**Preemtive Schedulling Algorithms**

**1)Shortest Remaining Time First (SRTF)**

#include<stdio.h>

#include<stdlib.h>

/// -> Shortst remaining time first Preemtive Algorithm

/// -> Adnan Ismail Shah Muzavor

#define MAX 1000

typedef struct array

{

int b\_time; ///burst\_time

int a\_time; ///arrival time

int rem\_btime; ///manipulative btime

} Prc;

float avg(int \*arr,int n)

{

int i=0;

float sum=0.0;

for(i=0; i<n; i++) sum+=(float)(arr[i]);

return sum/(float)n;

}

void display(int \*tt,int \*wt,int n)

{

int i;

printf("\n");

printf("\nProcess \t Waiting Time \t Turnaround Time");

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

{printf("\n %d \t\t %d \t\t %d",i+1,wt[i],tt[i]);}

printf("\n Avg \t\t %.2f \t\t %.2f",avg(wt,n),avg(tt,n));

}

///Display the input/sorted/processed input

void raw\_display(int order[],Prc p1[],int n)

{

int i;

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

{printf(" p%d \t\t%d\t\t\t%d\n",order[i]+1,p1[i].a\_time,p1[i].b\_time); }

}

void SRTF(Prc \*p,int first\_p,int n)

{

int count=0; ///No processses completed as yet

int time=0,i=0,gantt\_len=0; ///We will increment timer b y one each time

int select\_process=-1;

int \*wt=(int\*)malloc(sizeof(int)\*n);

int \*tt=(int\*)malloc(sizeof(int)\*n);

int \*gantt\_processno=(int\*)malloc(sizeof(int)\*50);

int \*gantt\_time=(int\*)malloc(sizeof(int)\*50);

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

{

wt[i]=0;

tt[i]=0;

}

///Iterate until all processes are done

for(time=0; count!=n; time++)

{

int small=MAX;

select\_process=-1; ///No process is selected

///1) -> Process selection

///Find process which had arrived udring this time

///If many consider one with minimum rem\_btime

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

{

///If process had arrived and it's not completed process

if(p[i].a\_time<=time && p[i].rem\_btime!=0)

{

///If no0 processs was selected select this

if(select\_process==-1)select\_process=i;

///If this process has lesser remaining burst time

else if(p[i].rem\_btime<p[select\_process].rem\_btime) select\_process=i;

}

}

///2) Since we are executing at rate of 1 second

/// decrese the burst time for this process by only if selected

///If process was selected

if(select\_process!=-1)

{

///add detials in gantt chart

gantt\_processno[gantt\_len]=select\_process;

gantt\_time[gantt\_len]=time;

gantt\_len++;

///decrement it's b time by one

p[select\_process].rem\_btime--;

///If process completed it's execution, calculate waiting and turn around time

if(p[select\_process].rem\_btime==0)

{

wt[select\_process]=time-p[select\_process].b\_time-p[select\_process].a\_time+1; ///+1 because execution will finish at time+1

tt[select\_process]=wt[select\_process]+p[select\_process].b\_time;

///Increment counter by one as one process completed the execution

count++;

}

}

}

display\_gantt(gantt\_processno,gantt\_time,time,gantt\_len);

display(tt,wt,n);

}

void display\_gantt(int \*gantt\_processno,int \* gantt\_time,int time,int gantt\_len)

{

int i;

printf("\n");

for(i=0; i<gantt\_len; i++)

{printf("-----");}

printf("\n");

for(i=0; i<gantt\_len; i++)

{ printf("| p%d ",gantt\_processno[i]+1); }

printf("|");

printf("\n");

for(i=0; i<gantt\_len; i++)

{printf("-----");}

printf("\n");

for(i=0; i<gantt\_len; i++)

{

if(gantt\_time[i]>=10) printf("%d ",gantt\_time[i]);

else printf(" %d ",gantt\_time[i]);

}

printf("%d",time);

}

///To take inputs

void input(Prc \*p,int\* p\_order,int n)

{

int i;

int min\_a=1000,min\_p=-1,min\_b=1000; ///choose process which arrive first.If 2?choose one with min b\_time

for(i=0; i<n; i++) p\_order[i]=i;

printf("\nEnter the BURST time and ARRIVAL time for: \n");

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

{

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

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

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

p[i].rem\_btime=p[i].b\_time;

}

printf("Process\t\tArrival time\t\tBurst time\n");

raw\_display(p\_order,p,n);

}

int main()

{

int n,i,first\_p;

int min\_a=1000,min\_p=-1,min\_b=1000; ///choose process which arrive first.If 2?choose one with min b\_time

