OPERATING SYSTEM SEM-V

SLIP 1

Q:1)PROCESS:-

```
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{
        int pid=fork();
        if(pid>0)
        {
        printf("parent process\n");
        printf("ID:%d\n\n",getpid());
        }
        else if(pid==0)
        {
        printf("child process\n");
        printf("ID:%d\n",getpid());
        sleep(10);
        printf("\n child process\n");
        printf("ID:%d\n",getpid());
        printf("parent terminated then p-ID:%d\n",getpid());
        }
        else
        {
        printf("failed to create child process");
        }
                waitpid(pid,NULL,0);
                return 0;
}
```

Q:2)LFU

```
#include<stdio.h>
#define MAX 20
int frame[MAX],ref[MAX],mem[MAX][MAX],faults, sp, m, n, time[MAX];
void accept()
{
```

```
 int i;
 printf("Enter no of frames:");
 scanf("%d", &n);
 printf("Enter no.  of references:");
 scanf("%d", &m);
 printf("enter reference string:");
 for(i=0;i<m;i++)
 {
 printf("[%d]=",i);
 scanf("%d", &ref[i]);
 }
}
void disp()
{
        int i,j;

        for(i=0;i<m;i++)
        printf("%3d",ref[i]);

        printf("\n\n");

        for(i=0;i<n;i++)
        {
        for(j=0;j<m;j++)
        {
                if(mem[i][j])
                        printf("%3d",mem[i][j]);
                else
                        printf(" ");
        }
                printf("\n");
        }
        printf("total page faults:%d\n",faults);

}
int search(int pno)
{
```

```
        int i;
        for(i=0;i<n;i++)
        {
                if(frame[i]==pno)
                return i;
        }
        return-1;
}

int get_lru()
{
 int i,min_i,min=9999;
 for(i=0;i<n;i++)
 {
  if(time[i]<min)
  {
  min=time[i];
  min_i=i;
  }
 }
  return min_i;
}


void lru()
{
  int i, j ,k;
  for(i=0;i<m && sp<n;i++)
  {
   k=search(ref[i]);
   if(k==-1)
   {
    frame[sp]=ref[i];
    time[sp];
    faults++;
    sp++;
    for(j=0;j<n;j++)
```

```
    mem[j][i]=frame[j];
   }
  else
  time[k]=i;
 }
 for(i=0;i<m;i++)
 {
 k=search(ref[i]);
 if(k==-1)
 {
 sp=get_lru();
 frame[sp]=ref[i];
 time[sp]=i;
 faults++;
 for(j=0;j<n;j++)
 mem[j][i]=frame[j];
 }
 else
    time[k]=i;
 }
}


int main()
{
  accept();
  lru();
  disp();

  return 0;
}
```

SLIP 2

Q:1) CHILD PARENT PROCESS

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

```
#include<stdlib.h>
#include<unistd.h>
int main()
{
	pid_t pid=fork();
	if(pid==0)
	{
		printf("\nI am child process");
		printf("\n Process Id = %d",getpid());
		 printf("\n priority : %d = %d\n", nice(-7),getpid());
	}
	else if(pid>0)
	{
		printf("\nI am parent process");
		printf("\n process id=%d",getpid());
		printf("\n process id = %d",getpid());
		printf("\n priority : %d=%d\n", nice(15),getpid());
	}
	else
	{
		printf("fork() system call fails");
	}
	return 0;
 }
```

Q:2) RR

```
#include<stdio.h>
#include<string.h>
struct Process
{
	char PName[5];
	int AT,BT,TAT,WT,CT;
	int tempBT;
}P[10];

int N;
int TQ;
```

```
void Input()
{
	int i;
	printf("n\Enter Number of process:");
	scanf("%d",&N);
	for(i=0;i<N;i++)
	{
		printf("\nEnter details of process %d",i+1);
		printf("\nEnter process name:");
		scanf("%s",P[i].PName);
		printf("\nEnter AT:");
		scanf("%d",&P[i].AT);
		printf("\nEnter BT:");
		scanf("%d",&P[i].BT);
		P[i].tempBT = P[i].BT;
	}
	printf("\nEnter Time Quantum:");
	scanf("%d",&TQ);
}
void SortProcessAT()
{
	int i,j,tmp;
	char temp[5];
	for(i=0;i<N;i++)
		for(j=0;j<N;j++)
			if(P[i].AT < P[j].AT)
			{
				strcpy(temp,P[i].PName);
				strcpy(P[i].PName,P[j].PName);
				strcpy(P[j].PName,temp);

				tmp=P[i].AT;
				P[i].AT=P[j].AT;
				P[j].AT=tmp;

				tmp=P[i].BT;
				P[i].BT=P[j].BT;
```

```
                        P[j].BT=tmp;

                        tmp=P[i].tempBT;
                        P[i].tempBT=P[j].tempBT;
                        P[j].tempBT=tmp;
                }
}
void RR()
{
        int time=0;
        int finish=0;
        int i;
        int flag=0;
        while(finish!=1)
        {
                flag=0;
                for(i=0;i<N;i++)
                {
                        if (P[i].AT<=time && P[i].tempBT != 0);
                        {
                                flag=1;
                                printf("|%d %s",time,P[i].PName);
                                if(P[i].tempBT > TQ)
                                {
                                        time = time +TQ;
                                        P[i].tempBT=P[i].tempBT-TQ;
                                }else
                                {
                                        time = time+P[i].tempBT;
                                        P[i].tempBT=0;
                                        P[i].CT=time;
                                }
                                printf("%d|",time);
                        }
                }
                if(flag==0)
                {
```

```
                printf("|%d ##",time);
                time++;
                printf(" %d|",time);
            }
            for(i=0;i<N;i++)
                if(P[i].tempBT==0)
                    continue;
                else
                    break;
            if(i==N)
                finish=1;
        }
}
void Output()
{
        int i;
        int totalTAT=0;
        int totalWT=0;
        for(i=0;i<N;i++)
        {
            P[i].TAT = P[i].CT - P[i].AT;
            totalTAT = totalTAT + P[i].TAT;
            P[i].WT = P[i].TAT - P[i].BT;
            totalWT = totalWT + P[i].WT;
        }
        printf("\nProcess Details");
        printf("\n****");
        printf("\nPName\tAT\tBT\tTAT\tWT");
        printf("\n****");
        for(i=0;i<N;i++)
        {

        printf("\n%s\t%d\t%d\t%d\t%d",P[i].PName,P[i].AT,P[i].BT,P[i].TAT,P[i].WT);
        }
        printf("\n***\n");
        printf("\nAVG TAT:%d / %d",totalTAT,N);
        printf("\nAVG WT:%d / %d",totalWT,N);
```

```
}
int main()
{
        Input();
        SortProcessAT();
        RR();
        Output();
        return 0;
}
```

