**1. Use the date and who commands in sequence (in one line) such that the output of date**

**will display on the screen and the output of who will be redirected to a file called**

**myfile2. Use the more command to check the contents of myfile2.**

date && (who>myfile2)

more myfile2

**2. Implement a program which show the difference between FCFS (without A.T), SJF(non-**

**pre-emptive) algorithm.**

|  |
| --- |
| #include<stdio.h> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| void fcfs(int p[],int b[n], int n){ |
| Int ct[n],tat[n],wt[n];  wt[0]=0  int tot=0; |

|  |
| --- |
| printf("FCFS:\n "); |
|  |

|  |
| --- |
| for (int i=0 ; i<n ; i++){ |
| Tot=tot+b[i] |
| ct[i]=tot |
| Tat[i]=ct[i]  Wt[i]=tat[i]-b[i] |

|  |
| --- |
|  |
|  |

|  |
| --- |
| } |
| printf("process\tbt\twt\ttat\tct\n");  for(i=0;i<n;i++)  {  printf("%d\t%d\t%d\t%d\t%d\n",p[i],bt[i],wt[i],tat[i],ct[i]);  } |

|  |
| --- |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| void sjf(int p[], int b[n],int n){ |
| int tat[n],wt[n],ct[n];  int a[n];  for (i=0;i<n;i++){  a[i]=b[i];  } |
|  |
|  |

|  |
| --- |
| int check=0;  int cta=0;  int ch[4];  for(i=0;i<4;i++)  {  int min=1000;  for(j=0;j<n;j++)  {  if(a[j]<min)  {  min=a[j];  ch[i]=j;  }  }  a[ch[i]]=1000;  printf("%d",ch[i]);  }  for(i=0;i<n;i++)  {  cta=cta+bt[ch[i]];  ct[ch[i]]=cta;  wt[ch[i]]=check;  check=check+bt[ch[i]];  tat[ch[i]]=ct[ch[i]];  }  Printf("SJF\n")  printf("process\tbt\twt\ttat\tct\n");  for(i=0;i<n;i++)  {  printf("%d\t%d\t%d\t%d\t%d\n",p[i],bt[i],wt[i],tat[i],ct[i]);  }  } |

|  |
| --- |
| void main(){ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int n; |
|  |

|  |
| --- |
| printf("enter number of processes\n"); |
|  |

|  |
| --- |
| scanf("%d",&n); |
|  |

|  |
| --- |
| int p[n],b[n]; |
|  |

|  |
| --- |
| printf("Enter BT of processes\n"); |
|  |

|  |
| --- |
| for (int i=0;i<n;i++){ |
| P[i]=i |
|  |

|  |
| --- |
| scanf("%d",&b[i]); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| fcfs(p,b,n); |
|  |

|  |
| --- |
| sjf(p,b,n); |
|  |

|  |
| --- |
| return; |
|  |

}

**3 Simulate SJF (non-pre-emptive) Scheduling Algorithm**

#include<stdio.h>

main()

{

int p[4]={1,2,3,4};

int bt[4]={3,5,8,4};

int wt[4]={0,0,0,0};

int tat[4]={0,0,0,0};

int ct[4]={0,0,0,0};

int i,j;

int a[4]={3,5,8,4};//burst time and a are same

int check=0;

int cta=0;

int ch[4];

for(i=0;i<4;i++)

{

int min=1000;

for(j=0;j<4;j++)

{

if(a[j]<min)

{

min=a[j];

ch[i]=j;

}

}

a[ch[i]]=1000;

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

}

for(i=0;i<4;i++)

{

cta=cta+bt[ch[i]];

ct[ch[i]]=cta;

wt[ch[i]]=check;

check=check+bt[ch[i]];

tat[ch[i]]=ct[ch[i]];

}

printf("process\tbt\twt\ttat\tct\n");

for(i=0;i<4;i++)

