/\* fork system call \*/#include<stdio.h>

#include <unistd.h>

#include<sys/types.h>

int main()

{

int id,childid;

id=getpid();

if((childid=fork())>0)

{

printf("\n i am in the parent process %d",id);

printf("\n i am in the parent process %d",getpid());

printf("\n i am in the parent process %d\n",getppid());

}

else

{

printf("\n i am in child process %d",id);

printf("\n i am in the child process %d",getpid());

printf("\n i am in the child process %d",getppid());

}

}

------------------------------------------------------

/\* waitexit system call \*/

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/wait.h>

int main()

{

int i, pid;

pid=fork( );if(pid== -1)

{

printf("fork failed");

exit(0);

}

else if(pid==0)

{

printf("\n Child process starts");

for(i=0; i<5; i++)

{

printf("\n Child process %d is called", i);

}

printf("\n Child process ends");

}

else

{

wait(0);

printf("\n Parent process ends");

}

exit(0);

}

------------------------------------------------------

/\* execl system call \*/

#include<sys/types.h>

#include<unistd.h>

#include<stdio.h>

int main()

{

printf("Before execl \n");

execl("/bin/ls","ls",(char\*)0);

printf("After Execl\n");

}

-----------------------------------------------------

/\* execv system call \*/

#include<stdio.h>

#include<unistd.h>

#include<sys/types.h>

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

{

printf("before execv\n");

execv("/bin/ls",argv);

printf("after execv\n");

}

-----------------------------------------------------

/\* opendir closedir readdir \*/

#include<stdio.h>

#include<dirent.h>

#include<errno.h>

#include<fcntl.h>

#include<unistd.h>

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

{

struct dirent \*direntp; DIR \*dirp; if(argc!=2)

{

printf("ussage %s directory name \n",argv[0]);

return 1;

}

if((dirp=opendir(argv[1]))==NULL)

{

perror("Failed to open directory \n");

return 1;

}

while((direntp=readdir(dirp))!=NULL)

printf("%s\n",direntp->d\_name);

while((closedir(dirp)==-1)&&(errno==EINTR));

return 0;

}

---------------------------------------------------

/\* open close read write\*/

#include<stdio.h>

#include<unistd.h>

#include<string.h>

#include<fcntl.h>

int main( )

{

int fd[2];

char buf1[25]= "just a test\n"; char

buf2[50];

fd[0]=open("file1", O\_RDWR);

fd[1]=open("file2", O\_RDWR);

write(fd[0], buf1, strlen(buf1));

printf("\n Enter the text now…");

gets(buf1);

write(fd[0], buf1, strlen(buf1));

lseek(fd[0], SEEK\_SET, 0);

read(fd[0], buf2, sizeof(buf1));

write(fd[1], buf2, sizeof(buf2));

close(fd[0]);

close(fd[1]);

printf("\n");

return 0;

}

--------------------------------------------------

/\* fcfs \*/

#include<stdio.h>

int main()

{

char pn[10][10];

int arr[10],bur[10],star[10],finish[10],tat[10],wt[10],i,n;

int totwt=0,tottat=0;

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

scanf("%d",&n);

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

{

printf("Enter the Process Name, Arrival Time & Burst Time:");

scanf("%s%d%d",pn[i],&arr[i],&bur[i]);

}

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

{

if(i==0)

{

star[i]=arr[i];

wt[i]=star[i]-arr[i];

finish[i]=star[i]+bur[i];

tat[i]=finish[i]-arr[i];

}

else

{

star[i]=finish[i-1];

wt[i]=star[i]-arr[i];

finish[i]=star[i]+bur[i];

tat[i]=finish[i]-arr[i];

}

}

printf("\nPName Arrtime Burtime Start TAT Finish");

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

{

printf("\n%s\t%6d\t\t%6d\t%6d\t%6d\t%6d",pn[i],arr[i],bur[i],star[i],tat[i],finish[i]);

totwt+=wt[i];

tottat+=tat[i];

}

printf("\nAverage Waiting time:%f", (float)totwt);

printf("\nAverage Turn Around Time:%f", (float)tottat);

}

-------------------------------------------------------------------------------------------

/\* A program to simulate the SJF CPU scheduling algorithm \*/

#include<stdio.h>

#include<string.h>

int main()

{

int i=0,pno[10],bt[10],n,wt[10],temp=0,j,tt[10];

float sum,at;

printf("\n Enter the no of process ");

scanf("\n %d",&n);

printf("\n Enter the burst time of each process");

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

{

printf("\n p%d",i);