SLIP 3

Q:1)NICE

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
        int pid, retnice;
        printf("press DEL to stop process\n");
        pid=fork();
for(::)
{
        if(pid==0)
        {
        retnice=nice(5);
        printf("child gets higher priority %d \n",retnice);
        sleep(1);
        }
        else
        {
        retnice=nice(-4);
        printf("parent gets lower priority %d\n",retnice);
        sleep(1);
        }
}
}
```

Q:2)LRU

```
#include<stdio.h>
#define MAX 20
int frame[MAX],ref[MAX],mem[MAX][MAX],faults, sp, m, n, time[MAX];
void accept()
{
 int i;
 printf("Enter no of frames:");
 scanf("%d", &n);
 printf("Enter no. of references:");
 scanf("%d", &m);
 printf("enter reference string:");
 for(i=0;i<m;i++)
 {
 printf("[%d]=",i);
 scanf("%d", &ref[i]);
 }
}
void disp()
{
        int i,j;

        for(i=0;i<m;i++)
        printf("%3d",ref[i]);

        printf("\n\n");

        for(i=0;i<n;i++)
        {
        for(j=0;j<m;j++)
        {
                if(mem[i][j])
                        printf("%3d",mem[i][j]);
                else
                        printf(" ");
        }
                printf("\n");
```

```
        }
        printf("total page faults:%d\n", faults);

}
int search(int pno)
{
        int i;
        for(i=0;i<n;i++)
        {
                if(frame[i]==pno)
                return i;
        }
        return-1;
}
int get_lru()
{
 int i,min_i, min=9999;
 for(i=0;i<n;i++)
 {
  if(time[i]<min)
  {
  min=time[i];
  min_i=i;
  }
 }
  return min_i;
}
void lru()
{
  int i,j,k;
  for(i=0;i<m && sp<n;i++)
  {
   k=search(ref[i]);
   if(k==-1)
   {
    frame[sp]=ref[i];
    time[sp];
```

```
   faults++;
   sp++;
   for(j=0;j<n;j++)
    mem[j][i]=frame[j];
  }
 else
 time[k]=i;
}
for(i=0;i<m;i++)
{
k=search(ref[i]);
if(k==-1)
{
sp=get_lru();
frame[sp]=ref[i];
time[sp]=i;
faults++;
for(j=0;j<n;j++)
mem[j][i]=frame[j];
}
else
   time[k]=i;
}
}
int main()
{
  accept();
  lru();
  disp();

  return 0;
}
```

SLIP 04

Q:1)NICE

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
        int pid, retnice;
        printf("press DEL to stop process\n");
        pid=fork();
for(;;)
{
        if(pid==0)
        {
        retnice=nice(5);
        printf("child gets higher priority %d \n",retnice);
        sleep(1);
        }
        else
        {
        retnice=nice(-4);
        printf("parent gets lower priority %d\n",retnice);
        sleep(1);
        }
}
}
```

Q:2)LIST

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<dirent.h>
#include<fcntl.h>
void list (char c,char *dn)
{
DIR *dir;
int cnt=0;
struct dirent *entry;
```

```
struct stat buff;
if((dir=opendir(dn))==NULL)
{
printf("Directory %s not found\n",dn);
return;
}
switch(c)
{
case 'f':
while((entry=readdir(dir))!=NULL)
{
stat (entry->d_name, &buff);
if(S_IFREG&buff.st_mode)
printf("%s\n",entry->d_name);
}
break;
case 'n':
while((entry=readdir(dir))!=NULL)
cnt++;
printf("total no of entries=%d\n",cnt);
break;
case 'i':
while((entry=readdir(dir))!=NULL)
{
stat(entry->d_name, &buff);
if(S_IFREG&buff.st_mode)
printf("%s\t%d\n",entry->d_name, buff.st_ino);
}
break;
default:
printf("Invalid argument.....\n");
}
closedir(dir);
}
main()
{
char command[80],t1[20],t2[20],t3[20],t4[20];
```

```
int n;
system("clear");
while(1)
{
printf("myShell$");
fflush(stdin);
fgets(command,80,stdin);
n=sscanf(command,"%s %s %s",t1,t2,t3,t4);
switch(n)
{
case 1:
if(!fork())
{
execlp(t1,t1,NULL);
perror(t1);
}
break;
case 2:
if(!fork())
{
execlp(t1,t1,t2,NULL);
perror(t1);
}
break;
case 3:
if(strcmp(t1,"list")==0)
list(t2[0],t3);
else
{
if(!fork())
{
execlp(t1,t1,t2,t3,NULL);
perror(t1);
}
}
break;
case 4:
```

