. Round Robin

[Write a simulation program to implement Round Robin CPU scheduling algorithm for the given time quantum as input. Also accept the number of processes and arrival time and CPU burst time for each process as input. The output should give the Gant Chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-2-2-iprogramx.html)

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct process\_info

{

char pname[20]; int at,bt,ct,bt1;

struct process\_info \*next;

}NODE;

int n,ts;

NODE \*first,\*last;

void accept\_info()

{

NODE \*p;

int i;

printf("Enter no.of process:");

scanf("%d",&n);

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

{

p = (NODE\*)malloc(sizeof(NODE));

printf("Enter process name:");

scanf("%s",p->pname);

printf("Enter arrival time:");

scanf("%d",&p->at);

printf("Enter first CPU burst time:"); scanf("%d",&p->bt);

p->bt1 = p->bt;

p->next = NULL;

if(first==NULL) first=p; else last->next=p;

last = p;

}

printf("Enter time slice:");

scanf("%d",&ts);

}

void print\_output()

{

NODE \*p; float avg\_tat=0,avg\_wt=0; printf("pname\tat\tbt\tct\ttat\twt\n");

p = first; while(p!=NULL)

{

int tat = p->ct-p->at;

int wt = tat-p->bt;

avg\_tat+=tat;

avg\_wt+=wt;

printf("%s\t%d\t%d\t%d\t%d\t%d\n",

p->pname,p->at,p->bt,p->ct,tat,wt);

p=p->next;

}

printf("Avg TAT=%f\tAvg WT=%f\n",

avg\_tat/n,avg\_wt/n);

}

void print\_input()

{

NODE \*p;

p = first;

printf("pname\tat\tbt\n");

while(p!=NULL)

{

printf("%s\t%d\t%d\n", p->pname,p->at,p->bt1);

p = p->next;

}

}

void sort()

{

NODE \*p,\*q;

int t;

char name[20];

p = first; while(p->next!=NULL) {

q=p->next; while(q!=NULL)

{

if(p->at > q->at)

{

strcpy(name,p->pname); strcpy(p->pname,q->pname);

strcpy(q->pname,name);

t = p->at; p->at = q->at;

q->at = t;

t = p->bt; p->bt = q->bt;

q->bt = t;

t = p->ct; p->ct = q->ct;

q->ct = t;

t = p->bt1; p->bt1 = q->bt1;

q->bt1 = t;

}

q=q->next;

}

p=p->next;

}

}

int time;

int is\_arrived()

{

NODE \*p;

p = first; while(p!=NULL)

{

if(p->at<=time && p->bt1!=0) return 1;

p=p->next;

}

return 0;

}

NODE \* delq()

{

NODE \*t;

t = first; first = first->next; t->next=NULL;

return t;

}

void addq(NODE \*t)

{

last->next = t; last = t;

}

struct gantt\_chart

{

int start; char pname[30]; int end;

}s[100],s1[100];

int k;

void rr()

{

int prev=0,n1=0;

NODE \*p;

while(n1!=n)

{

if(!is\_arrived())

{

time++; s[k].start = prev; strcpy(s[k].pname,"\*"); s[k].end = time; k++; prev=time;

}

else

{ p = first; while(1)

{

if(p->at<=time && p->bt1!=0)

break;

p = delq(); addq(p); p = first;

}

if(p->bt1<=ts)

{

time+=p->bt1;

p->bt1=0;

}

else

{

time+=ts; p->bt1-=ts;

}

p->ct = time;

s[k].start = prev; strcpy(s[k].pname,p->pname);

s[k].end = time;

k++; prev = time;

if(p->bt1==0) n1++;

p = delq(); addq(p);

}

print\_input();

}

}

void print\_gantt\_chart()

{ int i,j,m;

s1[0] = s[0];

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

{

if(strcmp(s[i].pname,s1[j].pname)==0)

s1[j].end = s[i].end; else s1[++j] = s[i];

}

printf("%d",s1[0].start);

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

{

m = (s1[i].end - s1[i].start);

for(k=0;k<m/2;k++)

printf("-");

printf("%s",s1[i].pname);

for(k=0;k<(m+1)/2;k++)

printf("-");

printf("%d",s1[i].end);

}

}

int main()

{

accept\_info(); sort();

rr();

print\_output();

print\_gantt\_chart();

return 0;

