//list

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

#include <dirent.h>

char comm[100],\*ptr;

char \*args[10];

int tot;

int list( char \*option, char \*dirname);

main()

{

do

{

printf("myshell$ ");

getcomm();

sep\_arg();

take\_action();

}while(1);

}

sep\_arg()

{

int i,j;

char \*token;

i = j=tot=0;

token=strtok(comm," ");

while(token!=NULL)

{

args[tot]=(char\*)malloc(20);

strcpy(args[tot++],token);

token=strtok(NULL," ");

}

return;

}

getcomm()

{

int len;

ptr = fgets(comm,80,stdin);

len = strlen(comm);

comm[len-1] = '\0';

return;

}

take\_action()

{

char str[100];

pid\_t pid;

int status;

if ( strlen(comm) <= 1 )

return;

if(strcmp(args[0],"list") == 0 )

list(args[1],args[2]);

else

{ pid = fork();

if( pid == 0 )

{

execvp(args[0],args);

strcpy(str,"./");

strcat(str,args[0]);

execvp(str,args);

printf("\n%s : command not found",comm);

exit(0);

}

else

waitpid(pid,&status,0);

}

}

int list( char \*option, char \*dirname)

{

DIR \*dp;

struct dirent \*dent;

int cnt;

if(strcmp(option,"f")==0)

{

dp= opendir(dirname);

if( dp == NULL )

{

printf("\nUnable to open directory\n");

return;

}

dent = readdir(dp);

while( dent != NULL )

{

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

dent= readdir(dp);

}

printf("\n");

}

else

if(strcmp(option,"n")==0)

{

cnt =0;

dp= opendir(dirname);

if( dp == NULL )

{

printf("\nUnable to open dir");

exit(0);

}

dent = readdir(dp);

while( dent != NULL )

{

cnt++;

dent= readdir(dp);

}

printf("\ntot dir entries = %d\n",cnt);

closedir(dp);

}

else

if(strcmp(option,"i")==0)

{ // list all dir entries with inode

dp= opendir(dirname);

if( dp == NULL )

{

printf("\nUnable to open dir");

exit(0);

}

dent = readdir(dp);

while( dent != NULL )

{

printf("\n%s %ld",dent->d\_name,dent->d\_ino);

dent= readdir(dp);

}

printf("\n");

closedir(dp);

}

else

printf("\nInvalid option\n");

return(0);

}

///search

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

char comm[100],\*ptr;

char \*args[10];

int tot\_arg;

int search(char option, char \*pattern, char \*fname);

main()

{

do

{

printf("myshell$ ");

getcomm();

sep\_arg();

take\_action();

}while(1);

}

sep\_arg()

{

int i,j;

i = j = tot\_arg = 0;

args[tot\_arg] = (char\*)malloc(sizeof(char) \* 20);

for(i=0; comm[i] !='\0' ; i++)

{

if( comm[i] == ' ')

{

args[tot\_arg][j] = '\0';

tot\_arg++;

args[tot\_arg] = (char\*)malloc(sizeof(char) \* 20);

j=0;

}

else

{

args[tot\_arg][j] =comm[i];

j++;

}

}

args[tot\_arg][j] = '\0';

return;

}

getcomm()

{

int len;

ptr = fgets(comm,80,stdin);

len = strlen(comm);

comm[len-1] = '\0';

return;

}

take\_action()

{

char str[100];

pid\_t pid;

int status;

ptr=comm;

if( ptr == NULL )

exit(0);

if ( strlen(comm) <= 1 )

return;

if(strcmp(args[0],"search") == 0 )

search(\*args[1],args[2],args[3]);

else

if( ( pid = fork() ) < 0 )

{

printf("\nUnable to create process");

}

else

if( pid == 0 )

{

execvp(args[0],args);

strcpy(str,"./");

strcat(str,args[0]);

execvp(str,args);

printf("\n%s : command not found",comm);

exit(0);

}

else

waitpid(pid,&status,0)

}

int search(char option, char \*pattern, char \*fname)

{

char buff[100],\*ptr;

int cnt;

FILE \*fp;

int i;

fp= fopen(fname,"r");

if( fp == NULL )

{

printf("\nUnable to open file");

exit(0);

