**Slip 1**

**Write the simulation program to implement demand paging and show the page scheduling and total number of page faults according to the LFU page replacement algorithm. Assume the memory of n frames. Reference String : 3,4,5,4,3,4,7,2,4,5,6,7,2,4,6**

#include<stdio.h>

#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,

sp,m,n,count[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:\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("%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(frames[i]==pno)

return i;

}

return -1;

}

int get\_lfu(int sp)

{

int i,min\_i,min=9999;

i=sp;

do

{

if(count[i]<min)

{

min = count[i];

min\_i = i;

}

i=(i+1)%n;

}while(i!=sp);

return min\_i;

}

void lfu()

{

int i,j,k;

for(i=0;i<m && sp<n;i++)

{

k=search(ref[i]);

if(k==-1)

{

frames[sp]=ref[i];

count[sp]++;

faults++;

sp++;

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

mem[j][i]=frames[j];

}

else

count[k]++;

}

sp=0;

for(;i<m;i++)

{

k = search(ref[i]);

if(k==-1)

{

sp = get\_lfu(sp);

frames[sp] = ref[i];

count[sp]=1;

faults++;

sp = (sp+1)%n;

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

mem[j][i] = frames[j];

}

else

count[k]++;

}

}

int main()

{

accept();

lfu();

disp();

return 0;

}

**Write a C program to implement the shell which displays the command prompt “myshell$”. It accepts the command, tokenize the command line and execute it by creating the child process. Also implement the additional command ‘typeline’ as typeline +n filename :- To print first n lines in the file. typeline -a filename :- To print all lines in the file.**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void typeline(char \*fn, char \*op)

{

int fh,i,j,n;

char c;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s not found.\n",fn);

return;

}

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

{

while(read(fh,&c,1)>0)

printf("%c",c);

close(fh);

return;

}

n = atoi(op);

if(n>0)

{

i=0;

while(read(fh,&c,1)>0)

{

printf("%c",c);

if(c=='\n') i++;

if(i==n) break;

}

}

if(n<0)

{

i=0;

while(read(fh,&c,1)>0)

{

if(c=='\n') i++;

}

lseek(fh,0,SEEK\_SET);

j=0;

while(read(fh,&c,1)>0)

{

if(c=='\n') j++;

if(j==i+n) break;

}

while(read(fh,&c,1)>0)

{

printf("%c",c);

}

}

close(fh);

}

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

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

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Slip 2**

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the FIFO page replacement algorithm. Assume the memory of n frames. Reference String : 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6**

#include<stdio.h>

#define MAX 20

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

void accept()

{

int i;

printf("Enter no.of frames:");

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("%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(frames[i]==pno)

return i;

}

return -1;

}

void fifo()

{

int i,j;

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

{

if(search(ref[i])==-1)

{

frames[sp] = ref[i];

sp = (sp+1)%n;

faults++;

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

mem[j][i] = frames[j];

}

}

}

int main()

{

accept();

fifo();

disp();

return 0;

}

**Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following ‘list’ commands as**

**myshell$ list f dirname :- To print names of all the files in current directory.**

**myshell$ list n dirname :- To print the number of all entries in the current directory**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void list(char \*dn, char op)

{

DIR \*dp;

struct dirent \*entry;

int dc=0,fc=0;

dp = opendir(dn);

if(dp==NULL)

{

printf("Dir %s not found.\n",dn);

return;

}

switch(op)

{

case 'f':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_REG)

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

}

break;

case 'n':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_DIR) dc++;

if(entry->d\_type==DT\_REG) fc++;

}

printf("%d Dir(s)\t%d File(s)\n",dc,fc);

break;

case 'i':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_REG)

printf("%s\t%d\n",entry->d\_name,entry->d\_fileno);

}

}

closedir(dp);

}

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

list(args[2],args[1][0]);

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

**Slip 3**

**Write the simulation program to implement demand paging and show the page scheduling and total number of page faults according to the LRU (using counter method) page replacement algorithm. Assume the memory of n frames.**

**Reference String : 3,5,7,2,5,1,2,3,1,3,5,3,1,6,2**

#include<stdio.h>

#define MAX 20

int frames[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:\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("%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(frames[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)

{

frames[sp]=ref[i];

time[sp]=i;

faults++;

sp++;

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

mem[j][i]=frames[j];

}

else

time[k]=i;

}

for(;i<m;i++)

{

k = search(ref[i]);

if(k==-1)

{

sp = get\_lru();

frames[sp] = ref[i];

time[sp] = i;

faults++;

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

mem[j][i] = frames[j];

}

else

time[k]=i;

}

}

int main()

{

accept();

lru();

disp();

return 0;

}

**Write a programto implement the toy shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands. count c filename :- To print number of characters in the file. count w filename :- To print number of words in the file. count l filename :- To print number of lines in the file.**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void count(char \*fn, char op)

{

int fh,cc=0,wc=0,lc=0;

char c;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s not found.\n",fn);

return;