```
if(!fork())
{
execlp(t1,t1,t2,t3,t4,NULL);
perror(t1);
}
}
}
}
```

SLIP 05

Q:1)COUNT (EDIT)

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<dirent.h>
#include<fcntl.h>
#include<string.h>
void count(char c,char *fn)
{
        int lc=0,wc=0,cc=0,handle;
        char ch;
        if((handle=open(fn,O_RDONLY))==-1)
        {
                printf("file is not found\n",fn);
                return;
        }
        while(read(handle,&ch,1)!=0)
        {
                if(ch==' ')
                        wc++;
                else if(ch=='\n')
                        lc++;
                else
```

```
                        cc++;
          }
          close(handle);
          switch(c);
          {
             case 'c':
        printf("Total no. of characters=%d\n",cc);
        break;

       case 'c':
        printf("Total no. of words=%d\n",wc);
        break;

       case 'c':
        printf("Total no. of lines=%d\n",lc);
        break;
          }

}
int main()
{
          char command[80],t1[20],t2[20],t3[20],t4[20];
          int n;
          system("clear");
          while(1)
          {
          printf("myshells");
          fflush(stdin);
          fgets(command,80,stdin);
          n=sscanf(command,"%s %s %s %s",t1,t2,t3,t4);
          switch(n)
          {
          case 1:
                    if(!fork())
                    {
                              execlp(t1,t1,NULL);
                              perror(t1);
```

```
                }
                break;
        case 2:
                if(!fork())
                {
                        execlp(t1,t1,t2,NULL);
                        perror(t1);
                }
                break;
        case 3:
                if(strcmp(t1,"count")==0)
                        count(t2[0],t3);
                else
                {

                if(!fork())
                {
                        execlp(t1,t1,t2,t3,NULL);
                        perror(t1);
                }
                }
                break;
        case 4:
                if(!fork())
                {
                        execlp(t1,t1,t2,t3,t4,NULL);
                        perror(t1);
                }
                break;
        }
}
}
```

Q:2)FCFS

```
#include<stdio.h>
#include<string.h>
```

```
struct Process
{
 char PName[5];
 int AT,BT,TAT,WT,CT;
 int tempBT;
 }P[10];
 int N;
void Input()
{
   int i;
   printf("\nenter number of process:");
   scanf("%d",&N);
   for(i = 0;i < N;i++)
   {
      printf("\nenter details of process %d",i+1);
      printf("\nenter process name:");
      scanf("%s",P[i].PName);

      printf("\nenter AT:");
      scanf("%d",&P[i].AT);

      printf("\nenter BT:");
      scanf("%d",&P[i].BT);

      P[i].tempBT = P[i].BT;
    }
}
void SortProcessAT()
{
      int i;
      int j;
      int tva1;
      char temp[5];

      for(i = 0; i < N; i++)
          for(j = 0; j < N; j++)
          {
```

```
            if(P[i].AT<P[j].AT)
            {
             //swap pname
                strcpy(temp,P[i].PName);
                strcpy(P[i].PName,P[j].PName);
                strcpy(P[j].PName,temp);
            //swap at
                tva1 = P[i].AT;
                P[i].AT = P[j].AT;
                P[j].AT = tva1;
            //swap bt
                tva1 = P[i].BT;
                P[i].BT = P[j].BT;
                P[j].BT = tva1;
            //swap tempbt
                 tva1 = P[i].tempBT;
                P[i].tempBT = P[j].tempBT;
                P[j].tempBT = tva1;
            }
           }
}
void Output()
{
      int i;
      int totalTAT = 0;
      int totalWT = 0;
      for(i = 0; i < N; i++)
      {
         P[i].TAT = P[i].CT - P[i].AT;
         totalTAT = totalTAT + P[i].TAT;

         P[i].WT = P[i].TAT - P[i].BT;
         totalWT = totalWT + P[i].WT;
      }

      printf("\nProcess Details");
      printf("\n*****");
```

```
        printf("\nPName\tAT\tBT\tTAT\tWT");
        printf("\n*****");
        for(i = 0; i < N; i++)
        {
printf("\n%s\t%d\t%d\t%d\t%d",P[i].PName,P[i].AT,P[i].BT,P[i].TAT,P[i].WT);
        }
        printf("\n*****\n");
        printf("\nAVG TAT:%d / %d",totalTAT,N);
        printf("\nAVG WT:%d / %d",totalWT,N);
}
void FCFS()
{
         int time = 0;
         int finish = 0;
         int i;
         int flag = 0;
         while(finish !=1)
         {
            flag = 0;
            for(i = 0; i < N; i++)
            {
                 if(P[i].AT <= time && P[i].tempBT != 0)
                 {
                    printf("|%d %s",time,P[i].PName);
                    time = time + P[i].tempBT;
                    printf("%d|",time);
                    P[i].tempBT = 0;
                    P[i].CT = time;
                    flag = i;
                 }
            }
            if(flag ==0)
            {
               printf("|%d # ",time);
               time++;
               printf("%d|",time);
            }
```

```
            for(i = 0; i < N; i++)
                if(P[i].tempBT == 0)
                    continue;
                else
                    break;
            if(i == N)
                finish = 1;
        }
}
int main()
{
        Input();
        SortProcessAT();
        FCFS();
        Output();

        return 0;
 }
```

SLIP 06

q:1)count

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<dirent.h>
#include<fcntl.h>
#include<string.h>
void count(char c,char *fn)
{
         int lc=0,wc=0,cc=0,handle;
         char ch;
         if((handle=open(fn,O_RDONLY))==-1)
         {
                  printf("file is not found\n",fn);
                  return;
         }
```

```
        while(read(handle,&ch,1)!=0)
        {
                if(ch==' ')
                        wc++;
                else if(ch=='\n')
                        lc++;
                else
                        cc++;
        }
        close(handle);
        switch(c);
        {
           case 'c':
       printf("Total no. of characters=%d\n",cc);
       break;

      case 'c':
       printf("Total no. of words=%d\n",wc);
       break;

      case 'c':
       printf("Total no. of lines=%d\n",lc);
       break;
        }

}
int main()
{
        char command[80],t1[20],t2[20],t3[20],t4[20];
        int n;
        system("clear");
        while(1)
        {
        printf("myshells");
        fflush(stdin);
        fgets(command,80,stdin);
        n=sscanf(command,"%s %s %s %s",t1,t2,t3,t4);
```