}

switch(option)

{

case 'a' : while ( !feof (fp))

{

buff[0] = '\0';

fgets(buff,80,fp);

ptr = strstr(buff,pattern);

if( ptr != NULL )

printf("%s",buff);

}

fclose(fp);

break;

case 'c' : cnt=0;

while ( !feof (fp))

{

buff[0] = '\0';

fgets(buff,80,fp);

ptr = strstr(buff,pattern);

while(ptr != NULL )

{

cnt++;

ptr = ptr + strlen(pattern);

ptr = strstr(ptr,pattern);

}

}

fclose(fp);

printf("\nThe serach string %s occurs %d times",pattern,cnt);

break;

case 'f' : while ( !feof (fp))

{

buff[0] = '\0';

fgets(buff,80,fp);

ptr = strstr(buff,pattern);

if( ptr != NULL )

{

printf("%s",buff);

break;

}

}

fclose(fp);

break;

}

return;

}

///count

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

char comm[100],\*ptr;

char \*args[10];

int tot;

int count( char \*option, char \*fname);

main()

{

do

{

printf("myshell$ ");

getcomm();

sep\_arg();

take\_action();

}while(1);

}

sep\_arg()

{

int i,j;

char \*token;

i = j = tot=0;

token=strtok(comm," ");

while(token!=NULL)

{

args[tot]=(char\*)malloc(20);

strcpy(args[tot++],token);

token=strtok(NULL," ");

}

return;

}

getcomm()

{

int len;

ptr = fgets(comm,80,stdin);

len = strlen(comm);

comm[len-1] = '\0';

return;

}

take\_action()

{

char str[100];

pid\_t pid;

int status;

ptr=comm;

if( ptr == NULL )

exit(0);

if ( strlen(comm) <= 1 )

return;

if(strcmp(args[0],"count") == 0 )

count(args[1],args[2]);

else

if( ( pid = fork() ) < 0 )

{

printf("\nUnable to create process");

}

else

if( pid == 0 )

{

execvp(args[0],args);

strcpy(str,"./");

strcat(str,args[0]);

execvp(str,args);

printf("\n%s : command not found",comm);

exit(0);

}

else

waitpid(pid,&status,0);

}

int count( char \*option, char \*fname)

{

int ccnt,wcnt,lcnt,wflag;

int ch;

FILE \*fp;

int i;

fp= fopen(fname,"r");

if( fp == NULL )

{

printf("\nUnable to open file");

exit(0);

}

ccnt =wcnt = lcnt = 0;

wflag=0;

ch = fgetc(fp);

while ( ch != EOF)

{

ccnt++;

if( ch ==' ' || ch== '\t')

{

if( wflag == 0 )

{

wcnt++;

wflag = 1;

}

}

else

if( ch == '\n')

{

lcnt++;

if( wflag == 0 )

{

wcnt++;

wflag = 1;

}

}

else

{

wflag = 0;

}

ch = fgetc(fp);

}//while

if( wflag == 0 )

{

wcnt++;

lcnt++;

}

fclose(fp);

if(strcmp(option,"c")==0)

{ printf("\nTot chars = %d\n",ccnt);

}

else

if(strcmp(option,"w")==0)

{

printf("\nTot words = %d\n",wcnt);

}

else

if(strcmp(option,"l")==0)

{

printf("\nTot lines = %d\n",lcnt);

}

else

printf("\nInvalid option");

}

///typeline

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

char comm[100],\*ptr;

char \*args[10];

int tot;

int typeline(char \*option, char \*fname);

main()

{

do

{

printf("myshell$ ");

getcomm();

sep\_arg();

take\_action();

}while(1);

}

sep\_arg()

{

int i,j;

char \*token;

i = j = tot=0;

token=strtok(comm," ");

while(token!=NULL)

{

args[tot]=(char\*)malloc(20);

strcpy(args[tot++],token);

token=strtok(NULL," ");

}

return;

}

getcomm()

{

int len;

ptr = fgets(comm,80,stdin);

len = strlen(comm);

comm[len-1] = '\0';

return;

}

take\_action()