}

# 3. Pre-emptive Shortest Job First (SJF)

[Write a simulation program to implement a Pre-emptive Shortest Job First (SJF) – CPU scheduling algorithm. Accept the number of Processes as input. Also accept arrival time and CPU burst time for each process as input. The output should give the Gantt chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-10-2-iprogramx.html)

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct process\_info

{

char pname[20]; int at,bt,ct,bt1;

struct process\_info \*next;

}NODE;

int n;

NODE \*first,\*last;

void accept\_info()

{

NODE \*p;

int i;

printf("Enter no.of process:"); scanf("%d",&n);

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

{

p = (NODE\*)malloc(sizeof(NODE));

printf("Enter process name:");

scanf("%s",p->pname);

printf("Enter arrival time:"); scanf("%d",&p->at);

printf("Enter first CPU burst time:"); scanf("%d",&p->bt);

p->bt1 = p->bt;

p->next = NULL;

if(first==NULL)

first=p; else last->next=p;

last = p;

}

}

void print\_output()

{

NODE \*p; float avg\_tat=0,avg\_wt=0;

printf("pname\tat\tbt\tct\ttat\twt\n");

p = first;

while(p!=NULL)

{

int tat = p->ct-p->at;

int wt = tat-p->bt;

avg\_tat+=tat;

avg\_wt+=wt;

printf("%s\t%d\t%d\t%d\t%d\t%d\n",

p->pname,p->at,p->bt,p->ct,tat,wt);

p=p->next;

}

printf("Avg TAT=%f\tAvg WT=%f\n",

avg\_tat/n,avg\_wt/n);

}

void print\_input()

{

NODE \*p;

p = first;

printf("pname\tat\tbt\n");

while(p!=NULL)

{

printf("%s\t%d\t%d\n", p->pname,p->at,p->bt1); p = p->next;

}

}

void sort()

{

NODE \*p,\*q;

int t;

char name[20];

p = first; while(p->next!=NULL)

{

q=p->next; while(q!=NULL)

{

if(p->at > q->at)

{

strcpy(name,p->pname); strcpy(p->pname,q->pname);

strcpy(q->pname,name);

t = p->at; p->at = q->at;

q->at = t;

t = p->bt; p->bt = q->bt;

q->bt = t;

t = p->ct; p->ct = q->ct;

q->ct = t;

t = p->bt1; p->bt1 = q->bt1;

q->bt1 = t;

}

q=q->next;

}

p=p->next;

}

}

int time;

NODE \* get\_sjf()

{

NODE \*p,\*min\_p=NULL;

int min=9999;

p = first; while(p!=NULL)

{

if(p->at<=time && p->bt1!=0 &&

p->bt1<min)

{

min = p->bt1; min\_p = p;

}

p=p->next;

}

return min\_p;

}

struct gantt\_chart

{

int start; char pname[30]; int end;

}s[100],s1[100];

int k;

void sjfp()

{

int prev=0,n1=0;

NODE \*p;

while(n1!=n)

{

p = get\_sjf();

if(p==NULL)

{

time++; s[k].start = prev; strcpy(s[k].pname,"\*");

s[k].end = time;

prev = time;

k++;

}

else

{

time++; s[k].start = prev; strcpy(s[k].pname, p->pname); s[k].end = time;

prev = time;

k++;

p->ct = time;

p->bt1--;

if(p->bt1==0)

n1++;

}

print\_input();

sort();

}

}

void print\_gantt\_chart()

{ int i,j,m;

s1[0] = s[0];

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

{

if(strcmp(s[i].pname,s1[j].pname)==0)

s1[j].end = s[i].end; else s1[++j] = s[i];

}

printf("%d",s1[0].start);

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

{

m = (s1[i].end - s1[i].start);

for(k=0;k<m/2;k++)

printf("-");

printf("%s",s1[i].pname);

for(k=0;k<(m+1)/2;k++)

printf("-");

printf("%d",s1[i].end);

}

}

int main()

{

accept\_info(); sort();

sjfp(); print\_output();

print\_gantt\_chart();

return 0;

}

1. [Write a simulation program to implement Pre-emptive Priority CPU scheduling algorithm. Accept the number of processes, arrival time, CPU burst time and priority for each process as input. Priorities should in High to Low order (1 is High). The output should give the Gantt chart, turnaround time and waiting time for each process.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-12-2-iprogramx.html)

[Also display the average turnaround time and average waiting time.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-12-2-iprogramx.html)