{

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

}

}

**4. Use the who command and redirect the result to a file called myfile1. Use the more command**

**to see the contents of myfile1.**

who>myfile1

more myfile1

**5. Create a hierarchy (using fork () system call) of client processes and a single server**

**process. Implement server broadcast to all the client processes using shared memory.**

|  |
| --- |
| #include<sys/types.h> |
|  |

|  |
| --- |
| #include<sys/ipc.h> |
|  |

|  |
| --- |
| #include<sys/shm.h> |
|  |

|  |
| --- |
| #include<stdio.h> |
|  |

|  |
| --- |
| #include<string.h> |
|  |

|  |
| --- |
| #include<unistd.h> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| main(int argc, char const \*argv[]) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| int pid = fork(); |
|  |

|  |
| --- |
| if(pid==0){ |
|  |

|  |
| --- |
| int id; |
|  |

|  |
| --- |
| id = shmget((key\_t)1000,100,IPC\_CREAT|0777); |
|  |

|  |
| --- |
| printf("Client1 Reading\n"); |
|  |

|  |
| --- |
| char \*shmptr=(char \*)shmat(id,NULL,0); |
|  |

|  |
| --- |
| write(STDOUT\_FILENO,shmptr,50); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else{ |
|  |

|  |
| --- |
| int id; |
|  |

|  |
| --- |
| id = shmget((key\_t)1000,100,IPC\_CREAT|0777); |
|  |

|  |
| --- |
| printf("Server Writing\n"); |
|  |

|  |
| --- |
| char \*shmptr = (char \*)shmat(id,NULL,0); |
|  |

|  |
| --- |
| strcpy(shmptr,"Hi\n"); |
|  |

|  |
| --- |
| shmdt(shmptr); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| int pid1 = fork(); |
|  |

|  |
| --- |
| if(pid1==0){ |
|  |

|  |
| --- |
| int id; |
|  |

|  |
| --- |
| id = shmget((key\_t)1000,100,IPC\_CREAT|0777); |
|  |

|  |
| --- |
| printf("Client2 Reading\n"); |
|  |

|  |
| --- |
| char \*shmptr=(char \*)shmat(id,NULL,0); |
|  |

|  |
| --- |
| write(STDOUT\_FILENO,shmptr,50); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| return 0; |
|  |

}

**6. Write a program that takes a command –line argument and reports complete status of a**

**file and display whether it is directory, a file, or something else.**

|  |
| --- |
| #include<stdio.h> |
|  |

|  |
| --- |
| #include<sys/stat.h> |
|  |

|  |
| --- |
| int main(int argc, char const \*argv[]) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| if(argc !=2){ |
|  |

|  |
| --- |
| printf("provide file name\n"); |
|  |

|  |
| --- |
| return 0; |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| struct stat vinni; |
|  |

|  |
| --- |
| stat(argv[1],&vinni); |
|  |

|  |
| --- |
| printf("blocks: %d\n",vinni.st\_blocks); |
|  |

|  |
| --- |
| printf("block size: %d\n",vinni.st\_blksize); |
|  |

|  |
| --- |
| printf("size: %d\n",vinni.st\_size); |
|  |

|  |
| --- |
| printf("uid: %d\n",vinni.st\_uid); |
|  |

|  |
| --- |
| printf("gid: %d\n",vinni.st\_gid); |
|  |

|  |
| --- |
| printf((vinni.st\_mode & S\_IFDIR)?"Directory\n":"File\n" ); |
|  |

|  |
| --- |
| return 0; |
|  |

}

**7. Use the ps, ps lx, ps tree and ps -aux command to display the process attributes.**

**8. Create server process and multiple client processes for communicate messages using**

**message queue.**

****

|  |
| --- |
| #include<sys/types.h> |
|  |

|  |
| --- |
| #include<sys/ipc.h> |
|  |

|  |
| --- |
| #include<sys/msg.h> |
|  |

|  |
| --- |
| #include<stdio.h> |
|  |

|  |
| --- |
| #include<string.h> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| struct msgbuf { |
|  |

|  |
| --- |
| long mtype; /\* message type, must be > 0 \*/ |
|  |

|  |
| --- |
| char mtext[1]; /\* message data \*/ |
|  |

|  |
| --- |
| }; |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| main(){ |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| int pid; |
|  |

|  |
| --- |
| pid = fork(); |
|  |

|  |
| --- |
| if (pid==0){ |
|  |

|  |
| --- |
| struct msgbuf msg; |
|  |

|  |
| --- |
| int id = msgget(5000,IPC\_CREAT|0777); |
|  |

|  |
| --- |
| int id1 = msgrcv(id,&msg,sizeof(msg),1,0); |
|  |

|  |
| --- |
| printf("First client %s\n",msg.mtext); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else{ |
|  |