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

scanf("%d",&n);

Prc \*p1=(Prc\*)malloc(sizeof(Prc)\*n);

int \*pr\_order=(int\*)malloc(sizeof(int)\*n); ///For storing process order

input(p1,pr\_order,n);

printf("\nAfter Shortest Remaining Time First Schedulling: ");

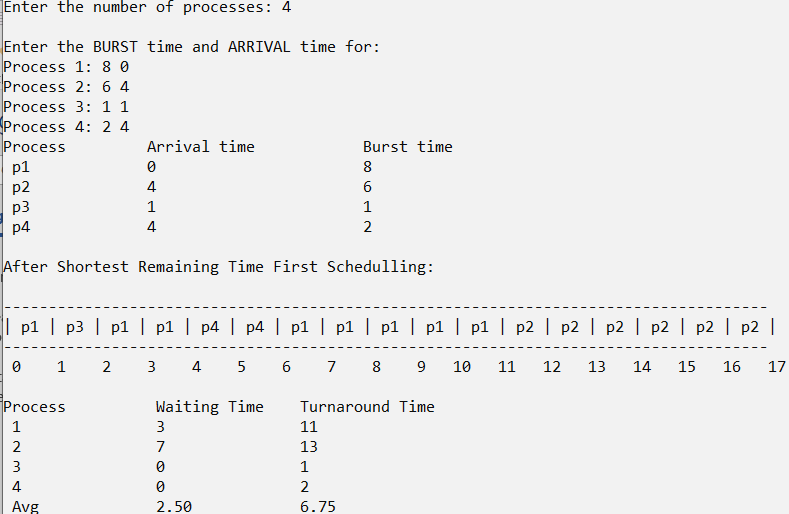
printf("\n");

SRTF(p1,pr\_order,n);

return 0;

}

**Output:**

****

**2) Round Robin Schedulling (RR)**

#include<stdio.h>

#include<stdlib.h>

/// -> Round Robin Schedulling

/// -> Adnan Ismail Shah Muzavor

float avg(int \*arr,int n)

{

int i=0;

float sum=0.0;

for(i=0; i<n; i++) sum+=(float)(arr[i]);

return sum/(float)n;

}

void display(int \*tt,int \*wt,int n)

{

int i;

printf("\nProcess \t Waiting Time \t Turnaround Time");

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

{

printf("\n %d \t\t %d \t\t %d",i+1,wt[i],tt[i]);

}

printf("\n Avg \t\t %.2f \t\t %.2f",avg(wt,n),avg(tt,n));

}

void RR(int \*pb,int n,int ts,int \*rem\_btime)

{

int \*wt=(int\*)malloc(sizeof(int)\*n);

int \*tt=(int\*)malloc(sizeof(int)\*n);

int \*gantt\_processno=(int\*)malloc(sizeof(int)\*50);

int \*gantt\_time=(int\*)malloc(sizeof(int)\*50);

int time=0,i=0,gantt\_len=0;

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

{

wt[i]=0;

tt[i]=0;

}

while(1)

{

int c=1; ///Assume all processes are done/completed

///Every tiem loop will start from process 01 i.e first process

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

{

if(rem\_btime[i]>0) c=0; ///Process is left

///Processs has sufficent burst time left

if(rem\_btime[i]>=ts)

{

rem\_btime[i]-=ts; ///Process executed for given time slice

gantt\_processno[gantt\_len]=i;

gantt\_time[gantt\_len]=time;

gantt\_len++;

time+=ts;

if(rem\_btime[i]==0)

{

///Process completed, find TT and WT

wt[i]=time-pb[i]; ///Method in reference text book

tt[i]=wt[i]+pb[i];///TT=WT+BT

continue;

}

}

///Process remaining time is lesser then time slice

else if(rem\_btime[i]!=0 && rem\_btime[i]<ts)

{

gantt\_processno[gantt\_len]=i;

gantt\_time[gantt\_len]=time;

gantt\_len++;

time+=rem\_btime[i]; ///Burst time lesser then time slice hence we add the btime

rem\_btime[i]=0;

wt[i]=time-pb[i]; ///Method in reference text book

tt[i]=wt[i]+pb[i];///TT=WT+BT

}

}

if(c==1)

{///Completed is true, hence break the outer loop

break; }

}

display\_gantt(gantt\_processno,gantt\_time,time,gantt\_len);

display(tt,wt,n);

}

void display\_gantt(int \*gantt\_processno,int \* gantt\_time,int time,int gantt\_len)