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct process\_info

{

char pname[20]; int at,bt,ct,bt1,p; struct process\_info \*next;

}NODE;

int n;

NODE \*first,\*last;

void accept\_info()

{

NODE \*p;

int i;

printf("Enter no.of process:");

scanf("%d",&n);

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

{

p = (NODE\*)malloc(sizeof(NODE));

printf("Enter process name:");

scanf("%s",p->pname);

printf("Enter arrival time:");

scanf("%d",&p->at);

printf("Enter first CPU burst time:"); scanf("%d",&p->bt);

printf("Enter priority:");

scanf("%d",&p->p);

p->bt1 = p->bt;

p->next = NULL;

if(first==NULL) first=p; else last->next=p;

last = p;

}

}

void print\_output()

{

NODE \*p; float avg\_tat=0,avg\_wt=0;

printf("pname\tat\tbt\tp\ttct\ttat\twt\n");

p = first; while(p!=NULL) {

int tat = p->ct-p->at;

int wt = tat-p->bt;

avg\_tat+=tat;

avg\_wt+=wt;

printf("%s\t%d\t%d\t%d\t%d\t%d\t%d\n",

p->pname,p->at,p->bt,p->p,p->ct,tat,wt);

p=p->next;

}

printf("Avg TAT=%f\tAvg WT=%f\n",

avg\_tat/n,avg\_wt/n);

}

void print\_input()

{

NODE \*p;

p = first;

printf("pname\tat\tbt\tp\n");

while(p!=NULL)

{

printf("%s\t%d\t%d\t%d\n", p->pname,p->at,p->bt1,p->p);

p = p->next;

}

}

void sort()

{

NODE \*p,\*q;

int t;

char name[20];

p = first; while(p->next!=NULL)

{

q=p->next; while(q!=NULL)

{

if(p->at > q->at)

{

strcpy(name,p->pname); strcpy(p->pname,q->pname);

strcpy(q->pname,name);

t = p->at; p->at = q->at;

q->at = t;

t = p->bt; p->bt = q->bt;

q->bt = t;

t = p->ct; p->ct = q->ct;

q->ct = t;

t = p->bt1; p->bt1 = q->bt1;

q->bt1 = t;

t = p->p; p->p = q->p;

q->p = t;

}

q=q->next;

}

p=p->next;

}

}

int time;

NODE \* get\_p()

{

NODE \*p,\*min\_p=NULL;

int min=9999;

p = first; while(p!=NULL)

{

if(p->at<=time && p->bt1!=0 && p->p<min)

{

min = p->p; min\_p = p;

}

p=p->next;

}

return min\_p;

}

struct gantt\_chart

{

int start; char pname[30]; int end;

}s[100],s1[100];

int k;

void pnp()

{

int prev=0,n1=0;

NODE \*p;

while(n1!=n)

{

p = get\_p();

if(p==NULL)

{

time++; s[k].start = prev; strcpy(s[k].pname,"\*");

s[k].end = time;

prev = time; k++;

}

else

{

time++; s[k].start = prev; strcpy(s[k].pname, p->pname); s[k].end = time;

prev = time;

k++;

p->ct = time;

p->bt1--;

if(p->bt1==0) n1++;

}

print\_input();

sort();

}

}

void print\_gantt\_chart()

{ int i,j,m;

s1[0] = s[0];

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

{

if(strcmp(s[i].pname,s1[j].pname)==0)

s1[j].end = s[i].end; else

s1[++j] = s[i];

}

printf("%d",s1[0].start);

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

{

m = (s1[i].end - s1[i].start);

for(k=0;k<m/2;k++)

printf("-");

printf("%s",s1[i].pname);

for(k=0;k<(m+1)/2;k++)

printf("-");

printf("%d",s1[i].end);

}

}

int main()

{

accept\_info(); sort(); pnp(); print\_output();

print\_gantt\_chart();

return 0;

}

# 4. Non-Pre-emptive Shortest Job First (SJF)

[Write a simulation program to implement a Non-Pre-emptive Shortest](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-19-2-iprogramx.html)

[Job First (SJF) – CPU scheduling algorithm. Accept the number of Processes and arrival time and CPU burst time for each process as input. The output should give the Gantt chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-19-2-iprogramx.html)

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct process\_info

{

char pname[20]; int at,bt,ct,bt1;

struct process\_info \*next;

}NODE;

int n;