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

}

for(i=0;i<n-1;i++)

{

for(j=i+1;j<n;j++)

{

if(bt[i]>bt[j])

{

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

temp=pno[i];

pno[i]=pno[j];

pno[j]=temp;

}

}

}

wt[0]=0;

for(i=1;i<n;i++)

{

wt[i]=bt[i-1]+wt[i-1];

sum=sum+wt[i];

}

printf("\n process no \t burst time\t waiting time \t turn around time\n");

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

{

tt[i]=bt[i]+wt[i];

at+=tt[i];

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

}

printf("\n\n\t Average waiting time%f\n\t Average turn around time%f", sum, at);

}

---------------------------------------------------------------------------------

/\* Round Robin \*/

#include<stdio.h>

struct process

{

int burst,wait,comp,f;

}p[20]={0,0};

int main()

{

int n,i,j,totalwait=0,totalturn=0,quantum,flag=1,time=0;

printf("\nEnter The No Of Process :");

scanf("%d",&n);

printf("\nEnter The Quantum time (in ms) :");

scanf("%d",&quantum);

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

{

printf("Enter The Burst Time (in ms) For Process #%2d :",i+1);

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

p[i].f=1;

}

printf("\nOrder Of Execution \n");

printf("\nProcess Starting Ending Remaining");

printf("\n\t\tTime \tTime \t Time");

while(flag==1)

{

flag=0;

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

{

if(p[i].f==1)

{

flag=1;

j=quantum;

if((p[i].burst-p[i].comp)>quantum)

{

p[i].comp+=quantum;

}

else

{

p[i].wait=time-p[i].comp;

j=p[i].burst-p[i].comp;

p[i].comp=p[i].burst;

p[i].f=0;

}

printf("\nprocess # %-3d %-10d %-10d %-10d", i+1, time, time+j, p[i].burst-p[i].comp);

time+=j;

}

}

}

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

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

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

{

printf("\nProcess # %-12d%-15d%-15d",i+1,p[i].wait,p[i].wait+p[i].burst);

totalwait=totalwait+p[i].wait;

totalturn=totalturn+p[i].wait+p[i].burst;

}

printf("\n\nAverage\n------------------ ");

printf("\nWaiting Time: %fms",totalwait/(float)n);

printf("\nTurnAround Time : %fms\n\n",totalturn/(float)n);

return 0;

}

-----------------------------------------------------------------------------------------

/\* echoserver using Pipes \*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

#define msgsize 29

int main()

{

int ser[2],cli[2],pid;

char inbuff[msgsize];

char \*msg="Thank you";

system("clear");

pipe(ser);

pipe(cli);

printf("\n server read id =%d,write id=%d",ser[0],ser[1]);

printf("\n client read id =%d,write id=%d",cli[0],cli[1]);

pid=fork();

if(pid==0)

{

printf("\n i am in child process !");

close(cli[0]);

close(ser[1]);

write(cli[1],msg,msgsize);

printf("\n message written to pipe..!");

sleep(2);

read(ser[0],inbuff,msgsize);

printf("\n echo message received from server");

printf("\n %s",inbuff);

}

else

{

close(cli[1]);

close(ser[0]);

printf("\n parent process");

read(cli[0],inbuff,msgsize);

write(ser[1],inbuff,msgsize);

printf("\n parent ended!");

}

}

--------------------------------------------------------------------------------

/\* echo server using messages \*/

#include<sys/ipc.h>

#include<stdio.h>

#include<string.h>

#include<sys/msg.h>

#include<stdlib.h>

#include<unistd.h>

struct

{

long mtype;

char mtext[20];

}send,recv;

int main()

{

int qid,pid,len;

qid=msgget((key\_t)0X2000,IPC\_CREAT|0666);

if(qid==-1)

{

perror("\n message failed");

exit(1);

}

send.mtype=1;

strcpy(send.mtext,"\n hello i am parent");

len=strlen(send.mtext);

pid=fork();

if(pid>0)

{

if(msgsnd(qid,&send,len,0)==-1)

{

perror("\n message sending failed");

exit(1);

}

printf("\n message has been posted");

sleep(2);

if(msgrcv(qid,&recv,100,2,0)==-1)

{

perror("\n msgrcv error:");

exit(1);

}

printf("\n message received from child - %s\n",recv.mtext);

}

else

{

send.mtype=2;

strcpy(send.mtext,"\n hi i am child"); len=strlen(send.mtext);

if(msgrcv(qid,&recv,100,1,0)==-1)