}

while(read(fh,&c,1)>0)

{

if(c==' ') wc++;

else if(c=='\n')

{

wc++;

lc++;

}

cc++;

}

close(fh);

switch(op)

{

case 'c':

printf("No.of characters:%d\n",cc);

break;

case 'w':

printf("No.of words:%d\n",wc);

break;

case 'l':

printf("No.of lines:%d\n",lc);

break;

}

}

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

count(args[2],args[1][0]);

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Slip 4**

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the MFU page replacement algorithm. Assume the memory of n frames. Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2**

#include<stdio.h>

#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,

sp,m,n,count[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:\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("%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(frames[i]==pno)

return i;

}

return -1;

}

int get\_mfu(int sp)

{

int i,max\_i,max=-9999;

i=sp;

do

{

if(count[i]>max)

{

max = count[i];

max\_i = i;

}

i=(i+1)%n;

}while(i!=sp);

return max\_i;

}

void mfu()

{

int i,j,k;

for(i=0;i<m && sp<n;i++)

{

k=search(ref[i]);

if(k==-1)

{

frames[sp]=ref[i];

count[sp]++;

faults++;

sp++;

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

mem[j][i]=frames[j];

}

else

count[k]++;

}

sp=0;

for(;i<m;i++)

{

k = search(ref[i]);

if(k==-1)

{

sp = get\_mfu(sp);

frames[sp] = ref[i];

count[sp]=1;

faults++;

sp = (sp+1)%n;

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

mem[j][i] = frames[j];

}

else

count[k]++;

}

}

int main()

{

accept();

mfu();

disp();

return 0;

}

**Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.**

**myshell$ search a filename pattern :- To search all the occurrence of pattern in the file. myshell$ search c filename pattern :- To count the number of occurrence of pattern in the file.**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void search(char \*fn, char op, char \*pattern)

{

int fh,count=0,i=0,j=0;

char buff[255],c,\*p;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s Not Found\n",fn);

return;

}

switch(op)

{

case 'f':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

i++;

if(strstr(buff,pattern))

{

printf("%d: %s",i,buff);

break;

}

}

}

break;

case 'c':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

p = buff;

while(p=strstr(p,pattern))

{

count++;

p++;

}

}

}

printf("Total No.of Occurrences = %d\n",count);

break;

case 'a':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j = 0;

i++;

if(strstr(buff,pattern))

printf("%d: %s",i,buff);

}

}

}//switch

close(fh);

}//search

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

search(args[3],args[1][0],args[2]);

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Slip 5**

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the optimal page replacement algorithm. Assume the memory of n frames.**

**Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2**

#include<stdio.h>

int nor,nof,refstring[30],F[10];

void accept()

{

int i;

printf("\nEnter the Reference String:\n ");

for(i=0;i<nor;i++)

{

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

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

}

}

int search(int page)

{

int i;

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

{

if(page==F[i])

return i;

}

return -1;

}

int getfno(int i)

{

int fno,prev,pos=0,fpos,flag;

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

{

flag=0;

for(prev=i+1;prev<nor;prev++)

{

if(F[fno]==refstring[prev])

{

flag=1;

if(prev>pos) //2<3

{

pos=prev; //pos=2

fpos=fno; //fpos=2

}

break;

}

}

if(flag==0)

{

fpos=fno;

break;

}

}

//printf("\nfpos=%d",fpos);

return fpos;

}

void optimal()

{

int i,j,k,fno,fault=0;

for(fno=0,i=0;fno<nof && i<nor;i++)

{

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

k=search(refstring[i]);

//printf("\nk=%d",k);

if(k==-1)

{

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

fno++;

}

}

while(i<nor)

{

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

k=search(refstring[i]);

//printf("\nk=%d",k);

if(k==-1)

{

fno=getfno(i);

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

}

i++;

}

printf("\nTotal no of Page fault: %d",fault);

}

main()

{

printf("\nEnter the Length of the string: ");

scanf("%d",&nor);

printf("\nEnter no. of Frames: ");

scanf("%d",&nof);

accept();

optimal();

}

**Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.**

**myshell$ search f filename pattern :- To display first occurrence of pattern in the file. myshell$ search c filename pattern :- To count the number of occurrence of pattern in the file.**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void search(char \*fn, char op, char \*pattern)

{

int fh,count=0,i=0,j=0;

char buff[255],c,\*p;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s Not Found\n",fn);

return;

}

switch(op)

{

case 'f':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

i++;

if(strstr(buff,pattern))

{

printf("%d: %s",i,buff);

break;

}

}

}

break;

case 'c':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

p = buff;

while(p=strstr(p,pattern))

{

count++;

p++;

}

}

}

printf("Total No.of Occurrences = %d\n",count);

break;

case 'a':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j = 0;

i++;

if(strstr(buff,pattern))

printf("%d: %s",i,buff);

}

}

}//switch

close(fh);

}//search

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

search(args[3],args[1][0],args[2]);