NODE \*first,\*last;

void accept\_info()

{

NODE \*p;

int i;

printf("Enter no.of process:"); scanf("%d",&n);

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

{

p = (NODE\*)malloc(sizeof(NODE));

printf("Enter process name:");

scanf("%s",p->pname);

printf("Enter arrival time:");

scanf("%d",&p->at);

printf("Enter first CPU burst time:"); scanf("%d",&p->bt);

p->bt1 = p->bt;

p->next = NULL;

if(first==NULL) first=p; else

last->next=p;

last = p;

}

}

void print\_output()

{

NODE \*p; float avg\_tat=0,avg\_wt=0;

printf("pname\tat\tbt\tct\ttat\twt\n");

p = first; while(p!=NULL)

{

int tat = p->ct-p->at;

int wt = tat-p->bt;

avg\_tat+=tat;

avg\_wt+=wt;

printf("%s\t%d\t%d\t%d\t%d\t%d\n",

p->pname,p->at,p->bt,p->ct,tat,wt);

p=p->next;

}

printf("Avg TAT=%f\tAvg WT=%f\n", avg\_tat/n,avg\_wt/n);

}

void print\_input()

{

NODE \*p;

p = first;

printf("pname\tat\tbt\n");

while(p!=NULL)

{

printf("%s\t%d\t%d\n", p->pname,p->at,p->bt1);

p = p->next;

}

}

void sort()

{

NODE \*p,\*q;

int t;

char name[20];

p = first; while(p->next!=NULL)

{

q=p->next; while(q!=NULL)

{

if(p->at > q->at)

{

strcpy(name,p->pname); strcpy(p->pname,q->pname);

strcpy(q->pname,name);

t = p->at; p->at = q->at;

q->at = t;

t = p->bt; p->bt = q->bt;

q->bt = t;

t = p->ct;

p->ct = q->ct;

q->ct = t;

t = p->bt1; p->bt1 = q->bt1;

q->bt1 = t;

}

q=q->next;

}

p=p->next;

}

}

int time;

NODE \* get\_sjf()

{

NODE \*p,\*min\_p=NULL;

int min=9999;

p = first; while(p!=NULL)

{

if(p->at<=time && p->bt1!=0 &&

p->bt1<min)

{

min = p->bt1; min\_p = p;

}

p=p->next;

}

return min\_p;

}

struct gantt\_chart

{

int start; char pname[30]; int end;

}s[100],s1[100];

int k;

void sjfnp()

{

int prev=0,n1=0;

NODE \*p;

while(n1!=n)

{

p = get\_sjf();

if(p==NULL)

{

time++; s[k].start = prev; strcpy(s[k].pname,"\*");

s[k].end = time;

prev = time; k++;

}

else

{

time+=p->bt1; s[k].start = prev; strcpy(s[k].pname, p->pname);

s[k].end = time;

prev = time;

k++;

p->ct = time;

p->bt1 = 0;

n1++;

}

print\_input();

sort();

}

}

void print\_gantt\_chart()

{ int i,j,m; s1[0] = s[0];

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

{

if(strcmp(s[i].pname,s1[j].pname)==0)

s1[j].end = s[i].end; else

s1[++j] = s[i];

}

printf("%d",s1[0].start);

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

{

m = (s1[i].end - s1[i].start);

for(k=0;k<m/2;k++)

printf("-");

printf("%s",s1[i].pname);

for(k=0;k<(m+1)/2;k++)

printf("-");

printf("%d",s1[i].end);

}

}

int main()

{

accept\_info(); sort(); sjfnp(); print\_output();

print\_gantt\_chart();

return 0;

}

## 5. non-pre-emptive Priority CPU scheduling

[Write a simulation program to implement non-pre-emptive Priority CPU scheduling algorithm. Accept the number of Processes and arrival time, CPU burst time and priority for each process as input. Priorities should in High to Low order (1 is High). The output should give the Gantt chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-20-2-iprogramx.html) #include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct process\_info

{

char pname[20]; int at,bt,ct,bt1,p; struct process\_info \*next;

}NODE;

int n;

NODE \*first,\*last;

void accept\_info()

{

NODE \*p;

int i;

printf("Enter no.of process:");

scanf("%d",&n);