{

perror("\n child message received failed");

exit(1);

}

if(msgsnd(qid,&send,len,0)==-1)

{

perror("\n child message send failed");

}

printf("\n received from parent - %s",recv.mtext);

}

}

---------------------------------------------------------------------------

/\* producerconsumer \*/

#include<stdio.h>

void main()

{

int buffer[10], bufsize, in, out, produce, consume, choice=0;

in = 0;

out = 0;

bufsize = 10;

while (choice !=3)

{

printf("\n1. Produce \t 2. Consume \t3. Exit");

printf("\nEnter your choice: ");

scanf("%d", &choice);

switch(choice)

{

case 1: if((in+1)%bufsize==out)

printf("\nBuffer is Full");

else

{

printf("\nEnter the value: ");

scanf("%d", &produce);

buffer[in] = produce;

in = (in+1)%bufsize;

}

break;

case 2: if(in == out)

printf("\nBuffer is Empty");

else

{

consume = buffer[out];

printf("\nThe consumed value is %d", consume);

out = (out+1)%bufsize;

}

break;

}

}

}

-----------------------------------------------------------------------

sender reciever

========================================================================

/\*mesh.h\*/

#include<sys/types.h>

#include<sys/ipc.h>

#include<sys/msg.h>

#include<stdio.h>

#include<stdlib.h>

#define MKEY1 5543L

#define MKEY2 4354L

#define PERMS 0666

typedef struct

{

long mtype;

char mdata[50];

}mesg;

------------------------------------------------------------------------

/\*sender\*/

#include "mesg.h"

#include<unistd.h>

mesg msg;

int main()

{

int mq\_id;

int n;

if((mq\_id=msgget(MKEY1,PERMS|IPC\_CREAT))<0)

{

printf("Sender: Error creating message");

exit(1);

}

msg.mtype=1111L;

n=read(0,msg.mdata,50);

msg.mdata[n]='\0';

msgsnd(mq\_id,&msg,50,0);

}

------------------------------------------------------------------------

/\*reciver\*/

#include "mesg.h"

#include<unistd.h>

mesg msg;

int main()

{

int mq\_id;

int n;

if( ( mq\_id=msgget(MKEY1, PERMS|IPC\_CREAT ) ) < 0)

{

printf("receiver: Error opening message");

exit(1);

}

msgrcv(mq\_id,&msg,50,1111L,0);

write(1,msg.mdata,50);

msgctl(mq\_id,IPC\_RMID,NULL);

}

=========================================================================

---------------------------------------------------------------------------

/\* FIFO \*/

#include<stdio.h>

int main()

{

int a[5],b[20],n,p=0,q=0,m=0,h,k,i,q1=1;

char f='F';

printf("Enter the Number of Pages:");

scanf("%d",&n);

printf("Enter %d Page Numbers:",n);

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

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

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

{

if(p==0)

{

if(q>=3)

q=0;

a[q]=b[i];

q++;

if(q1<3)

{

q1=q;

}

}

printf("\n%d",b[i]);

printf("\t");

for(h=0;h<q1;h++)

printf("%d",a[h]);

if((p==0)&&(q<=3))

{

printf("-->%c",f);

m++;

}

p=0;

for(k=0;k<q1;k++)

{

if(b[i+1]==a[k])

p=1;

}

}

printf("\nNo of faults:%d",m);

}

----------------------------------------------------------------------------

/\* LRU \*/

#include<stdio.h>

int main()

{

int a[5],b[20],p=0,q=0,m=0,h,k,i,q1=1,j,u,n;

char f='F';

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

scanf("%d",&n);

printf("Enter %d Page Numbers:",n);

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

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

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

{

if(p==0)

{

if(q>=3)

q=0;

a[q]=b[i];

q++;

if(q1<3)

{

q1=q;

}

}

printf("\n%d",b[i]);

printf("\t");

for(h=0;h<q1;h++)

printf("%d",a[h]);

if((p==0)&&(q<=3))

{

printf("-->%c",f);

m++;

}

p=0;

if(q1==3)

{

for(k=0;k<q1;k++)

{

if(b[i+1]==a[k])

p=1;

}

for(j=0;j<q1;j++)

{

u=0;

k=i;

while(k>=(i-1)&&(k>=0))

{

if(b[k]==a[j])

u++;

k--;

}

if(u==0)

q=j;

}

}

else

{

for(k=0;k<q;k++)

{

if(b[i+1]==a[k])

p=1;

}

}

}

printf("\nNo of faults:%d",m);

}

--------------------------------------------------------------------------