{

char str[100];

pid\_t pid;

int status;

if ( strlen(comm) <= 1 )

return;

if(strcmp(args[0],"typeline") == 0 )

typeline(args[1],args[2]);

else

{ pid = fork();

if( pid == 0 )

{

execvp(args[0],args);

strcpy(str,"./");

strcat(str,args[0]);

execvp(str,args);

printf("\n%s : command not found\n\n",comm);

exit(0);

}

else

waitpid(pid,&status,0);

}

}

int typeline( char \*option, char \*fname)

{

FILE \*fp;

char str[80];

int tot\_lines,ch,n,cnt;

fp= fopen(fname,"r");

if( fp == NULL )

{

printf("\nUnable to open file");

return;

}

tot\_lines = 0;

while(!feof(fp))

{ fgets(str,80,fp);

tot\_lines++;

}

fclose(fp);

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

{

fp= fopen(fname,"r");

while(!feof(fp))

{ fgets(str,80,fp); if(feof(fp)) break;

printf("%s",str);

}

fclose(fp);

}

else

if( option[0] =='+')

{

n = atoi(option+1);

if( n > tot\_lines)

{

printf("\nInvalid number of lines..");

return;

}

fp= fopen(fname,"r");

cnt = 0;

while(cnt<n)

{ fgets(str,80,fp); if(feof(fp)) break;

printf("%s",str);

cnt++;

}

fclose(fp);

}

else

if( option[0] =='-')

{

n = atoi(option+1);

if( n > tot\_lines)

{

printf("\nInvalid number of lines..");

return;

}

fp= fopen(fname,"r");

cnt = 0;

while( cnt < tot\_lines -n )

{

fgets(str,80,fp); if(feof(fp)) break;

cnt++;

}

while(!feof(fp))

{ fgets(str,80,fp);

printf("%s",str);

}

fclose(fp);

}

else

{

printf("\nInvalid option");

exit(0);

}

return(0);

}

///fifo

#include<stdio.h>

struct frmnode

{

int pno;

}frames[20];

int n;

int page\_found(int pno)

{

int fno;

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

if(frames[fno].pno==pno)

return fno;

return -1;

}

int get\_free\_frame()

{

int fno;

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

if (frames[fno].pno==-1)//free

return(fno);

return(-1);

}

int get\_fifo\_frame()

{

static int fno=-1;

fno=(fno+1)%n;

return(fno);

}

int main()

{

int p\_request[]={1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6};

int size=sizeof(p\_request)/2;

int page\_falts=0,i,j,fno;

printf("\nHow many frames:");

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

{ frames[i].pno=-1;

}

scanf("%d",&n);

printf("\nPageNo Page Frames Page Fault");

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

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

{

j=page\_found(p\_request[i]);

if(j==-1)

{

j=get\_free\_frame();

if (j==-1)

j=get\_fifo\_frame();

page\_falts++;

frames[j].pno=p\_request[i];

printf("\n%4d\t ",p\_request[i]);

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

printf("%4d",frames[fno].pno);

printf(" : YES");

}

else

{

printf("\n%4d\t ",p\_request[i]);

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

printf("%4d",frames[fno].pno);

printf(" : NO");

}

}

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

printf("\n Number of Page\_Falts=%d",page\_falts);

}

///lru

#include<stdio.h>

struct frmnode

{

int pno,counter;

} frames[20];

int n;

int page\_found(int pno)

{

int fno;

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

if(frames[fno].pno==pno)

return fno;

return -1;

}

int get\_free\_frame()

{int fno;

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

if (frames[fno].pno==-1)

return(fno);

return(-1);

}

int get\_lru\_Frame()

{

int lrufno=0,fno;

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

if(frames[fno].counter<frames[lrufno].counter)

lrufno=fno;

return lrufno;

}

int main()

{

int p\_request[]={5,8,10,14,10,9,5,10,8,5,1,10,9,12,10};

int size=sizeof(p\_request)/2,currtime;

int page\_falts=0,i,j,fno;

printf("\nHow many frames:"); scanf("%d",&n);

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

{ frames[i].pno=-1;

}

printf("\nPageNo Page Frames Page Fault");

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

currtime=0;

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

{

j=page\_found(p\_request[i]);

if(j==-1)