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Slip 6**

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the MRU page replacement algorithm. Assume the memory of n frames.**

**Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2**

#include<stdio.h>

int RefString[20],PT[10],count[5],seq[4],nof,nor,frmno;

void Accept()

{

int i;

printf("\n enter the reference sting:");

for(i=0;i<nor;i++)

{

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

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

}

}

int search(int s)

{

int i;

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

if(PT[i]==s)

return(i);

return(-1);

}

int checkcount(int e)

{

int i,cnt=0,Pos=99,Posi,j,flag,cnt1=0;

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

{

if(count[i]>cnt)

{

cnt=count[i];

// frmno=i;

}

}

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

{

if(count[i]==cnt)

{

for(j=e-1;j>=0;j--)

{

if(PT[i]==RefString[j])

{

if(j<Pos)

{

Pos=j;

Posi=i;

}

break;

}

}

}

}

return Posi;

}

void MFU()

{

int i,j,k,faults=0,frameno,cnt;

for(k=0,i=0;k<nof && i<nor;i++)

{

getch();

cnt=0;

printf("%2d",RefString[i]);

frameno=search(RefString[i]);

if(frameno==-1)

{

PT[k]=RefString[i];

count[k]=cnt;

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

{

if(PT[j])

printf("%2d(%d)\t",PT[j],count[j]);

}

faults++;

k++;

}

else

{

cnt=count[frameno];

count[frameno]=++cnt;

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

{

if(PT[j])

printf("%2d(%d)\t",PT[j],count[j]);

}

printf(" No page fault");

}

printf("\n\n");

}

k=0;

while(i<nor)

{

getch();

cnt=0;

printf("%2d",RefString[i]);

frameno=search(RefString[i]);

if(frameno==-1)

{

k=checkcount(i);

PT[k]=RefString[i];

count[k]=cnt;

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

{

printf("%2d(%d)\t",PT[j],count[j]);

}

faults++;

}

else

{

cnt=count[frameno];

count[frameno]=++cnt;

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

{

printf("%2d(%d)\t",PT[j],count[j]);

}

printf(" No page fault");

}

i++;

printf("\n\n");

}

printf("total page faults:%d",faults);

}

int main()

{

printf("\n enter the length of Reference string:");

scanf("%d",&nor);

printf("\n enter the number of frames:");

scanf("%d",&nof);

Accept();

MFU();

}

**Write a programto implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.**

**myshell$ search f filename pattern :- To display first occurrence of pattern in the file. myshell$ search a filename pattern :- To search all the occurrence of pattern in the file.**

**//MRU not found therefore, MFU**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void search(char \*fn, char op, char \*pattern)

{

int fh,count=0,i=0,j=0;

char buff[255],c,\*p;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s Not Found\n",fn);

return;

}

switch(op)

{

case 'f':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

i++;

if(strstr(buff,pattern))

{

printf("%d: %s",i,buff);

break;

}

}

}

break;

case 'c':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

p = buff;

while(p=strstr(p,pattern))

{

count++;

p++;

}

}

}

printf("Total No.of Occurrences = %d\n",count);

break;

case 'a':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j = 0;

i++;

if(strstr(buff,pattern))

printf("%d: %s",i,buff);

}

}

}//switch

close(fh);

}//search

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

search(args[3],args[1][0],args[2]);

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Slip 7**

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the Optimal page replacement algorithm. Assume the memory of n frames.**

**Reference String : 7, 5, 4, 8, 5, 7, 2, 3, 1, 3, 5, 9, 4, 6, 2**

#include<stdio.h>

int nor,nof,refstring[30],F[10];

void accept()

{

int i;

printf("\nEnter the Reference String:\n ");

for(i=0;i<nor;i++)

{

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

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

}

}

int search(int page)

{

int i;

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

{

if(page==F[i])

return i;

}

return -1;

}

int getfno(int i)

{

int fno,prev,pos=0,fpos,flag;

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

{

flag=0;

for(prev=i+1;prev<nor;prev++)

{

if(F[fno]==refstring[prev])

{

flag=1;

if(prev>pos) //2<3

{

pos=prev; //pos=2

fpos=fno; //fpos=2

}

break;

}

}

if(flag==0)

{

fpos=fno;

break;

}

}

//printf("\nfpos=%d",fpos);

return fpos;

}

void optimal()

{

int i,j,k,fno,fault=0;

for(fno=0,i=0;fno<nof && i<nor;i++)

{

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

k=search(refstring[i]);

//printf("\nk=%d",k);

if(k==-1)

{

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

fno++;

}

}

while(i<nor)

{

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

k=search(refstring[i]);

//printf("\nk=%d",k);

if(k==-1)

{

fno=getfno(i);

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

}

i++;

}

printf("\nTotal no of Page fault: %d",fault);

}

main()

{

printf("\nEnter the Length of the string: ");

scanf("%d",&nor);

printf("\nEnter no. of Frames: ");

scanf("%d",&nof);

accept();

optimal();