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

{

p = (NODE\*)malloc(sizeof(NODE));

printf("Enter process name:");

scanf("%s",p->pname);

printf("Enter arrival time:");

scanf("%d",&p->at);

printf("Enter first CPU burst time:"); scanf("%d",&p->bt);

printf("Enter priority:");

scanf("%d",&p->p);

p->bt1 = p->bt;

p->next = NULL;

if(first==NULL) first=p; else last->next=p;

last = p;

}

}

void print\_output()

{

NODE \*p; float avg\_tat=0,avg\_wt=0;

printf("pname\tat\tbt\tp\ttct\ttat\twt\n");

p = first; while(p!=NULL)

{

int tat = p->ct-p->at;

int wt = tat-p->bt;

avg\_tat+=tat;

avg\_wt+=wt;

printf("%s\t%d\t%d\t%d\t%d\t%d\t%d\n",

p->pname,p->at,p->bt,p->p,p->ct,tat,wt);

p=p->next;

}

printf("Avg TAT=%f\tAvg WT=%f\n",

avg\_tat/n,avg\_wt/n);

}

void print\_input()

{

NODE \*p;

p = first;

printf("pname\tat\tbt\tp\n");

while(p!=NULL)

{

printf("%s\t%d\t%d\t%d\n", p->pname,p->at,p->bt1,p->p);

p = p->next;

}

}

void sort()

{

NODE \*p,\*q;

int t;

char name[20];

p = first; while(p->next!=NULL)

{

q=p->next; while(q!=NULL)

{

if(p->at > q->at)

{

strcpy(name,p->pname); strcpy(p->pname,q->pname);

strcpy(q->pname,name);

t = p->at; p->at = q->at;

q->at = t;

t = p->bt; p->bt = q->bt;

q->bt = t;

t = p->ct; p->ct = q->ct;

q->ct = t;

t = p->bt1; p->bt1 = q->bt1;

q->bt1 = t;

t = p->p; p->p = q->p; q->p = t;

}

q=q->next;

}

p=p->next;

}

}

int time;

NODE \* get\_p()

{

NODE \*p,\*min\_p=NULL;

int min=9999;

p = first; while(p!=NULL)

{

if(p->at<=time && p->bt1!=0 && p->p<min)

{

min = p->p; min\_p = p;

}

p=p->next;

}

return min\_p;

}

struct gantt\_chart

{

int start; char pname[30]; int end;

}s[100],s1[100];

int k;

void pnp()

{

int prev=0,n1=0;

NODE \*p;

while(n1!=n)

{

p = get\_p();

if(p==NULL)

{

time++; s[k].start = prev; strcpy(s[k].pname,"\*");

s[k].end = time;

prev = time;

k++;

}

else

{

time+=p->bt1; s[k].start = prev; strcpy(s[k].pname, p->pname); s[k].end = time;

prev = time;

k++;

p->ct = time;

p->bt1 = 0;

n1++;

}

print\_input();

sort();

}

}

void print\_gantt\_chart()

{ int i,j,m;

s1[0] = s[0];

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

{

if(strcmp(s[i].pname,s1[j].pname)==0)

s1[j].end = s[i].end; else s1[++j] = s[i];

}

printf("%d",s1[0].start); for(i=0;i<=j;i++)

{

m = (s1[i].end - s1[i].start);

for(k=0;k<m/2;k++)

printf("-");

printf("%s",s1[i].pname);

for(k=0;k<(m+1)/2;k++)

printf("-");

printf("%d",s1[i].end);

}

}

int main()

{

accept\_info(); sort(); pnp(); print\_output();

print\_gantt\_chart();

return 0;

}

6.

[Write a program to implement a toy shell (Command Interpreter). It has its own prompt say “MyShell $”. Any normal shell command is executed from this shell (MyShell$) by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following commands:](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-4-2-iprogramx.html)

* [list f dirname : To print names of all the files in current directory](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-4-2-iprogramx.html) o [list n dirname : To print the number of all entries in the current directory.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-4-2-iprogramx.html)
* [list i dirname : To print names and inodes of the files in the current directory.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-4-2-iprogramx.html)

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0; char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p; p=strtok(NULL," ");

}

tok[i]=NULL;

}

void list(char \*dn, char op)

{

DIR \*dp; struct dirent \*entry;

int dc=0,fc=0;

dp = opendir(dn);

if(dp==NULL)

{

printf("Dir %s not found.\n",dn);

return;

}

switch(op)

{ case 'f':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_REG) printf("%s\n",entry->d\_name);