{

j=get\_free\_frame();

if (j==-1)

j=get\_lru\_Frame();

page\_falts++;

frames[j].pno=p\_request[i];

frames[j].counter=currtime;

printf("\n%4d\t ",p\_request[i]);

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

printf("%4d:%2d",frames[fno].pno,frames[fno].counter);

printf(" : YES");

}

else

{frames[j].counter=currtime;

printfmfu("\n%4d\t ",p\_request[i]);

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

printf("%4d:%2d",frames[fno].pno,frames[fno].counter);

printf(" : NO");

}

currtime++;

}

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

printf("\n Number of Page\_Falts=%d",page\_falts);

}

//FCFS

#include <stdio.h>

#include <stdlib.h>

struct proc\_info

{ int atime;

int cpub;

} proc[20];

struct rqnode

{

int pid;

struct rqnode \*link;

} \*first=NULL, \*curr, \*prev, \*last;

struct ganttchart

{ int stime;

int pid;

int etime;

} gchart[20];

int i,n,ctime,gi=0,wtime[20],ttime[20];

void getprocs();

void start();

void addprocq();

void attachtoq(int );

void addgchart(int);

void dispgchart();

void disptimes();

int main()

{

getprocs();

ctime=0;

start();

dispgchart();

disptimes();

}

void getprocs()

{ printf("\nHow many processes: "); scanf("%d",&n);

printf("\nPID\tATIME\tCPUB\n");

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

{ printf("%d\t",i);

scanf("%d%d",&proc[i].atime,&proc[i].cpub);

}

}

void start()

{ int pid;

addprocq();

pid=getfirstproc();

while(!finished())

{

if(pid!=-1)

{

ctime++;

proc[pid].cpub--;

if(proc[pid].cpub==0)

{

printf("\nProc %d completed at time %d..",pid,ctime);

ttime[pid]=ctime-proc[pid].atime;

pid=getfirstproc();

}

}

else

ctime++;

addprocq();

}

gchart[gi].etime=ctime;

}

void addprocq()

{

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

{ if(proc[i].atime==ctime)

attachtoq(i);

}

}

void attachtoq(int pid)

{

curr=(struct rqnode \*)malloc(sizeof(struct rqnode));

curr->pid=pid;curr->link=NULL;

if(first==NULL)

first=curr;

else

last->link=curr;

last=curr;

}

int finished()

{

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

{ if(proc[i].cpub!=0)

return(0);

}

return(1);

}

int getfirstproc()

{

int pid;

if(first==NULL)

return(-1);

pid=first->pid;

curr=first;

first=first->link;

free(curr);

addgchart(pid);

wtime[pid]=ctime-proc[pid].atime;

return(pid);

}

void addgchart(int pid)

{

gchart[++gi].pid=pid;

gchart[gi].stime=ctime;

gchart[gi-1].etime=gchart[gi].stime;

}

void dispgchart()

{

printf("\n");

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

printf("|----");

printf("|\n");

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

printf("| %d ",gchart[i].pid);

printf("|\n");

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

printf("|----");

printf("|\n");

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

printf("%d ",gchart[i].stime);

printf("%d\n",gchart[gi].etime);

}

void disptimes()

{ int sumwt=0,sumtt=0,pid;

printf("\nPID\tWtime");

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

{

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

sumwt+=wtime[i];

}

printf("\nAverage:%.2f",(float)sumwt/n);

printf("\n\*Turnaround Time\*");

printf("\nPID\t ttime");

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

{

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

sumtt+=ttime[i];

}

printf("\nAverage:%.2f",(float)sumtt/n);

}

///optimal

#include<stdio.h>

struct frmnode

{

int pno,refloc;

}frames[20];

int n;

int page\_found(int pno)

{

int fno;

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

if(frames[fno].pno==pno)

return fno;

return -1;

}

int get\_free\_frame()

{ int fno;

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

if (frames[fno].pno==-1)//free

return(fno);

return(-1);

}

int get\_opt\_frame()

{

int optfno=0,fno;

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

if(frames[fno].refloc>frames[optfno].refloc)

optfno=fno;

return optfno;

}

int main()

{

int p\_request[]={1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6};

int size=sizeof(p\_request)/2;

int page\_falts=0,i,j,k,fno;

printf("\nHow many frames:"); scanf("%d",&n);