}

**Write a program to implement shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.**

**myshell$ search a filename pattern :- To search all the occurrence of pattern in the file. myshell$ search c filename pattern :- To count the number of occurrence of pattern in the file.**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void search(char \*fn, char op, char \*pattern)

{

int fh,count=0,i=0,j=0;

char buff[255],c,\*p;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s Not Found\n",fn);

return;

}

switch(op)

{

case 'f':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

i++;

if(strstr(buff,pattern))

{

printf("%d: %s",i,buff);

break;

}

}

}

break;

case 'c':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

p = buff;

while(p=strstr(p,pattern))

{

count++;

p++;

}

}

}

printf("Total No.of Occurrences = %d\n",count);

break;

case 'a':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j = 0;

i++;

if(strstr(buff,pattern))

printf("%d: %s",i,buff);

}

}

}//switch

close(fh);

}//search

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

search(args[3],args[1][0],args[2]);

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Slip 8**

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the LRU page replacement algorithm. Assume the memory of n frames.**

**Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2**

#include<stdio.h>

int nor,nof,refstring[30],F[10];

void accept()

{

int i;

printf("\nEnter the Reference String:\n ");

for(i=0;i<nor;i++)

{

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

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

}

}

int search(int page)

{

int i;

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

{

if(page==F[i])

return i;

}

return -1;

}

int getfno(int i)

{

int fno,prev,pos=99,fpos;

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

{

for(prev=i-1;prev>=0;prev--)

{

if(F[fno]==refstring[prev])

{

if(prev<pos)

{

pos=prev;

fpos=fno;

}

break;

}

}

}

;

return fpos;

}

void LRU()

{

int i,j,k,fno,fault=0;

for(fno=0,i=0;fno<nof && i<nor;i++)

{

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

k=search(refstring[i]);

if(k==-1)

{

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

fno++;

}

}

while(i<nor)

{

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

k=search(refstring[i]);

if(k==-1)

{

fno=getfno(i);

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

}

i++;

}

printf("\nTotal no of Page fault: %d",fault);

}

main()

{

printf("\nEnter the Length of the string: ");

scanf("%d",&nor);

printf("\nEnter no. of Frames: ");

scanf("%d",&nof);

accept();

LRU();

}

**Write a programto implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.**

**myshell$ search f filename pattern :- To display first occurrence of pattern in the file. myshell$ search c filename pattern :- To count the number of occurrence of pattern in the file.**

[Home](https://iprogramx.blogspot.com/)TYBcs OS-Syspro Slip 13-2 | IProgramX

# **TYBcs OS-Syspro Slip 13-2 | IProgramX**

*by -* [IProgram X](https://www.blogger.com/profile/17054938767780569057) *on -* January 25, 2019

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void search(char \*fn, char op, char \*pattern)

{

int fh,count=0,i=0,j=0;

char buff[255],c,\*p;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s Not Found\n",fn);

return;

}

switch(op)

{

case 'f':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

i++;

if(strstr(buff,pattern))

{

printf("%d: %s",i,buff);

break;

}

}

}

break;

case 'c':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

p = buff;

while(p=strstr(p,pattern))

{

count++;

p++;

}

}

}

printf("Total No.of Occurrences = %d\n",count);

break;

case 'a':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j = 0;

i++;

if(strstr(buff,pattern))

printf("%d: %s",i,buff);

}

}

}//switch

close(fh);

}//search

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

search(args[3],args[1][0],args[2]);

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Slip 9**

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the FIFO page replacement algorithm. Assume the memory of n frames.**

**Reference String : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2**

#include<stdio.h>

int nor,nof,refstring[30],F[10];

void accept()

{

int i;

printf("\nEnter the Reference String:\n ");

for(i=0;i<nor;i++)

{

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

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

}

}

int search(int page)

{

int i;

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

{

if(page==F[i])

return i;

}

return -1;

}

void FIFO()

{

int i,j,k,fno=0,fault=0;

for(i=0;i<nor;i++)

{

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

k=search(refstring[i]);

//printf("\nk=%d",k);

if(k==-1)

{

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

fno=(fno+1)%nof;

}

}

printf("\nTotal no of Page fault: %d",fault);

}

main()

{

printf("\nEnter the Length of the string: ");

scanf("%d",&nor);

printf("\nEnter no. of Frames: ");

scanf("%d",&nof);

accept();

FIFO();

}

**Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.**

**myshell$ search f filename pattern :- To display first occurrence of pattern in the file. myshell$ search a filename pattern :- To search all the occurrence of pattern in the file.**

[Home](https://iprogramx.blogspot.com/)TYBcs OS-Syspro Slip 13-2 | IProgramX

# **TYBcs OS-Syspro Slip 13-2 | IProgramX**

*by -* [IProgram X](https://www.blogger.com/profile/17054938767780569057) *on -* January 25, 2019

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void search(char \*fn, char op, char \*pattern)