```
switch(n)
{
case 1:
        if(!fork())
        {
                execlp(t1,t1,NULL);
                perror(t1);
        }
        break;
case 2:
        if(!fork())
        {
                execlp(t1,t1,t2,NULL);
                perror(t1);
        }
        break;
case 3:
        if(strcmp(t1,"count")==0)
                count(t2[0],t3);
        else
        {

        if(!fork())
        {
                execlp(t1,t1,t2,t3,NULL);
                perror(t1);
        }
        }
        break;
case 4:
        if(!fork())
        {
                execlp(t1,t1,t2,t3,t4,NULL);
                perror(t1);
        }
        break;
}
```

```
}
}
```

Q:2) mru

```
#include<stdio.h>
#define MAX 20

int frame[MAX], ref[MAX],mem[MAX][MAX],faults,sp,m,n,time[MAX];

void accept()
{
        int i;

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

        printf("Enter no of references:");
        scanf("%d",&m);

        printf("Enter reference string \n:");
        for(i=0;i<m;i++)
        {
                printf("[%d]=",i);
                scanf("%d",&ref[i]);
        }
}

void disp()
{
        int i,j;

        for(i=0;i<m;i++)
        printf("%3d",ref[i]);

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

```
                for(j=0;j<m;j++)
                {
                        if(mem[i][j])
                                printf("%d",mem[i][j]);
                        else
                                printf("   ");
                }
        printf("\n");
        }
        printf("Total Page Faults: %d\n",faults);
}

int search(int pno)
{
        int i;
        for(i=0;i<n;i++)
        {
                if(frame[i]==pno)
                return i;
        }
        return -1;
}


void get_mru()
{
        int i,max_i,max=9999;
        for(i=0;i<n;i++)
        {
                if(time[i]>max)
                {
                        max=time[i];
                        max_i=i;
                }
        }
        //return max_i;
}
```

```
void mru()
{
	int i,j,k;
	for(i=0;i<m && sp<n;i++)
	{
		k=search(ref[i]);
		if(k==-1)
		{
			frame[sp]=ref[i];
			time[sp]=i;
			faults++;
			sp++;

			for(j=0;j<n;j++)
				mem[j][i]=frame[j];
		}
		else
		time[k]=i;
	}
}


int main()
{
	accept();
	mru();
	disp();

	return 0;
}
```

SlIP 07

q:1)count

```
#include<stdio.h>
#include<sys/types.h>
```

```
#include<sys/stat.h>
#include<unistd.h>
#include<dirent.h>
#include<fcntl.h>
#include<string.h>
void count(char c,char *fn)
{
        int lc=0,wc=0,cc=0,handle;
        char ch;
        if((handle=open(fn,O_RDONLY))==-1)
        {
                printf("file is not found\n",fn);
                return;
        }
        while(read(handle,&ch,1)!=0)
        {
                if(ch==' ')
                        wc++;
                else if(ch=='\n')
                        lc++;
                else
                        cc++;
        }
        close(handle);
        switch(c);
        {
           case 'c':
       printf("Total no. of characters=%d\n",cc);
       break;

      case 'c':
       printf("Total no. of words=%d\n",wc);
       break;

      case 'c':
       printf("Total no. of lines=%d\n",lc);
       break;
```

```
        }

}
int main()
{
        char command[80],t1[20],t2[20],t3[20],t4[20];
        int n;
        system("clear");
        while(1)
        {
        printf("myshells");
        fflush(stdin);
        fgets(command,80,stdin);
        n=sscanf(command,"%s %s %s %s",t1,t2,t3,t4);
        switch(n)
        {
        case 1:
                if(!fork())
                {
                        execlp(t1,t1,NULL);
                        perror(t1);
                }
                break;
        case 2:
                if(!fork())
                {
                        execlp(t1,t1,t2,NULL);
                        perror(t1);
                }
                break;
        case 3:
                if(strcmp(t1,"count")==0)
                        count(t2[0],t3);
                else
                {

                if(!fork())
```

```
                {
                        execlp(t1,t1,t2,t3,NULL);
                        perror(t1);
                }
                }
                break;
        case 4:
                if(!fork())
                {
                        execlp(t1,t1,t2,t3,t4,NULL);
                        perror(t1);
                }
                break;
        }
}
}
```

q:2)fifo

```
#include<stdio.h>
#define MAX 20

int frame[MAX],ref[MAX],mem[MAX][MAX],faults,sp,m,n,time[MAX];
void accept()
{
        int i;
        printf("enter the no.of frame");
        scanf("%d",&n);
        printf("enter the reference:");
        scanf("%d",&m);
        printf("enter reference string:");
        for(i=0;i<m;i++)
        {
                printf("[%d]=",i);
                scanf("%d",&ref[i]);
                }
}
void disp()
```

```
{
        int i,j;

        for(i=0;i<m;i++)
        printf("%3d",ref[i]);

        printf("\n\n");

        for(i=0;i<n;i++)
        {
        for(j=0;j<m;j++)
        {
                if(mem[i][j])
                        printf("%3d",mem[i][j]);
                else
                        printf(" ");
        }
                printf("\n");
        }
        printf("total page faults:%d\n",faults);

}
int search(int pno)
{
        int i;
        for(i=0;i<n;i++)
        {
                if(frame[i]==pno)
                return i;
        }
        return-1;
}
void fifo()
{
        int i,j;
        for(i=0;i<m;i++)
        {
```

```
                if(search(ref[i])==-1)
                {
                        frame[sp]=ref[i];
                        sp=(sp+1)%n;
                        faults++;
                        for(j=0;j<n;j++)
                        mem[j][i]=frame[j];
                }
        }
}


int main()
{
        accept();
        fifo();
        disp();

        return 0;
}
```

Slip 08

q:1)nice

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
        int pid,retnice;
        printf("press DEL to stop process\n");
        pid=fork();
for(;;)
{
        if(pid==0)
        {
        retnice=nice(5);
        printf("child gets higher priority %d \n",retnice);
```