|  |
| --- |
| printf("Server Sending:\n"); |
|  |

|  |
| --- |
| struct msgbuf msg; |
|  |

|  |
| --- |
| int id = msgget(5000,IPC\_CREAT|0777); |
|  |

|  |
| --- |
| strcpy(msg.mtext,"Hello"); |
|  |

|  |
| --- |
| msg.mtype = 1; |
|  |

|  |
| --- |
| msgsnd(id,&msg,sizeof(msg),0); |
|  |

|  |
| --- |
| } |
|  |
| |  | | --- | | int pid1; | |  |  |  | | --- | | Pid1 = fork(); | |  |  |  | | --- | | if (pid1==0){ | |  |  |  | | --- | | struct msgbuf msg; | |  |  |  | | --- | | int id = msgget(5000,IPC\_CREAT|0777); | |  |  |  | | --- | | int id1 = msgrcv(id,&msg,sizeof(msg),2,0); | |  |  |  | | --- | | printf("Second client %s\n",msg.mtext); | |  |  |  | | --- | | } | |

}

**9. Use system call to get the attributes of a file.**

stat filename

**10. Create a hierarchy (using fork() system call) of client processes as shown in the figure**

**and a single server process. Implement server broadcast to all the client processes using**

**message queues.**

#include<stdio.h>

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

struct msgq{

        long mtype;

        char msg[200];

};

int main(){

         struct msgq m1,m2,m3;

                int i,mid,m1id,m2id,m3id;

                m1.mtype=1;

                m2.mtype=2;

                m3.mtype=3;

                //printf("parent process\n");

                mid=msgget(7000,IPC\_CREAT|0777);

                printf("enter the message to first client\n");

                scanf("%[^\n]",&m1.msg);

                char ch;

                scanf("%c",&ch);

                printf("enter the message to second client\n");

                scanf("%[^\n]",&m2.msg);

                char ch1;

                scanf("%c",&ch1);

                printf("enter the message to third client\n");

                scanf("%[^\n]",&m3.msg);

                m1id=msgsnd(mid,&m1,sizeof(m1),IPC\_CREAT|0777);

                m2id=msgsnd(mid,&m2,sizeof(m2),IPC\_CREAT|0777);

                m3id=msgsnd(mid,&m3,sizeof(m3),IPC\_CREAT|0777);

        int id=fork();

        if (id==0){

                printf("child1 process\n");

                struct msgq m1;

                        int mid1,msgid1;

                        mid1=msgget(7000,IPC\_CREAT|0777);

                        m1.mtype=1;

                        msgid1=msgrcv(mid1,&m1,sizeof(m1),1,0777);

                        printf("%s\n",m1.msg);

                int id2=fork();

                if(id2==0){

                        printf("child-child process\n");

                        struct msgq m3;

                        int mid3,msgid3;

                        mid3=msgget(7000,IPC\_CREAT|0777);

                        m3.mtype=3;

                        msgid3=msgrcv(mid3,&m3,sizeof(m3),3,0777);

                        printf("%s\n",m3.msg);

                        printf("he");

                }

                else{

                //      printf("%d\n",getpid());

                }

        }

        else{

                int id1=fork();

                if(id1==0){

                        printf("child2 process\n");

                        struct msgq m2;

                        int mid2,msgid2;

                        mid2=msgget(7000,IPC\_CREAT|0777);

                        m2.mtype=2;

                        msgid2=msgrcv(mid2,&m2,sizeof(m2),2,0777);

                 printf("%s\n",m2.msg);

                }

        }

        return 0;