}

break; case 'n':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_DIR) dc++; if(entry->d\_type==DT\_REG) fc++;

}

printf("%d Dir(s)\t%d File(s)\n",dc,fc); break;

case 'i':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_REG)

printf("%s\t%d\n",entry->d\_name,entry->d\_fileno);

}

}

closedir(dp);

}

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$"); fflush(stdin); fgets(buff,80,stdin); buff[strlen(buff)-1]='\0'; make\_toks(buff,args); if(strcmp(args[0],"list")==0) list(args[2],args[1][0]); else

{

pid = fork(); if(pid>0) wait();

else

{

if(execvp(args[0],args)==-1) printf("Bad command.\n");

}

}

}

return 0;

}

1. [Write a program to implement a toy shell (Command Interpreter). It has its own prompt say “MyShell $”. Any normal shell command is executed from this shell (MyShell$) by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following commands:](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-12-2-iprogramx_25.html)

* [search f file name :- To search first occurrence of the pattern in the file](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-12-2-iprogramx_25.html)
* [search a file name :- To search all the occurrence of the pattern in the file](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-12-2-iprogramx_25.html)
* [search c file name :- To count the number of occurrence of the pattern in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-12-2-iprogramx_25.html)

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," "); while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void search(char \*fn, char op, char \*pattern)

{

int fh,count=0,i=0,j=0;

char buff[255],c,\*p;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s Not Found\n",fn);

return;

}

switch(op)

{ case 'f':

while(read(fh,&c,1))

{

buff[j++]=c; if(c=='\n')

{

buff[j]='\0'; j=0;

i++;

if(strstr(buff,pattern))

{

printf("%d: %s",i,buff); break;

}

}

}

break; case 'c':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0'; j=0; p = buff;

while(p=strstr(p,pattern))

{

count++;

p++;

}

}

}

printf("Total No.of Occurrences = %d\n",count);

break; case 'a':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0'; j = 0; i++; if(strstr(buff,pattern)) printf("%d: %s",i,buff);

}

}

}//switch close(fh);

}//search

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$"); fflush(stdin); fgets(buff,80,stdin); buff[strlen(buff)-1]='\0'; make\_toks(buff,args); if(strcmp(args[0],"search")==0) search(args[3],args[1][0],args[2]); else

{

pid = fork(); if(pid>0) wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

1. [Write a program to implement a toy shell (Command Interpreter). It has its own prompt say “MyShell $”. Any normal shell command is executed from this shell (MyShell$) by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following commands:](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-14-2-iprogramx.html) o [typeline +n filename :- To print first n lines in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-14-2-iprogramx.html) o [typeline -n filename :- To print last n lines in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-14-2-iprogramx.html)

o [typeline a filename :- To print all the lines in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-14-2-iprogramx.html)

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p; p=strtok(NULL," ");

}

tok[i]=NULL;

}

void typeline(char \*fn, char \*op)

{ int fh,i,j,n; char c;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s not found.\n",fn); return;

}

if(strcmp(op,"a")==0)

{

while(read(fh,&c,1)>0) printf("%c",c); close(fh);

return;

}

n = atoi(op); if(n>0)

{

i=0; while(read(fh,&c,1)>0)

{

printf("%c",c); if(c=='\n') i++; if(i==n) break;

}

}

if(n<0)

{

i=0; while(read(fh,&c,1)>0) {

if(c=='\n') i++;

}

lseek(fh,0,SEEK\_SET); j=0; while(read(fh,&c,1)>0)

{

if(c=='\n') j++;

if(j==i+n) break;

}

while(read(fh,&c,1)>0)

{

printf("%c",c);

}

}

close(fh);

}

int main()

{

char buff[80],\*args[10]; int pid;

while(1)

{

printf("myshell$"); fflush(stdin); fgets(buff,80,stdin); buff[strlen(buff)-1]='\0'; make\_toks(buff,args); if(strcmp(args[0],"typeline")==0) typeline(args[2],args[1]);

else

{

pid = fork(); if(pid>0) wait(); else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

1. [Write a program to implement a toy shell (Command Interpreter). It has its own prompt say “MyShell $”. Any normal shell command is executed from this shell (MyShell$) by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following commands:](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-16-2-iprogramx.html)

o [count c filename :- To print number of characters in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-16-2-iprogramx.html) o [count w filename :- To print number of words in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-16-2-iprogramx.html) o [count 1 filename :- To print number of lines in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-16-2-iprogramx.html)