```
        sleep(1);
        }
        else
        {
        retnice=nice(-4);
        printf("parent gets lower priority %d\n",retnice);
        sleep(1);
        }
}
}
```

q:2)typeline

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<dirent.h>
#include<fcntl.h>
void typeline(char *s , char *fn)
{
int handle,i=0,cnt=0,n;
char ch;
if((handle=open(fn,O_RDONLY))==-1)
{
printf("File %s not found\n",fn);
return;
}
if(strcmp(s, "a")==0)
{
while(read(handle,&ch,1)!=0)
printf("%c",ch);
close(handle);
return;
}
n=atoi(s);
if(n>0)
```

```
{
while(read(handle,&ch,1)!=0)
{
if(ch=='\n')
i++;
if(i==n)
break;
printf("%c",ch);
}
printf("\n");
close(handle);
return;
}
if(n<0)
{
while(read(handle,&ch,1)!=0)
{
if(ch=='\n')
cnt++;
}
lseek(handle,0,SEEK_SET);
while(read(handle,&ch,1)!=0)
{
if(ch=='\n')
i++;
if(i==cnt+n-1)
break;
}
while(read(handle,&ch,1)!=0)
printf("%c",ch);
printf("\n");
close(handle);
}
}
 int main()
{
char command[80],t1[20],t2[20],t3[20],t4[20];
```

```
int n;
system("clear");
while(1)
{
printf("myShell$");
fflush(stdin);
fgets(command,80,stdin);
n=sscanf(command,"%s %s %s %s",t1,t2,t3,t4);
switch(n)
{
case 1:
if(!fork())
{
execlp(t1,t1,NULL);
perror(t1);
}
break;
case 2:
if(!fork())
{
execlp(t1,t1,t2,NULL);
perror(t1);
}
break;
case 3:
    if(strcmp(t1,"typeline")==0)
    typeline(t2[0],t3);
     else
       {
        if(!fork())
          {
       execlp(t1,t1,t2,t3,NULL);
      perror(t1);
}
}
break;
case 4:
```

```
if(!fork())
{
execlp(t1,t1,t2,t3,t4,NULL);
perror(t1);
}
}
}
}
```

Slip 09

q:1)list

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<dirent.h>
#include<fcntl.h>
void list (char c,char *dn)
{
DIR *dir;
int cnt=0;
struct dirent *entry;
struct stat buff;
if((dir=opendir(dn))==NULL)
{
printf("Directory %s not found\n",dn);
return;
}
switch(c)
{
case 'f':
while((entry=readdir(dir))!=NULL)
{
stat (entry->d_name,&buff);
if(S_IFREG&buff.st_mode)
printf("%s\n",entry->d_name);
}
```

```
break;
case 'n':
while((entry=readdir(dir))!=NULL)
cnt++;
printf("total no of entries=%d\n",cnt);
break;
case 'i':
while((entry=readdir(dir))!=NULL)
{
stat(entry->d_name,&buff);
if(S_IFREG&buff.st_mode)
printf("%s\t%d\n",entry->d_name,buff.st_ino);
}
break;
default:
printf("Invalid argument.....\n");
}
closedir(dir);
}
main()
{
char command[80],t1[20],t2[20],t3[20],t4[20];
int n;
system("clear");
while(1)
{
printf("myShell$");
fflush(stdin);
fgets(command,80,stdin);
n=sscanf(command,"%s %s %s",t1,t2,t3,t4);
switch(n)
{
case 1:
if(!fork())
{
execlp(t1,t1,NULL);
perror(t1);
```

```
}
break;
case 2:
if(!fork())
{
execlp(t1,t1,t2,NULL);
perror(t1);
}
break;
case 3:
if(strcmp(t1,"list")==0)
list(t2[0],t3);
else
{
if(!fork())
{
execlp(t1,t1,t2,t3,NULL);
perror(t1);
}
}
break;
case 4:
if(!fork())
{
execlp(t1,t1,t2,t3,t4,NULL);
perror(t1);
}
}
}
}
```

q:2)fifo

```
#include<stdio.h>
#define MAX 20

int frame[MAX],ref[MAX],mem[MAX][MAX],faults,sp,m,n,time[MAX];
void accept()
```

```
{
        int i;
        printf("enter the no.of frame");
        scanf("%d",&n);
        printf("enter the reference:");
        scanf("%d",&m);
        printf("enter reference string:");
        for(i=0;i<m;i++)
        {
                printf("[%d]=",i);
                scanf("%d",&ref[i]);
                }
}
void disp()
{
        int i,j;

        for(i=0;i<m;i++)
        printf("%3d",ref[i]);

        printf("\n\n");

        for(i=0;i<n;i++)
        {
        for(j=0;j<m;j++)
        {
                if(mem[i][j])
                        printf("%3d",mem[i][j]);
                else
                        printf(" ");
        }
                printf("\n");
        }
        printf("total page faults:%d\n",faults);

}
int search(int pno)
```

```
{
        int i;
        for(i=0;i<n;i++)
        {
                if(frame[i]==pno)
                return i;
        }
        return-1;
}
void fifo()
{
        int i,j;
        for(i=0;i<m;i++)
        {
                if(search(ref[i])==-1)
                {
                        frame[sp]=ref[i];
                        sp=(sp+1)%n;
                        faults++;
                        for(j=0;j<n;j++)
                        mem[j][i]=frame[j];
                }
        }
}


int main()
{
        accept();
        fifo();
        disp();

        return 0;
}
```

Slip 10

q:1)provess

```
#include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        pid_t pid=fork();
        if(pid==0)
        {
                printf("\nI am child process");
                printf("\n Process Id = %d",getpid());
                 printf("\n priority : %d = %d\n",nice(-7),getpid());
        }
        else if(pid>0)
        {
                printf("\nI am parent process");
                printf("\n process id=%d",getpid());
                printf("\n process id = %d",getpid());
                printf("\n prioritty:%d=%d\n",nice(15),getpid());
        }
        else
        {
                printf("fork() system call fails");
        }
        return 0;
 }
```

q:2)sjf

```
 #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 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+=p->bt1;
          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++;
     }