{

int i;

printf("\n");

for(i=0; i<gantt\_len; i++){printf("-----");}

printf("\n");

for(i=0; i<gantt\_len; i++){ printf("| p%d ",gantt\_processno[i]+1);}

printf("|");

printf("\n");

for(i=0; i<gantt\_len; i++){printf("-----");}

printf("\n");

for(i=0; i<gantt\_len; i++)

{

if(gantt\_time[i]>=10) printf("%d ",gantt\_time[i]);

else printf(" %d ",gantt\_time[i]);

}

printf("%d",time);

printf("\n");

}

int main()

{

int n,i,first\_p,ts;

int min\_a=1000,min\_p=-1,min\_b=1000; ///choose process which arrive first.If 2?choose one with min b\_time

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

scanf("%d",&n);

int \*p\_btime=(int\*)malloc(sizeof(int)\*n); ///number of processes

int \*btime\_cpy=(int\*)malloc(sizeof(int)\*n); ///Manipulative burst time

int \*pr\_order=(int\*)malloc(sizeof(int)\*n); ///For storing process order

printf("\nEnter the burst time for each process: \n");

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

{

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

scanf("%d",&p\_btime[i]);

btime\_cpy[i]=p\_btime[i];

}

printf("\nEnter the time slice: ");

scanf("%d",&ts);

printf("\n");

printf("Process \t Burst Time");

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

{printf("\np%d \t\t %d",i+1,p\_btime[i]); }

printf("\n");

printf("\nAfter Round Robin Schedulling: ");

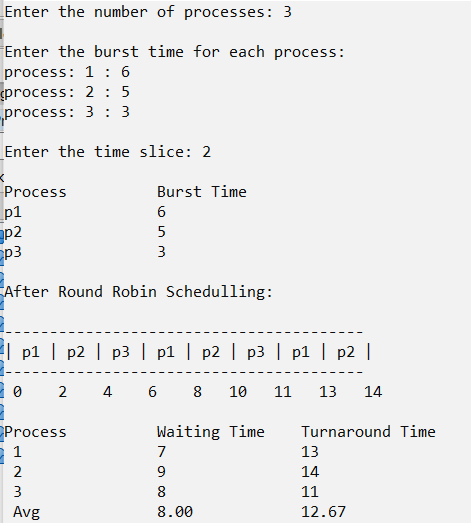
printf("\n");

RR(p\_btime,n,ts,btime\_cpy);

return 0;

}

**Output:**

****

***3) Preemtive Priority Schedulling (PP)***

#include<stdio.h>

#include<stdlib.h>

/// -> Preemtive Priority Schedulling

/// -> Adnan Ismail Shah Muzavor

#define MAX 1000

typedef struct array

{

int b\_time; ///burst\_time

int a\_time; ///arrival time

int rem\_btime; ///manipulative btime

int priority; ///priority of process

} Prc;

float avg(int \*arr,int n)

{

int i=0;

float sum=0.0;

for(i=0; i<n; i++) sum+=(float)(arr[i]);

return sum/(float)n;

}

void display(int \*tt,int \*wt,int n)

{

int i;

printf("\n");

printf("\nProcess \t Waiting Time \t Turnaround Time");

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

{printf("\n %d \t\t %d \t\t %d",i+1,wt[i],tt[i]); }

printf("\n Avg \t\t %.2f \t\t %.2f",avg(wt,n),avg(tt,n));

}

///Display the input/sorted/processed input

void raw\_display(int order[],Prc p1[],int n)

{

int i;

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

{printf(" p%d \t\t%d\t\t\t%d\t\t\t%d\n",order[i]+1,p1[i].a\_time,p1[i].b\_time,p1[i].priority); }

}

void PPS(Prc \*p,int first\_p,int n)

{

int count=0; ///No processses completed as yet

int time=0,i=0,gantt\_len=0; ///We will increment timer b y one each time

int select\_process=-1;

int \*wt=(int\*)malloc(sizeof(int)\*n);

int \*tt=(int\*)malloc(sizeof(int)\*n);

int \*gantt\_processno=(int\*)malloc(sizeof(int)\*50);

int \*gantt\_time=(int\*)malloc(sizeof(int)\*50);

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

{wt[i]=0;

tt[i]=0; }

///Iterate until all processes are done

for(time=0; count!=n; time++)

{

int small=MAX;

select\_process=-1; ///No process is selected

///1) -> Process selection

///Find process which had arrived udring this time

///If many consider one with minimum rem\_btime

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

{

///If process had arrived and it's not completed process

if(p[i].a\_time<=time && p[i].rem\_btime!=0)