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

{ frames[i].pno=-1;

}

printf("\nPageNo Page Frames Page Fault");

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

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

{

j=page\_found(p\_request[i]);

if(j==-1)

{

j=get\_free\_frame();

if (j==-1)

j=get\_opt\_frame();

page\_falts++;

frames[j].pno=p\_request[i];

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

{

for(k=i+1;k<size;k++)

if(frames[fno].pno==p\_request[k])

{ frames[fno].refloc=k; break;}

else

frames[fno].refloc=size;

}

printf("\n%4d\t ",p\_request[i]);

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

printf("%4d:%2d",frames[fno].pno,frames[fno].refloc);

printf(" : YES");

}

else

{

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

{

for(k=i+1;k<size;k++)

if(frames[fno].pno==p\_request[k])

{ frames[fno].refloc=k; break;}

else

frames[fno].refloc=size;

}

printf("\n%4d\t ",p\_request[i]);

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

printf("%4d:%2d",frames[fno].pno,frames[fno].refloc);

printf(" : NO");

}

}

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

printf("\n Number of Page\_Falts=%d",page\_falts);

}

///mfu

#include<stdio.h>

struct node

{

int pno,freq;

}frames[20];

int n;

int page\_found(int pno)

{

int fno;

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

if(frames[fno].pno==pno)

return fno;

return-1;

}

int get\_free\_frame()

{

int fno;

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

if(frames[fno].pno==-1)

return(fno);

return(-1);

}

int get\_mfu\_frame()

{

int fno;

int selfno=0;

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

if(frames[fno].freq>frames[selfno].freq)

selfno=fno;

return selfno;

}

void main()

{

int p\_request[]={1,3,3,2,5,2,1,4,2,2,5};

int size=11;

int page\_faults=0,i,j,fno;

printf("How many frames:"); scanf("%d",&n);

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

{

frames[i].pno=-1;

frames[i].freq=0;

}

printf("\n Pageno Page Frames Page Fault");

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

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

{

j=page\_found(p\_request[i]);

if(j==-1)

{

j=get\_free\_frame();

if(j==-1)

j=get\_mfu\_frame();

page\_faults++;

frames[j].pno=p\_request[i];

frames[j].freq=1;

printf("\n%4d\t",p\_request[i]);

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

printf("%4d:%2d",frames[fno].pno,frames[fno].freq);

printf(": YES");

}

else

{

printf("\n%4d\t",p\_request[i]);

frames[j].freq++;

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

printf("%4d:%2d",frames[fno].pno,frames[fno].freq);

printf(":NO");

}

}

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

printf("\nNumber of page Faults=%d",page\_faults);}

//non primitive priority scheduling

#include <stdio.h>

#include<string.h>

struct input

{

char pname[10];

int bt, at, tbt, ft, p;

} tab[10];

struct gantt

{

char pname[10];

int start, end;

} g[30], g1[30];

int n, i, time, prev, k;

void getinput()

{

printf("\nEnter No of Processes: ");

scanf("%d", &n);

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

{

printf("\nEnter Process Name: ");

scanf("%s", tab[i].pname);

printf("Arrival Time:");

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

printf("Burst Time: ");

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

tab[i].tbt = tab[i].bt;

printf("Enter the Priority:");

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

}

}

void printinput()

{

printf("\nPname\tAT\tBT\tpriority");

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

printf("\n%s\t%d\t%d\t%d", tab[i].pname, tab[i].at, tab[i].bt,tab[i].p);

}

void printoutput()

{

int TWT = 0, TTAT = 0;

float ATAT, AWT;

printf("\nPname\tAT\tBT\tpriority\tFT\tWT\tTAT");

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

{

printf("\n%s\t%d\t%d\t%d\t\t%d\t%d\t%d", tab[i].pname, tab[i].at, tab[i].bt,

tab[i].p,tab[i].ft, tab[i].ft-tab[i].at - tab[i].bt, tab[i].ft - tab[i].at);

TWT = TWT + (tab[i].ft - tab[i].at - tab[i].bt);

TTAT = TTAT + (tab[i].ft - tab[i].at);