}

**11. Learn the top command to display the resource utilization statistics of processes**

**• Open a terminal and type the top command**

**• Start a browser and see the effect on the top display**

**• Compile a C program and observe the same effect (Use a long loop -say while(1) to observe**

**the effect)**

**• From the top display, answer the following:**

**– How much memory is free in the system? Which process is taking more CPU?**

**– Which process has got maximum memory share?**

top

**12 Implement multiple Clients-Server models using shared memory.**

****

|  |
| --- |
| #include<sys/types.h> |
|  |

|  |
| --- |
| #include<sys/ipc.h> |
|  |

|  |
| --- |
| #include<sys/shm.h> |
|  |

|  |
| --- |
| #include<stdio.h> |
|  |

|  |
| --- |
| #include<string.h> |
|  |

|  |
| --- |
| #include<unistd.h> |
|  |

|  |
| --- |
|  |
|  |

|  |
| --- |
| main(int argc, char const \*argv[]) |
|  |

|  |
| --- |
| { |
|  |

|  |
| --- |
| int pid = fork(); |
|  |

|  |
| --- |
| if(pid==0){ |
|  |

|  |
| --- |
| int id; |
|  |

|  |
| --- |
| id = shmget((key\_t)1000,100,IPC\_CREAT|0777); |
|  |

|  |
| --- |
| printf("Client1 Reading\n"); |
|  |

|  |
| --- |
| char \*shmptr=(char \*)shmat(id,NULL,0); |
|  |

|  |
| --- |
| write(STDOUT\_FILENO,shmptr,50); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| else{ |
|  |

|  |
| --- |
| int id; |
|  |

|  |
| --- |
| id = shmget((key\_t)1000,100,IPC\_CREAT|0777); |
|  |

|  |
| --- |
| printf("Server Writing\n"); |
|  |

|  |
| --- |
| char \*shmptr = (char \*)shmat(id,NULL,0); |
|  |

|  |
| --- |
| strcpy(shmptr,"Hi\n"); |
|  |

|  |
| --- |
| shmdt(shmptr); |
|  |

|  |
| --- |
| } |
|  |

|  |
| --- |
| int pid1 = fork(); |
|  |

|  |
| --- |
| if(pid1==0){ |
|  |

|  |
| --- |
| int id; |
|  |

|  |
| --- |
| id = shmget((key\_t)1000,100,IPC\_CREAT|0777); |
|  |

|  |
| --- |
| printf("Client2 Reading\n"); |
|  |

|  |
| --- |
| char \*shmptr=(char \*)shmat(id,NULL,0); |
|  |

|  |
| --- |
| write(STDOUT\_FILENO,shmptr,50); |
|  |

|  |
| --- |
| } |
|  |
| |  | | --- | | int pid2 = fork(); | |  |  |  | | --- | | if(pid2==0){ | |  |  |  | | --- | | int id; | |  |  |  | | --- | | id = shmget((key\_t)1000,100,IPC\_CREAT|0777); | |  |  |  | | --- | | printf("Client3 Reading\n"); | |  |  |  | | --- | | char \*shmptr=(char \*)shmat(id,NULL,0); | |  |  |  | | --- | | write(STDOUT\_FILENO,shmptr,50); | |  |  |  | | --- | | } | |

|  |
| --- |
| return 0; |
|  |

}

**13. Test the source code below:**

**for(i = 1;i ≤ 10;i + +){**

**fork();**

**printf(“The process with the PID=%d”,getpid());**

**}**

**In the next phase, modify the code, such as after all created processes have finished execution, in**

**a file process management.txt the total num-ber of created processes should be stored.**

#include<stdio.h>

#include<math.h>

main()

{

        int i;

        int count=0;

        for(i=0;i<11;i++)

        {

                fork();

                printf(" id:%d\n",getpid());

                count++;

        }

        //printf("%d",count);

        //int ans=pow(2,count);

        //printf("ans: %lf\n",ans);

        int base=2;

         long int  result = 1;

        //printf("Enter a base number: ");

        //scanf("%d", &base);

        //printf("Enter an exponent: ");

        //scanf("%d", &exponent);

        while (count != 0)

        {

                result \*= base;

                --count;

         }

    printf("Answer = %lld", result);

}

**14 Implement full duplex communication using pipes.**

#include<stdio.h>

#include<string.h>

main(int argc, char\* argv[argc]){

        int a[2];

        char t[20];

        char b[20];

        pipe(a);

        int pid=fork();

        if(pid==0){

                read(a[0],t,strlen(t));

                printf("%s\n",t);

                write(a[1],argv[2],5);

        }

        else{

                write(a[1],argv[1],5);

                sleep(1);

                read(a[0],b,10);

                printf("%s",b);

        }

**14** WAP **to create a child process and it should display the status of a file.**

#include<stdio.h>

#include<sys/stat.h>

#include<time.h>

main()

{

int pid=fork();

if(pid==0){

printf("CHild process\n");

struct stat var;

stat("./hyndavi",&var);

printf("id of device %d\n",var.st\_dev);

printf("inode no %d\n",var.st\_ino);