{

///If no0 processs was selected select this

if(select\_process==-1)select\_process=i;

///If this process has higher priority, execute it

else if(p[i].priority<p[select\_process].priority) select\_process=i;

///If this process has same priority but lesser remaining burst time

else if(p[i].priority==p[select\_process].priority && p[i].rem\_btime<p[select\_process].rem\_btime) select\_process=i;

}

}

///2) Since we are executing at rate of 1 second

/// decrese the burst time for this process by only if selected

///If process was selected

if(select\_process!=-1)

{

///add detials in gantt chart

gantt\_processno[gantt\_len]=select\_process;

gantt\_time[gantt\_len]=time;

gantt\_len++;

///decrement it's b time by one

p[select\_process].rem\_btime--;

///If process completed it's execution, calculate waiting and turn around time

if(p[select\_process].rem\_btime==0)

{

wt[select\_process]=time-p[select\_process].b\_time-p[select\_process].a\_time+1; ///+1 because execution will finish at time+1

tt[select\_process]=wt[select\_process]+p[select\_process].b\_time;

///Increment counter by one as one process completed the execution

count++;

}

}

}

display\_gantt(gantt\_processno,gantt\_time,time,gantt\_len);

display(tt,wt,n);

}

void display\_gantt(int \*gantt\_processno,int \* gantt\_time,int time,int gantt\_len)

{

int i;

printf("\n");

for(i=0; i<gantt\_len; i++)

{printf("-----");}

printf("\n");

for(i=0; i<gantt\_len; i++)

{printf("| p%d ",gantt\_processno[i]+1); }

printf("|");

printf("\n");

for(i=0; i<gantt\_len; i++)

{printf("-----"); }

printf("\n");

for(i=0; i<gantt\_len; i++)

{

if(gantt\_time[i]>=10) printf("%d ",gantt\_time[i]);

else printf(" %d ",gantt\_time[i]);

}

printf("%d",time);

}

///To take inputs

void input(Prc \*p,int\* p\_order,int n)

{

int i;

int min\_a=1000,min\_p=-1,min\_b=1000; ///choose process which arrive first.If 2?choose one with min b\_time

for(i=0; i<n; i++) p\_order[i]=i;

printf("\nEnter the BURST time, ARRIVAL time and PRIORITY for: \n");

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

{

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

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

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

scanf("%d",&p[i].priority);

p[i].rem\_btime=p[i].b\_time;

}

printf("Process\t\tArrival time\t\tBurst time\t\tPriority\n");

raw\_display(p\_order,p,n);

}

int main()

{

int n,i,first\_p;

int min\_a=1000,min\_p=-1,min\_b=1000; ///choose process which arrive first.If 2?choose one with min b\_time

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

scanf("%d",&n);

Prc \*p1=(Prc\*)malloc(sizeof(Prc)\*n);

int \*pr\_order=(int\*)malloc(sizeof(int)\*n); ///For storing process order

input(p1,pr\_order,n);

printf("\nAfter Preemtive Priority Schedulling First Schedulling: ");

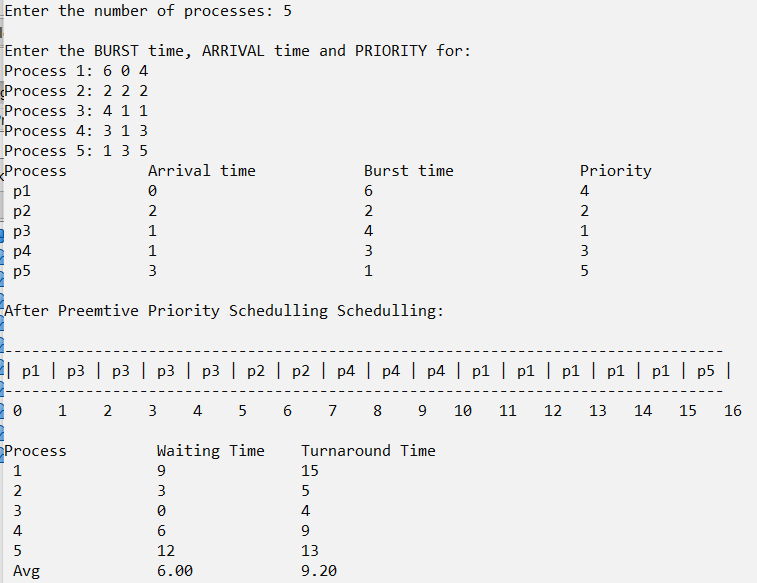
printf("\n");

PPS(p1,pr\_order,n);

return 0;

}

***Output:***

****