}

printf("\nTotal WT: %d", TWT);

printf("\nTotal TAT:%d", TTAT);

AWT = (float)TWT / n;

ATAT = (float)TTAT / n;

printf("\nAverage WT: %f", AWT);

printf("\nAverage TAT:%f", ATAT);

}

void sort()

{

int pass;

struct input temp;

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

{

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

{

if (tab[i].at > tab[i + 1].at)

{

temp = tab[i];

tab[i] = tab[i + 1];

tab[i + 1] = temp;

}

}

}

}

int arrived(int time)

{

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

{

if (tab[i].at <= time && tab[i].tbt != 0)

return 1;

}

return 0;

}

int gethighpriority(int time)

{

int processpos, min = 99;

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

{

if (tab[i].at <= time && tab[i].tbt != 0 && tab[i].p < min)

{

min = tab[i].p;

processpos = i;

}

}

return processpos;

}

void processinput()

{

int finish = 0, j;

k = 0;

while (finish != n)

{

if (arrived(time))

{

i = gethighpriority(time);

for (j = 0; j < tab[i].bt; j++)

{

time++;

tab[i].tbt--;

g[k].start = prev;

g[k].end = time;

prev = time;

strcpy(g[k++].pname, tab[i].pname);

tab[i].ft = time;

if (tab[i].tbt == 0)

{

finish++;

break;

}

}

}

else

{

time++;

g[k].start = prev;

g[k].end = time;

strcpy(g[k++].pname, "idle");

prev = time;

}

}

}

void ganttchart()

{

int i, j = 0;

printf("\ngantt chart is:\n");

g1[0] = g[0];

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

{

if (strcmp(g[i].pname, g1[j].pname) == 0)

g1[j].end = g[i].end;

else

{

j++;

g1[j] = g[i];

}

}

printf("\nStart\tpname\tEnd");

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

{

printf("\n%d\t%s\t%d", g1[i].start, g1[i].pname, g1[i].end);

}

}

int main()

{

getinput();

printinput();

sort();

processinput();

printoutput();

ganttchart();

}

///nice ()system call

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

void main()

{

pid\_t pid;

int retnice;

retnice=nice(2);

pid=fork();

if(pid==0)

{

retnice=nice(5);

printf("child gets higher CPU priority%d\n",retnice);

}

else

{

retnice=nice(10);

printf("parent gets lower CPU priority%d\n",retnice);

sleep(1);

}

}

///bubble sort and insertion sort

#include<stdio.h>

#include<sys/types.h>

#include<stdlib.h>

#include<unistd.h>

void bubblesort(int a[20],int n)

{

int i,j,temp;

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

{

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

{

if(a[j]>a[j+1])

{

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

}

}

}

}

void insertionsort(int a[20],int n)

{

int i,j,temp;

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

{

temp=a[i];

j=i-1;

while(j>=0 &&temp<=a[j])

{

a[j+1]=a[j];

j=j-1;

}

a[j+1]=temp;

}

}

void fork1()

{

int a[30],a1[30],n,i;

printf("\n\tEnter the size of array:");

scanf("%d",&n);

printf("\n\tEnter the array element:");

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

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

int pid=fork();

if(pid==0)

{

sleep(10);

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

printf("child process id=%d\n",getpid());

insertionsort(a,n);

printf("\n Elements sorted using insertion sort:");

printf("\n");

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

printf("%d\t",a[i]);

printf("\n parent process id=%d\n",getpid());

}

else

{

printf("\n parent process");

printf("\n parent process id=%d\n",getpid());

bubblesort(a,n);

printf("\n Elements sorted using bubble sort:");

printf("\n");

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

printf("%d\t",a[i]);

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

}

}

int main()

{

fork1();

return 0;

}

//use fork()andsleep()

#include<stdio.h>

#include<sys/types.h>

#include<unistd.h>

int main()

{

int pid;

pid=getpid();

printf("current process id is:%d\n",pid);

printf("\n forking child process...\n");

pid=fork();

if(pid>0)

{

printf("\n process can not be created");

}

else

{

if(pid==0)

{

printf("\n child process is sleeping...");

sleep(5);

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

}

else

{

printf("\n parent process completed...\n");

}

}

return 0;

}