{

int fh,count=0,i=0,j=0;

char buff[255],c,\*p;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s Not Found\n",fn);

return;

}

switch(op)

{

case 'f':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

i++;

if(strstr(buff,pattern))

{

printf("%d: %s",i,buff);

break;

}

}

}

break;

case 'c':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j=0;

p = buff;

while(p=strstr(p,pattern))

{

count++;

p++;

}

}

}

printf("Total No.of Occurrences = %d\n",count);

break;

case 'a':

while(read(fh,&c,1))

{

buff[j++]=c;

if(c=='\n')

{

buff[j]='\0';

j = 0;

i++;

if(strstr(buff,pattern))

printf("%d: %s",i,buff);

}

}

}//switch

close(fh);

}//search

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

search(args[3],args[1][0],args[2]);

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Slip 10**

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the FIFO page replacement algorithm. Assume the memory of n frames.**

**Reference String : 2, 4, 5, 6, 9, 4, 7, 3, 4, 5, 6, 7, 2, 4, 7, 1**

#include<stdio.h>

int nor,nof,refstring[30],F[10];

void accept()

{

int i;

printf("\nEnter the Reference String:\n ");

for(i=0;i<nor;i++)

{

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

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

}

}

int search(int page)

{

int i;

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

{

if(page==F[i])

return i;

}

return -1;

}

void FIFO()

{

int i,j,k,fno=0,fault=0;

for(i=0;i<nor;i++)

{

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

k=search(refstring[i]);

//printf("\nk=%d",k);

if(k==-1)

{

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

fno=(fno+1)%nof;

}

}

printf("\nTotal no of Page fault: %d",fault);

}

main()

{

printf("\nEnter the Length of the string: ");

scanf("%d",&nor);

printf("\nEnter no. of Frames: ");

scanf("%d",&nof);

accept();

FIFO();

}

**Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following ‘list’ commands as**

**myshell$ list f dirname :- To print names of all the files in current directory.**

**myshell$ list i dirname :- To print names and inodes of the files in the current directory.**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void list(char \*dn, char op)

{

DIR \*dp;

struct dirent \*entry;

int dc=0,fc=0;

dp = opendir(dn);

if(dp==NULL)

{

printf("Dir %s not found.\n",dn);

return;

}

switch(op)

{

case 'f':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_REG)

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

}

break;

case 'n':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_DIR) dc++;

if(entry->d\_type==DT\_REG) fc++;

}

printf("%d Dir(s)\t%d File(s)\n",dc,fc);

break;

case 'i':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_REG)

printf("%s\t%d\n",entry->d\_name,entry->d\_fileno);

}

}

closedir(dp);

}

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

list(args[2],args[1][0]);

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Slip 11**

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the LFU page replacement algorithm. Assume the memory of n frames.**

**Reference String : 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6**

#include<stdio.h>

#define MAX 20

int frames[MAX],ref[MAX],mem[MAX][MAX],faults,

sp,m,n,count[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:\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("%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(frames[i]==pno)

return i;

}

return -1;

}

int get\_lfu(int sp)

{

int i,min\_i,min=9999;

i=sp;

do

{

if(count[i]<min)

{

min = count[i];

min\_i = i;

}

i=(i+1)%n;

}while(i!=sp);

return min\_i;

}

void lfu()

{

int i,j,k;

for(i=0;i<m && sp<n;i++)

{

k=search(ref[i]);

if(k==-1)

{

frames[sp]=ref[i];

count[sp]++;

faults++;

sp++;

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

mem[j][i]=frames[j];

}

else

count[k]++;

}

sp=0;

for(;i<m;i++)

{

k = search(ref[i]);

if(k==-1)

{

sp = get\_lfu(sp);

frames[sp] = ref[i];

count[sp]=1;

faults++;

sp = (sp+1)%n;

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

mem[j][i] = frames[j];

}

else

count[k]++;

}

}

int main()

{

accept();

lfu();

disp();

return 0;

}

**Write a C program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following ‘list’ commands as**

**myshell$ list f dirname :- To print names of all the files in current directory.**

**myshell$ list n dirname :- To print the number of all entries in the current directory**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void list(char \*dn, char op)

{

DIR \*dp;

struct dirent \*entry;

int dc=0,fc=0;

dp = opendir(dn);

if(dp==NULL)

{

printf("Dir %s not found.\n",dn);

return;

}

switch(op)

{

case 'f':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_REG)

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

}

break;

case 'n':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_DIR) dc++;

if(entry->d\_type==DT\_REG) fc++;

}

printf("%d Dir(s)\t%d File(s)\n",dc,fc);

break;

case 'i':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_REG)

printf("%s\t%d\n",entry->d\_name,entry->d\_fileno);

}

}

closedir(dp);

}

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

list(args[2],args[1][0]);

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Slip12**

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the LRU page replacement algorithm. Assume the memory of n frames. Reference String : 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6**