     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;
    }
}
int main()
{
        accept_info();
        sort();
        sjfp();
        print_output();
        print_gantt_chart();
        return 0;
}
```

SLIP 11

Q:1)LIST

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<dirent.h>
#include<fcntl.h>
void list (char c,char *dn)
{
DIR *dir;
int cnt=0;
struct dirent *entry;
struct stat buff;
if((dir=opendir(dn))==NULL)
{
printf("Directory %s not found\n",dn);
return;
}
switch(c)
{
```

```
case 'f':
while((entry=readdir(dir))!=NULL)
{
stat (entry->d_name,&buff);
if(S_IFREG&buff.st_mode)
printf("%s\n",entry->d_name);
}
break;
case 'n':
while((entry=readdir(dir))!=NULL)
cnt++;
printf("total no of entries=%d\n",cnt);
break;
case 'i':
while((entry=readdir(dir))!=NULL)
{
stat(entry->d_name,&buff);
if(S_IFREG&buff.st_mode)
printf("%s\t%d\n",entry->d_name,buff.st_ino);
}
break;
default:
printf("Invalid argument.....\n");
}
closedir(dir);
}
main()
{
char command[80],t1[20],t2[20],t3[20],t4[20];
int n;
system("clear");
while(1)
{
printf("myShell$");
fflush(stdin);
fgets(command,80,stdin);
n=sscanf(command,"%s %s %s",t1,t2,t3,t4);
```

```
switch(n)
{
case 1:
if(!fork())
{
execlp(t1,t1,NULL);
perror(t1);
}
break;
case 2:
if(!fork())
{
execlp(t1,t1,t2,NULL);
perror(t1);
}
break;
case 3:
if(strcmp(t1,"list")==0)
list(t2[0],t3);
else
{
if(!fork())
{
execlp(t1,t1,t2,t3,NULL);
perror(t1);
}
}
break;
case 4:
if(!fork())
{
execlp(t1,t1,t2,t3,t4,NULL);
perror(t1);
}
}
}
}
```

Q;2)INSERTION BUBBLE SORT

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>

int main()
{
   int pid, n,num[20], i, j, temp ,key;
   printf("enter length of array: ");
   scanf("%d",&n);

   printf("Enter elements of array :");
   for(i=0;i<n;i++)
    scanf("%d",&num[i]);

   pid=fork();
    if(pid==0)
    {
       printf("Child process will execute by Bubble sort\n");
       for(i=0;i<n;i++)
       {
          for(j=0;j<n;j++)
          {
             if(num[j]>num[j+1])
             {
                temp=num[j];
                num[j]=num[j+1];
                num[j+1]=temp;
             }
          }
       }

       printf("The sorted array is: ");
       for(i=0;i<n;i++)
       {
```

```
        printf("%d",num[i]);
      }
      printf("\n");
   }

   else
   {
      printf("Parent process will execute by insretation sort\n");
      for(i=0;i<n;i++)
      {
         key=num[i];
         j=i-1;
         while(j>0 && num[j]>key)
         {
            num[j+1]=num[j];
            j=j-1;
         }
         num[j+1]=key;
      }

       printf("The sorted array is: ");
      for(i=0;i<n;i++)
      {
         printf("%d",num[i]);
      }
      printf("\n");
   }
}
```

SLIP 12

Q:1)COUNT

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<dirent.h>
#include<fcntl.h>
```

```
#include<string.h>
void count(char c,char *fn)
{
        int lc=0,wc=0,cc=0,handle;
        char ch;
        if((handle=open(fn,O_RDONLY))==-1)
        {
                printf("file is not found\n",fn);
                return;
        }
        while(read(handle,&ch,1)!=0)
        {
                if(ch==' ')
                        wc++;
                else if(ch=='\n')
                        lc++;
                else
                        cc++;
        }
        close(handle);
        switch(c);
        {
           case 'c':
       printf("Total no. of characters=%d\n",cc);
       break;

      case 'c':
       printf("Total no. of words=%d\n",wc);
       break;

      case 'c':
       printf("Total no. of lines=%d\n",lc);
       break;
         }

}
int main()
```

```
{
        char command[80],t1[20],t2[20],t3[20],t4[20];
        int n;
        system("clear");
        while(1)
        {
        printf("myshells");
        fflush(stdin);
        fgets(command,80,stdin);
        n=sscanf(command,"%s %s %s %s",t1,t2,t3,t4);
        switch(n)
        {
        case 1:
                if(!fork())
                {
                        execlp(t1,t1,NULL);
                        perror(t1);
                }
                break;
        case 2:
                if(!fork())
                {
                        execlp(t1,t1,t2,NULL);
                        perror(t1);
                }
                break;
        case 3:
                if(strcmp(t1,"count")==0)
                        count(t2[0],t3);
                else
                {

                if(!fork())
                {
                        execlp(t1,t1,t2,t3,NULL);
                        perror(t1);
                }
```

```
                }
                break;
        case 4:
                if(!fork())
                {
                        execlp(t1,t1,t2,t3,t4,NULL);
                        perror(t1);
                }
                break;
        }
}
}
```