//printf((var.st\_mode & S\_IRUSR)? "r":"-");

printf("no of links created %d\n",var.st\_nlink);

printf("uid %d \n",var.st\_uid);

printf("gid %d \n",var.st\_gid);

printf("no of blocs %ld \n",var.st\_blocks);

printf("size %ld \n",var.st\_size);

printf("User permissions ");

printf((var.st\_mode & S\_IRUSR)? "r":"-");

printf((var.st\_mode & S\_IWUSR)? "w":"-");

printf((var.st\_mode & S\_IXUSR)? "e\n":"-\n");

printf("Group permissions ");

printf((var.st\_mode & S\_IRGRP)? "r":"-");

printf((var.st\_mode & S\_IWGRP)? "W":"-");

printf((var.st\_mode & S\_IXGRP)? "E\n":"-\n");

printf("Others permissions ");

printf((var.st\_mode & S\_IROTH)? "r":"-");

printf((var.st\_mode & S\_IWOTH)? "W":"-");

printf((var.st\_mode & S\_IXOTH)? "E":"-");

printf("\nChange time");

printf(ctime(&var.st\_ctime));

printf("\nA time");

printf(ctime(&var.st\_atime));

printf("\nModified time");

printf(ctime(&var.st\_mtime));

}

else{

printf("Parent process\n");

}

}

**15 Write programs using the following system calls of UNIX operating system: getenv() and setenv().**

#include<stdio.h>

#include<stdlib.h>

main(){

        printf("lang: %s\n",getenv("LANG"));

        printf("pwd: %s\n",getenv("PWD"));

        printf("login-name: %s\n",getenv("LOGNAME"));

        setenv("LOGNAME","cherry",1);

        printf("login-name: %s\n",getenv("LOGNAME"));

        unsetenv("LOGNAME");

        printf("name: %s\n",getenv("LOGNAME"));

        printf("path: %s\n",getenv("PATH"));

        setenv("PATH","C:/Users/Desktop",0);

        printf("changed-path: %s\n",getenv("PATH"));

}

**16 Implement half duplex communication using pipes**

#include<stdio.h>

#include<string.h>

main(){

        //char a1[5];

        int a[2];

        pipe(a);

        //int pid=fork();

        char s[]="my name is vinnie";

        int l=strlen(s);

        printf("%d\n",l);

        char b[2]="cbit";

        printf("%s\n",b);

        char a1[l];

        int pid=fork();

        if(pid==0)

        {

                read(a[0],a1,l);

                printf("%s",a1);

        }

        else{

                write(a[1],s,l);

        }

}

**17 WAP using FCFS using Arrival time.**

#include<stdio.h>

main()

{

        int p[4]={1,2,3,4};

        int bt[4]={3,5,8,4};

        int at[4]={3,2,4,1};

        int wt[4],tat[4],ct[4];

        int i,j,ctime=0;

        int a[4]={3,2,4,1};//arrival time and a are same

        int ch[4],minat;

        for(i=0;i<4;i++)

        {

                int min=1000;

                for(j=0;j<4;j++)

                {

                        if(a[j]<min)

                        {

                                min=a[j];

                                ch[i]=j;

                        }

                }

                a[ch[i]]=1000;

                //printf("%d",ch[i]);

        }

        minat=at[ch[0]];

        for(i=0;i<4;i++)

        {

                at[i]-=minat;

                ctime+=bt[ch[i]];

                ct[ch[i]]=ctime;

        }

        for(i=0;i<4;i++)

        {

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

                wt[ch[i]]=tat[ch[i]]-bt[ch[i]];

        }

        printf("process\tbt[]\tat[]\tct[]\ttat[]\twt[]\n");

        for(i=0;i<4;i++)

        {

                printf("p[%d]\t%d\t%d\t%d\t%d\t%d\n",i+1,bt[i],at[i],ct[i],tat[i],wt[i]);

        }

}

**19** **Wap to demonstrate orphan and zombie processes.**

**Orphan**

#include<stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

// Create a child process

int i = fork();

if (i > 0)

printf("in parent process : pid = %d \t parent id = %d \n",getpid(),getppid());

// Note that pid is 0 in child process

// and negative if fork() fails

else if (i == 0)

{

sleep(30);

printf("in child process : pid = %d \t parent id = %d\n",getpid(),getppid());

}

return 0;

}

**Zombie**

#include <sys/wait.h>

#include<stdio.h>

void ChildProc(void)