#include<stdio.h>

int nor,nof,refstring[30],F[10];

void accept()

{

int i;

printf("\nEnter the Reference String:\n ");

for(i=0;i<nor;i++)

{

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

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

}

}

int search(int page)

{

int i;

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

{

if(page==F[i])

return i;

}

return -1;

}

int getfno(int i)

{

int fno,prev,pos=99,fpos;

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

{

for(prev=i-1;prev>=0;prev--)

{

if(F[fno]==refstring[prev])

{

if(prev<pos)

{

pos=prev;

fpos=fno;

}

break;

}

}

}

;

return fpos;

}

void LRU()

{

int i,j,k,fno,fault=0;

for(fno=0,i=0;fno<nof && i<nor;i++)

{

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

k=search(refstring[i]);

if(k==-1)

{

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

fno++;

}

}

while(i<nor)

{

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

k=search(refstring[i]);

if(k==-1)

{

fno=getfno(i);

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

}

i++;

}

printf("\nTotal no of Page fault: %d",fault);

}

main()

{

printf("\nEnter the Length of the string: ");

scanf("%d",&nor);

printf("\nEnter no. of Frames: ");

scanf("%d",&nof);

accept();

LRU();

}

**Write a program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following ‘list’ commands as**

**myshell$ list f dirname :- To print names of all the files in current directory.**

**myshell$ list n dirname :- To print the number of all entries in the current directory**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <dirent.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void list(char \*dn, char op)

{

DIR \*dp;

struct dirent \*entry;

int dc=0,fc=0;

dp = opendir(dn);

if(dp==NULL)

{

printf("Dir %s not found.\n",dn);

return;

}

switch(op)

{

case 'f':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_REG)

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

}

break;

case 'n':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_DIR) dc++;

if(entry->d\_type==DT\_REG) fc++;

}

printf("%d Dir(s)\t%d File(s)\n",dc,fc);

break;

case 'i':

while(entry=readdir(dp))

{

if(entry->d\_type==DT\_REG)

printf("%s\t%d\n",entry->d\_name,entry->d\_fileno);

}

}

closedir(dp);

}

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

list(args[2],args[1][0]);

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Slip 13**

**Write a C program to implement the shell which displays the command prompt “myshell$”. It accepts the command, tokenize the command line andexecute it by creating the child process. Also implement the additional command ‘typeline’ as**

**typeline -a filename :- To print all lines in the file.**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void typeline(char \*fn, char \*op)

{

int fh,i,j,n;

char c;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s not found.\n",fn);

return;

}

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

{

while(read(fh,&c,1)>0)

printf("%c",c);

close(fh);

return;

}

n = atoi(op);

if(n>0)

{

i=0;

while(read(fh,&c,1)>0)

{

printf("%c",c);

if(c=='\n') i++;

if(i==n) break;

}

}

if(n<0)

{

i=0;

while(read(fh,&c,1)>0)

{

if(c=='\n') i++;

}

lseek(fh,0,SEEK\_SET);

j=0;

while(read(fh,&c,1)>0)

{

if(c=='\n') j++;

if(j==i+n) break;

}

while(read(fh,&c,1)>0)

{

printf("%c",c);

}

}

close(fh);

}

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

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

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Write the simulation program for Round Robin scheduling for given time quantum. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give the Gantt chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.**

#include<stdio.h>

struct input

{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

char pname[10];

int start,end;

}g[30],g1[30];

int n,time,prev,k,tq;

void getinput()

{

int i;

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

scanf("%d",&n);

printf("\nEnter Time quantum: ");

scanf("%d",&tq);

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;

}

}

void printinput()

{

// int TWT=0,TTAT=0;

int i;

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

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

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

}

void printoutput()

{

int TWT=0,TTAT=0,i;

float ATAT,AWT;

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

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

{

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

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)

{

int i;

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

{

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

return 1;

}

return 0;

}

void processinput()

{

int finish=0,j;

int i=0;

k=0;

while(finish!=n)

{

if(arrived(time))

{

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

{

for(j=0;j<tq;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;

}

if(time<tab[(i+1)%n].at)

i=0;

else

i=(i+1)%n;

}

}

void ganttchart()

{

int i,j=0;

printf("\n\*\*\*\*\*\*Each Unit Gantt chart\*\*\*\*\*\*");

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

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

{

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

}

printf("\n\*\*\*\*\*\*\*\*Final Gantt Chart\*\*\*\*\*\*\*");

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();

printf("\nData After Sorting: ");

printinput();

processinput();

printoutput();

ganttchart();

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

{

tab[i].tbt=tab[i].bt=rand()%10+1;

tab[i].at=tab[i].ft+2;

}

printinput();

processinput();

printoutput();

ganttchart();

}

**Slip 14**

**Write a C program to implement the shell which displays the command prompt “myshell$”. It accepts the command, tokenize the command line andexecute it by creating the child process. Also implement the additional command ‘typeline’ as**