1. #include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p; p=strtok(NULL," ");

}

tok[i]=NULL;

}

void count(char \*fn, char op)

{

int fh,cc=0,wc=0,lc=0;

char c;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s not found.\n",fn); return;

}

while(read(fh,&c,1)>0)

{

if(c==' ') wc++;

else if(c=='\n')

{

wc++; lc++;

}

cc++;

}

close(fh);

switch(op)

{ case 'c':

printf("No.of characters:%d\n",cc); break; case 'w':

printf("No.of words:%d\n",wc);

break; case 'l':

printf("No.of lines:%d\n",lc); break;

}

}

int main()

{

char buff[80],\*args[10]; int pid;

while(1)

{

printf("myshell$"); fflush(stdin); fgets(buff,80,stdin); buff[strlen(buff)-1]='\0'; make\_toks(buff,args); if(strcmp(args[0],"count")==0) count(args[2],args[1][0]);

else

{

pid = fork(); if(pid>0) wait(); else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

} }

return 0;

}

[Write a program to implement a toy shell (Command Interpreter). It has its own prompt say “MyShell $”. Any normal shell command is executed from this shell (MyShell$) by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following commands:](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-17-2-iprogramx.html)

o [count c filename :- To print number of characters in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-17-2-iprogramx.html) o [count w filename :- To print number of words in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-17-2-iprogramx.html) o [count 1 filename :- To print number of lines in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-17-2-iprogramx.html)

1. #include <sys/types.h> #include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void count(char \*fn, char op)

{

int fh,cc=0,wc=0,lc=0;

char c;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s not found.\n",fn); return;

}

while(read(fh,&c,1)>0)

{

if(c==' ') wc++;

else if(c=='\n')

{

wc++; lc++;

}

cc++;

}

close(fh);

switch(op)

{ case 'c':

printf("No.of characters:%d\n",cc); break; case 'w':

printf("No.of words:%d\n",wc);

break; case 'l':

printf("No.of lines:%d\n",lc); break;

}

}

int main()

{

char buff[80],\*args[10]; int pid;

while(1)

{

printf("myshell$"); fflush(stdin); fgets(buff,80,stdin); buff[strlen(buff)-1]='\0'; make\_toks(buff,args); if(strcmp(args[0],"count")==0) count(args[2],args[1][0]);

else

{

pid = fork(); if(pid>0) wait(); else

{

if(execvp(args[0],args)==-1) printf("Bad command.\n");

}

}

}

return 0;

}

[Write a program to implement a toy shell (Command Interpreter). It has its own prompt say “MyShell $”. Any normal shell command is executed from this shell (MyShell$) by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following commands:](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-21-2-iprogramx.html)

o [count c filename :- To print number of characters in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-21-2-iprogramx.html) o [count w filename :- To print number of words in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-21-2-iprogramx.html) o [count 1 filename :- To print number of lines in the file.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-21-2-iprogramx.html)

1. #include <sys/types.h> #include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p; p=strtok(NULL," ");

}

tok[i]=NULL;

}

void count(char \*fn, char op)

{

int fh,cc=0,wc=0,lc=0;

char c;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s not found.\n",fn);

return;

}

while(read(fh,&c,1)>0)

{

if(c==' ') wc++; else if(c=='\n')

{

wc++; lc++;

}

cc++;

}

close(fh);

switch(op)

{ case 'c':

printf("No.of characters:%d\n",cc); break; case 'w':

printf("No.of words:%d\n",wc);

break; case 'l':

printf("No.of lines:%d\n",lc); break;

}

}

int main()

{

char buff[80],\*args[10]; int pid;

while(1)

{

printf("myshell$"); fflush(stdin); fgets(buff,80,stdin); buff[strlen(buff)-1]='\0'; make\_toks(buff,args); if(strcmp(args[0],"count")==0) count(args[2],args[1][0]);

else

{

pid = fork(); if(pid>0) wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

[Write a program to implement a Banker‟s Algorithm. Accept the total number of processes (n) and resource types (m) as input. Also accept the number of initial instances for each resource type. Allocation and Max of size “nxm” as input and perform the following operations:](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-23-2-iprogramx.html)

* [a) Calculate and display the final contents of Available array of size](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-23-2-iprogramx.html)