Q:2)FCFS

```
#include<stdio.h>
#include<string.h>
struct Process
{
 char PName[5];
 int AT,BT,TAT,WT,CT;
 int tempBT;
 }P[10];
 int N;
void Input()
{
   int i;
   printf("\nenter number of process:");
   scanf("%d",&N);
   for(i = 0;i < N;i++)
   {
      printf("\nenter details of process %d",i+1);
      printf("\nenter process name:");
      scanf("%s",P[i].PName);

      printf("\nenter AT:");
      scanf("%d",&P[i].AT);
```

```
        printf("\nenter BT:");
        scanf("%d",&P[i].BT);

        P[i].tempBT = P[i].BT;
    }
}
void SortProcessAT()
{
        int i;
        int j;
        int tva1;
        char temp[5];

        for(i = 0; i < N; i++)
            for(j = 0; j < N; j++)
            {
              if(P[i].AT<P[j].AT)
              {
               //swap pname
                    strcpy(temp,P[i].PName);
                    strcpy(P[i].PName,P[j].PName);
                    strcpy(P[j].PName,temp);
              //swap at
                    tva1 = P[i].AT;
                    P[i].AT = P[j].AT;
                    P[j].AT = tva1;
              //swap bt
                    tva1 = P[i].BT;
                    P[i].BT = P[j].BT;
                    P[j].BT = tva1;
              //swap tempbt
                     tva1 = P[i].tempBT;
                    P[i].tempBT = P[j].tempBT;
                    P[j].tempBT = tva1;
              }
            }
}
```

```
void Output()
{
	int i;
	int totalTAT = 0;
	int totalWT = 0;
	for(i = 0; i < N; i++)
	{
	   P[i].TAT = P[i].CT - P[i].AT;
	   totalTAT = totalTAT + P[i].TAT;

	   P[i].WT = P[i].TAT - P[i].BT;
	   totalWT = totalWT + P[i].WT;
	}

	printf("\nProcess Details");
	printf("\n*****");
	printf("\nPName\tAT\tBT\tTAT\tWT");
	printf("\n*****");
	for(i = 0; i < N; i++)
	{
printf("\n%s\t%d\t%d\t%d\t%d",P[i].PName,P[i].AT,P[i].BT,P[i].TAT,P[i].WT);
	}
	printf("\n*****\n");
	printf("\nAVG TAT:%d / %d",totalTAT,N);
	printf("\nAVG WT:%d / %d",totalWT,N);
}
void FCFS()
{
	 int time = 0;
	 int finish = 0;
	 int i;
	 int flag = 0;
	 while(finish !=1)
	 {
	    flag = 0;
	    for(i = 0; i < N; i++)
	    {
```

```
            if(P[i].AT <= time && P[i].tempBT != 0)
            {
               printf("|%d %s",time,P[i].PName);
               time = time + P[i].tempBT;
               printf("%d|",time);
               P[i].tempBT = 0;
               P[i].CT = time;
               flag = i;
            }
         }
         if(flag ==0)
         {
            printf("|%d # ",time);
            time++;
            printf("%d|",time);
         }
         for(i = 0; i < N; i++)
            if(P[i].tempBT == 0)
               continue;
            else
               break;
         if(i == N)
            finish = 1;
      }
}
int main()
{
      Input();
      SortProcessAT();
      FCFS();
      Output();

      return 0;
 }
```

SLIP 13

Q:1)PROCESS

```
 #include<stdio.h>
#include<stdlib.h>
#include<unistd.h>
int main()
{
        pid_t pid=fork();
        if(pid==0)
        {
                printf("\nI am child process");
                printf("\n Process Id = %d",getpid());
                 printf("\n priority : %d = %d\n",nice(-7),getpid());
        }
        else if(pid>0)
        {
                printf("\nI am parent process");
                printf("\n process id=%d",getpid());
                printf("\n process id = %d",getpid());
                printf("\n prioritty:%d=%d\n",nice(15),getpid());
        }
        else
        {
                printf("fork() system call fails");
        }
        return 0;
 }
```

Q:2)SEARCH

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<dirent.h>
#include<fcntl.h>
#include<string.h>
void search(char c, char *s,char *fn)
{
        int handle,i=1,cnt=0,j=0;
```

```
char ch,buff[80],*p;
if((handle=open(fn,O_RDONLY))==-1)
{
        printf("file is not found\n",fn);
        return;
}
switch(c)
{
case'f':
        while(read(handle,&ch,1)!=0)
        {
                if(ch=='\n')
                {
                        buff[j]='\0';
                        j=0;
                        if(strstr(buff,s)!=NULL)
                        {
                                printf("%d : %s\n",i,buff);
                                break;
                        }
                        i++;
                }
                else
                        buff[j++]=ch;
        }
        break;
case 'c':
    while(read(handle,&ch,1)!=0)
    {
            if(ch=='\n')
            {
                buff[j]='\0';
                j=0;
                if(strstr(buff,s)!=NULL)
                {
                        p=buff;
    while((p=strstr(p,s))!=NULL)
```

```
                        {
                            cnt++;
                            p++;
                        }

                    }
                }
                else
                    buff[j++]=ch;
            }
            printf("Total no .of Occurrence = %d\n",cnt);
            break;
    case 'a' :
            while(read(handle,&ch,1)!=0)
            {
                if(ch=='\n')
                {
                        buff[j]='\0';
                        j=j;
                        if(strstr(buff,s)!=NULL)
                        printf("%d:%s\n",i,buff);
                        i++;
                }
                else
                        buff[j++]=ch;
            }
    }
    close(handle);
}

main()
{
        char command[80],t1[20],t2[20],t3[20],t4[20];
        int n;
        system("clear");
        while(1)
        {
```

```
            printf("myShell$");
            fflush(stdin);
            fgets(command,80,stdin);
            n=sscanf(command,"%s %s %s %s",t1,t2,t3,t4);
            switch(n)
            {
            case1:
                        if(!fork())
                        {
                              execlp(t1,t1,NULL);
                              perror(t1);
                        }
                        break;
        case 2:
                        if(!fork())
                        {
                              execlp(t1,t1,t2,NULL);
                              perror(t1);
                        }
                        break;
        case 3:
                    if(!fork())
                {
                        execlp(t1,t1,t2,t3,NULL);
                        perror(t1);
                        }
                        break;
            case 4:
                         if(!fork())
                        {
                              execlp(t1,t1,t2,t3,t4,NULL);
                              perror(t1);
                        }
                }
      }
}
```

SLIP 14

Q:1)PROCESS