**typeline +n filename :- To print first n lines in the file.**

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

void make\_toks(char \*s, char \*tok[])

{

int i=0;

char \*p;

p = strtok(s," ");

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void typeline(char \*fn, char \*op)

{

int fh,i,j,n;

char c;

fh = open(fn,O\_RDONLY);

if(fh==-1)

{

printf("File %s not found.\n",fn);

return;

}

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

{

while(read(fh,&c,1)>0)

printf("%c",c);

close(fh);

return;

}

n = atoi(op);

if(n>0)

{

i=0;

while(read(fh,&c,1)>0)

{

printf("%c",c);

if(c=='\n') i++;

if(i==n) break;

}

}

if(n<0)

{

i=0;

while(read(fh,&c,1)>0)

{

if(c=='\n') i++;

}

lseek(fh,0,SEEK\_SET);

j=0;

while(read(fh,&c,1)>0)

{

if(c=='\n') j++;

if(j==i+n) break;

}

while(read(fh,&c,1)>0)

{

printf("%c",c);

}

}

close(fh);

}

int main()

{

char buff[80],\*args[10];

int pid;

while(1)

{

printf("myshell$");

fflush(stdin);

fgets(buff,80,stdin);

buff[strlen(buff)-1]='\0';

make\_toks(buff,args);

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

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

else

{

pid = fork();

if(pid>0)

wait();

else

{

if(execvp(args[0],args)==-1)

printf("Bad command.\n");

}

}

}

return 0;

}

**Write a C program to simulate Non-preemptive Shortest Job First (SJF) – scheduling. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time**

#include<stdio.h>

struct input

{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

int start,end;

char pname[10];

}g[50],g1[10];

int n,i,k,time,prev;

void getinput()

{

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

scanf("%d",&n);

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

{

printf("\nProcess Name: ");

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

printf("Burst Time: ");

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

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

printf("Arrival Time:" );

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

}

}

void printinput()

{

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

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

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

}

void sort()

{

struct input temp;

int j;

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

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

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

{

temp=tab[j];

tab[j]=tab[j+1];

tab[j+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 getsmallburst(int time)

{

int min=99,mini;

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

{

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

{

min=tab[i].tbt;

mini=i;

}

}

return mini;

}

void processinput()

{

int j,finish=0;

// time=tab[0].at;

while(finish!=n)

{

if(arrived(time))

{

i=getsmallburst(time);

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

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

// printinput();

prev=time;

tab[i].ft=time;

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

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

{

finish++;

break;

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

prev=time;

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

}

}

// printinput();

}

void printoutput()

{

int TTAT=0,TWT=0;

float ATAT,AWT;

printf("\n\*\*\*\*\*\*Final Table\*\*\*\*\*");

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

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

{

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

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

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

}

ATAT=(float)TTAT/n;

AWT=(float)TWT/n;

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

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

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

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

}

void printganttchart()

{

int j=0;

g1[0]=g[0];

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

{

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

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

else

{

j++;

g1[j]=g[i];

}

}

printf("\n\*\*\*\*\*\*Each unit Gantt chart\*\*\*\*\*\*");

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

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

printf("\n\*\*\*\*\*\*Final Gantt chart\*\*\*\*\*\*");

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();

printf("\nEntered data is: ");

printinput();

sort();

printf("\nData after Sorting" );

printinput();

processinput();

printoutput();

printganttchart();

}

**Slip 15**

**Write a C program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following ‘list’ commands as**

**myshell$ list f dirname :- To print names of all the files in current directory.**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<dirent.h>

char \*buff,\*t1,\*t2,\*t3,ch;

int pid;

void list(char t2,char \*t3)

{

DIR \*dir;

struct dirent \*entry;

int cnt=0;

dir=opendir(t3);

if (dir==NULL)

{

printf("Directory %s not found",t3);

return;

}

switch(t2)

{

case 'f' : while((entry=readdir(dir))!=NULL)

{

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)

{

printf("\n%s\t %d",entry->d\_name,entry->d\_ino);

}

break;

default : printf("Invalid argument");

}

closedir(dir);

}

main()

{

while(1)

{

printf("myshell$");

fflush(stdin);

t1=(char \*)malloc(80);

t2=(char \*)malloc(80);

t3=(char \*)malloc(80);

buff=(char \*)malloc(80);

fgets(buff,80,stdin);

sscanf(buff,"%s %s %s",t1,t2,t3);

if(strcmp(t1,"pause")==0)

exit(0);

else if(strcmp(t1,"list")==0)

list(t2[0],t3);

else

{

pid=fork();

if(pid<0)

printf("Child process is not created\n");

else if(pid==0)

{

execlp("/bin",NULL);

if(strcmp(t1,"exit")==0)

exit(0);

system(buff);

}

else

{

wait(NULL);

exit(0);

}

}

}

}

**Write the program to simulate preemptive Shortest Job First (SJF) – scheduling. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. Theoutput should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time**