[“m”](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-23-2-iprogramx.html) o [b) Calculate and display the contents of Need matrix of size “nxm”](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-23-2-iprogramx.html) o [Using Safety and Resource-Request algorithm perform the following operations:](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-23-2-iprogramx.html)

* [a) Check whether system is in safe sate or not.](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-23-2-iprogramx.html)
* [b) If request of size “m” arrives from process Pi, can it be granted immediately by keeping system in safe state?](https://iprogramx.blogspot.com/2019/01/tybcs-os-syspro-slip-23-2-iprogramx.html)

2. #include<stdio.h>

#define MAX 10

int m,n,total[MAX],avail[MAX],alloc[MAX][MAX], max[MAX][MAX],need[MAX][MAX],work[MAX],finish[MAX], seq[MAX],request[MAX];

void accept()

{ int i,j;

printf("Enter no.of process:"); scanf("%d",&n);

printf("Enter no.of resource types:"); scanf("%d",&m);

printf("Enter total no.of resources of each resource type:\n");

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

{

printf("%c:",65+i); scanf("%d",&total[i]);

}

printf("Enter no.of allocated resources of each resource type by each process:\n");

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

{

printf("P%d:\n",i); for(j=0;j<m;j++)

{

printf("%c:",65+j); scanf("%d",&alloc[i][j]);

}

}

printf("Enter no.of maximum resources of each resource type by each process:\n");

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

{

printf("P%d:\n",i); for(j=0;j<m;j++)

{

printf("%c:",65+j); scanf("%d",&max[i][j]);

}

}

}

void calc\_avail()

{ int i,j,s; for(j=0;j<m;j++)

{

s=0; for(i=0;i<n;i++) s+=alloc[i][j];

avail[j] = total[j] - s;

}

}

void calc\_need()

{ int i,j;

for(i=0;i<n;i++) for(j=0;j<m;j++) need[i][j]=max[i][j]-alloc[i][j];

}

void print()

{ int i,j;

printf("\tAllocation\tMax\tNeed\n\t");

for(i=0;i<3;i++)

{

for(j=0;j<m;j++) printf("%3c",65+j); printf("\t");

}

printf("\n"); for(i=0;i<n;i++)

{

printf("P%d\t",i); for(j=0;j<m;j++) printf("%3d",alloc[i][j]); printf("\t"); for(j=0;j<m;j++) printf("%3d",max[i][j]); printf("\t"); for(j=0;j<m;j++) printf("%3d",need[i][j]); printf("\n");

}

printf("Available\n"); for(j=0;j<m;j++) printf("%3c",65+j); printf("\n"); for(j=0;j<m;j++) printf("%3d",avail[j]);

printf("\n");

}

int check(int s)

{ int i,j; i = s; do

{ if(!finish[i])

{

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

{

if(need[i][j]>work[j]) break;

}

if(j==m) return i;

}

i=(i+1)%n; }while(i!=s);

return -1;

}

void banker()

{

int i,j,k=0;

for(i=0;i<n;i++) finish[i]=0;

for(j=0;j<m;j++) work[j] = avail[j];

i=0; while((i=check(i))!=-1)

{

printf("Process P%d resource granted.\n",i); finish[i] = 1;

for(j=0;j<m;j++) work[j] += alloc[i][j];

printf("finish("); for(j=0;j<n;j++) printf("%d,",finish[j]); printf("\b)\nwork("); for(j=0;j<m;j++) printf("%d,",work[j]); printf("\b)\n");

seq[k++]=i;

i=(i+1)%n;

}

if(k==n)

{

printf("System is in safe state.\n"); printf("Safe sequence:"); for(j=0;j<n;j++) printf("P%d ",seq[j]);

}

else

{

printf("System is not in safe state.");

}

printf("\n");

}

int main()

{

int i,j,pno;

accept(); calc\_avail(); calc\_need(); print(); banker();

printf("Enter process no:"); scanf("%d",&pno);

printf("Enter resource request of process P%d\n",pno);

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

{

printf("%c:",65+j); scanf("%d",&request[j]);

}

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

{

if(request[j]>need[pno][j])

break;

}

if(j==m)

{

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

{

if(request[j]>avail[j]) break;

}

if(j==m)

{

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

{

avail[j]-=request[j]; alloc[pno][j]+=request[j]; need[pno][j]-=request[j]; print(); banker();

}

}

else printf("Process P%d must wait.\n",pno);

}

else printf("Process P%d has exceeded its maximum claim\n",pno);

return 0;

}