```
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{

        int pid=fork();
        if(pid>0)
        {
        printf("parent process\n");
        printf("ID:%d\n\n",getpid());
        }
        else if(pid==0)
        {
        printf("child process\n");
        printf("ID:%d\n",getpid());
        sleep(10);
        printf("\n child process\n");
        printf("ID:%d\n",getpid());
        printf("parent terminated then p-ID:%d\n",getpid());
        }
        else
        {
        printf("failed to create child process");
        }
                waitpid(pid,NULL,0);
                return 0;
}
```

Q;2)TYPELINE

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
```

```
#include<dirent.h>
#include<fcntl.h>
void typeline(char *s , char *fn)
{
int handle,i=0,cnt=0,n;
char ch;
if((handle=open(fn,O_RDONLY))==-1)
{
printf("File %s not found\n",fn);
return;
}
if(strcmp(s, "a")==0)
{
while(read(handle,&ch,1)!=0)
printf("%c",ch);
close(handle);
return;
}
n=atoi(s);
if(n>0)
{
while(read(handle,&ch,1)!=0)
{
if(ch=='\n')
i++;
if(i==n)
break;
printf("%c",ch);
}
printf("\n");
close(handle);
return;
}
if(n<0)
{
while(read(handle,&ch,1)!=0)
{
```

```
if(ch=='\n')
cnt++;
}
lseek(handle,0,SEEK_SET);
while(read(handle,&ch,1)!=0)
{
if(ch=='\n')
i++;
if(i==cnt+n-1)
break;
}
while(read(handle,&ch,1)!=0)
printf("%c",ch);
printf("\n");
close(handle);
}
}
 int main()
{
char command[80],t1[20],t2[20],t3[20],t4[20];
int n;
system("clear");
while(1)
{
printf("myShell$");
fflush(stdin);
fgets(command,80,stdin);
n=sscanf(command,"%s %s %s %s",t1,t2,t3,t4);
switch(n)
{
case 1:
if(!fork())
{
execlp(t1,t1,NULL);
perror(t1);
}
break;
```

```
case 2:
if(!fork())
{
execlp(t1,t1,t2,NULL);
perror(t1);
}
break;
case 3:
    if(strcmp(t1,"typeline")==0)
    typeline(t2[0],t3);
     else
       {
        if(!fork())
          {
       execlp(t1,t1,t2,t3,NULL);
      perror(t1);
}
}
break;
case 4:
if(!fork())
{
execlp(t1,t1,t2,t3,t4,NULL);
perror(t1);
}
}
}
}
```

SLIP 15

Q:1)PROCESS

```
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{

         int pid=fork();
```

```
        if(pid>0)
        {
        printf("parent process\n");
        printf("ID:%d\n\n",getpid());
        }
        else if(pid==0)
        {
        printf("child process\n");
        printf("ID:%d\n",getpid());
        sleep(10);
        printf("\n child process\n");
        printf("ID:%d\n",getpid());
        printf("parent terminated then p-ID:%d\n",getpid());
        }
        else
        {
        printf("failed to create child process");
        }
                waitpid(pid,NULL,0);
                return 0;
}

Q:2)FCFS
#include<stdio.h>
#include<string.h>
struct Process
{
 char PName[5];
 int AT,BT,TAT,WT,CT;
 int tempBT;
 }P[10];
 int N;
void Input()
{
   int i;
   printf("\nenter number of process:");
   scanf("%d",&N);
```

```
   for(i = 0;i < N;i++)
   {
      printf("\nenter details of process %d",i+1);
      printf("\nenter process name:");
      scanf("%s",P[i].PName);

      printf("\nenter AT:");
      scanf("%d",&P[i].AT);

      printf("\nenter BT:");
      scanf("%d",&P[i].BT);

      P[i].tempBT = P[i].BT;
    }
}
void SortProcessAT()
{
      int i;
      int j;
      int tva1;
      char temp[5];

      for(i = 0; i < N; i++)
          for(j = 0; j < N; j++)
          {
            if(P[i].AT<P[j].AT)
            {
             //swap pname
                 strcpy(temp,P[i].PName);
                 strcpy(P[i].PName,P[j].PName);
                 strcpy(P[j].PName,temp);
            //swap at
                 tva1 = P[i].AT;
                 P[i].AT = P[j].AT;
                 P[j].AT = tva1;
            //swap bt
                 tva1 = P[i].BT;
```

```
                P[i].BT = P[j].BT;
                P[j].BT = tva1;
            //swap tempbt
                 tva1 = P[i].tempBT;
                P[i].tempBT = P[j].tempBT;
                P[j].tempBT = tva1;
             }
           }
}
void Output()
{
      int i;
      int totalTAT = 0;
      int totalWT = 0;
      for(i = 0; i < N; i++)
      {
         P[i].TAT = P[i].CT - P[i].AT;
         totalTAT = totalTAT + P[i].TAT;

         P[i].WT = P[i].TAT - P[i].BT;
         totalWT = totalWT + P[i].WT;
      }

      printf("\nProcess Details");
      printf("\n*****");
      printf("\nPName\tAT\tBT\tTAT\tWT");
      printf("\n*****");
      for(i = 0; i < N; i++)
      {
printf("\n%s\t%d\t%d\t%d\t%d",P[i].PName,P[i].AT,P[i].BT,P[i].TAT,P[i].WT);
      }
      printf("\n*****\n");
      printf("\nAVG TAT:%d / %d",totalTAT,N);
      printf("\nAVG WT:%d / %d",totalWT,N);
}
void FCFS()
{
```

```
        int time = 0;
        int finish = 0;
        int i;
        int flag = 0;
        while(finish !=1)
        {
           flag = 0;
           for(i = 0; i < N; i++)
           {
                if(P[i].AT <= time && P[i].tempBT != 0)
                {
                   printf("|%d %s",time,P[i].PName);
                   time = time + P[i].tempBT;
                   printf("%d|",time);
                   P[i].tempBT = 0;
                   P[i].CT = time;
                   flag = i;
                }
           }
           if(flag ==0)
           {
              printf("|%d # ",time);
              time++;
              printf("%d|",time);
           }
           for(i = 0; i < N; i++)
                if(P[i].tempBT == 0)
                     continue;
                else
                     break;
           if(i == N)
                finish = 1;
        }
}
int main()
{
        Input();
```

```
    SortProcessAT();
    FCFS();
    Output();

    return 0;
}
```