{

int i;

for(i = 0; i < 100; i++)

{

printf("Child : %d from process <%d> : Parent <%d>\n",i, getpid(), getppid());

}

}

void ParentProc(void)

{

int i;

for(i = 0; i < 20; i++)

{

printf("Parent : %d from process <%d> : Parent <%d>\n",i, getpid(), getppid());

}

}

int main(void)

{

pid\_t child;

child = fork();

if(child == 0)

{

ChildProc();

}

else

{

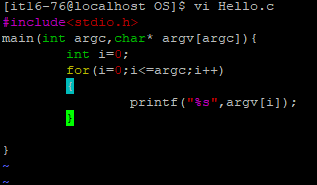
ParentProc();

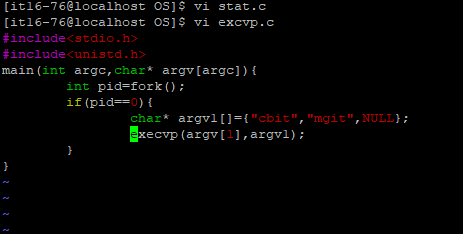
//waitpid(child,NULL,0);

}

}

**20 Wap using fork() and execvp()**

****

****

**21 WAP using Round Robin algorithm (time stamp 3ms)**

#include<stdio.h>

int main()

{

int i=0, limit, total = 0, x, counter = 0, time\_quantum;

int wait\_time = 0, turnaround\_time = 0, arrival\_time[10], burst\_time[10],tat[10],wt[10], temp[10],ct[10];

float average\_wait\_time, average\_turnaround\_time;

printf("\nEnter Total Number of Processes:\t");

scanf("%d", &limit);

x = limit;

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

{

printf("\nEnter Details of Process[%d]\n", i + 1);

printf("Arrival Time:\t");

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

printf("Burst Time:\t");

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

temp[i] = burst\_time[i];

}

printf("\nEnter Time Quantum:\t");

scanf("%d", &time\_quantum);

printf("\nProcess ID\t\tBurst Time\t Turnaround Time\t Waiting Time\n Completion time \n");

while(x != 0)

{

if(arrival\_time[i] > total)

{

i++;

}

if(temp[i] <= time\_quantum && temp[i] > 0)

{

total = total + temp[i];

temp[i] = 0;

ct[i]=total;

tat[i]=ct[i]-arrival\_time[i];

wt[i]=tat[i]-burst\_time[i];

counter = 1;

}

else if(temp[i] > 0)

{

temp[i] = temp[i] - time\_quantum;

total = total + time\_quantum;

}

if(temp[i] == 0 && counter == 1)

{

x--;

printf("\nProcess[%d]\t\t%d\t\t %d\t\t\t %d\t\t\t %d", i + 1, burst\_time[i], tat[i], wt[i],ct[i]);

counter = 0;

}

if(i == limit - 1)

{

i = 0;

}

else

{

i = (i++)%limit;

}

}

return 0;

}

**22 SJF premptive**

#include<stdio.h>

int main()

{

int at[10],bt[10],rt[10],endTime,i,smallest;

int remain=0,n,time,sum\_wait=0,sum\_turnaround=0;

printf("Enter no of Processes : ");

scanf("%d",&n);

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

{

printf("Enter arrival time for Process P%d : ",i+1);

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

printf("Enter burst time for Process P%d : ",i+1);

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

rt[i]=bt[i];

}

printf("\n\nProcess\t|Arrival Time|Burst Time|Completion Time|Turnaround Time| Waiting Time\n\n");

rt[9]=9999;

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

{

smallest=9;

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

{

if(at[i]<=time && rt[i]<rt[smallest] && rt[i]>0)

{

smallest=i;

}

}

rt[smallest]--;

if(rt[smallest]==0)

{

remain++;

endTime=time+1;

printf("\nP[%d]\t|\t%d\t|\t%d\t|\t%d\t|\t%d\t|\t%d",smallest+1,at[smallest],bt[smallest],endTime,endTime-at[smallest],endTime-bt[smallest]-at[smallest]);

sum\_wait+=endTime-bt[smallest]-at[smallest];

sum\_turnaround+=endTime-at[smallest];

}

}

printf("\n\nAverage waiting time = %f\n",sum\_wait\*1.0/n);

printf("Average Turnaround time = %f",sum\_turnaround\*1.0/5);

return 0;

}