#include<stdio.h>

struct input

{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

int start,end;

char pname[10];

}g[50],g1[10];

int n,i,k,time,prev;

void getinput()

{

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

scanf("%d",&n);

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

{

printf("\nProcess Name: ");

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

printf("Burst Time: ");

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

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

printf("Arrival Time:" );

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

}

}

void printinput()

{

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

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

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

}

void sort()

{

struct input temp;

int j;

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

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

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

{

temp=tab[j];

tab[j]=tab[j+1];

tab[j+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 getsmallburst(int time)

{

int min=99,mini;

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

{

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

{

min=tab[i].tbt;

mini=i;

}

}

return mini;

}

void processinput()

{

int j,finish=0;

// time=tab[0].at;

while(finish!=n)

{

if(arrived(time))

{

i=getsmallburst(time);

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

// printinput();

prev=time;

tab[i].ft=time;

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

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

{

finish++;

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

prev=time;

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

}

}

// printinput();

}

void printoutput()

{

int TTAT=0,TWT=0;

float ATAT,AWT;

printf("\n\*\*\*\*\*\*Final Table\*\*\*\*\*");

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

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

{

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

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

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

}

ATAT=(float)TTAT/n;

AWT=(float)TWT/n;

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

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

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

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

}

void printganttchart()

{

int j=0;

g1[0]=g[0];

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

{

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

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

else

{

j++;

g1[j]=g[i];

}

}

printf("\n\*\*\*\*\*\*Each unit Gantt chart\*\*\*\*\*\*");

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

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

printf("\n\*\*\*\*\*\*Final Gantt chart\*\*\*\*\*\*");

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();

printf("\nEntered data is: ");

printinput();

sort();

printf("\nData after Sorting" );

printinput();

processinput();

printoutput();

printganttchart();

}

**Slip 16**

**Write a programto implement the toy shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.**

**count c filename :- To print number of characters in the file.**

**count w filename :- To print number of words in the file.**

#include<stdio.h>

char \*buff,\*t1,\*t2,\*t3,ch;

FILE \*fp;

int pid;

void count(char \*t2,char \*t3)

{

int charcount=0,wordcount=0,linecount=0;

fp=fopen(t3,"r");

if(fp==NULL)

printf("\nError in opening the file");

else

{

while((ch=fgetc(fp))!=EOF)

{

if(ch==' ')

wordcount++;

else if(ch=='\n')

{

linecount++;

wordcount++;

}

else

charcount++;

}

}

fclose(fp);

if(strcmp(t2,"c")==0)

printf("\nTotal no of characters: %d",charcount);

else if(strcmp(t2,"w")==0)

printf("\nTotal no of Words: %d",wordcount);

else if(strcmp(t2,"l")==0)

printf("\nTotal no of lines: %d",linecount);

else

printf("Command not found");

}

main()

{

while(1)

{

printf("\nMyshell$");

buff=(char\*)malloc(80);

t1=(char\*)malloc(80);

t2=(char\*)malloc(80);

t3=(char\*)malloc(80);

fgets(buff,80,stdin);

sscanf(buff,"%s %s %s",t1,t2,t3);

if(strcmp(t1, "pause")==0)

exit(0);

if(strcmp(t1,"count")==0)

count(t2,t3);

else

{

pid=fork();

if(pid<0)

printf("\nchild Process is not craeted\n");

else if(pid==0)

{

execlp("/bin",NULL);

system(buff);

}

else

{

wait(NULL);

exit(0);

}

}

}

}

**Write the program to simulate Non preemptive priority scheduling. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.**

#include<stdio.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 Process: ");

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("\nEnter the Priority:");

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

}

}

void printinput()

{

// int TWT=0,TTAT=0;

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

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

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

}

void printoutput()

{

int TWT=0,TTAT=0;

float ATAT,AWT;

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

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

{

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].fttab[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++)//i=0,1

{ // p1,p3 min=2

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;

}

// i++;

}

}

void ganttchart()

{

int i,j=0;

printf("\n\*\*\*\*\*\*Each Unit Gantt chart\*\*\*\*\*\*");

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

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

{

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

}

printf("\n\*\*\*\*\*\*\*\*Final Gantt Chart\*\*\*\*\*\*\*");

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();

printf("\nData After Sorting: ");

printinput();

processinput();

printoutput();

ganttchart();

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

{

tab[i].tbt=tab[i].bt=rand()%10+1;

tab[i].at=tab[i].ft+2;

}

printinput();

processinput();

printoutput();

ganttchart();

}

**Slip 17**

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the Optimal page replacement algorithm. Assume the memory of n frames. Reference String : 7, 5, 4, 8, 5, 7, 2, 3, 1, 3, 5, 9, 4, 6,**

#include<stdio.h>

int nor,nof,refstring[30],F[10];

void accept()

{

int i;

printf("\nEnter the Reference String:\n ");

for(i=0;i<nor;i++)

{

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

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

}

}

int search(int page)

{

int i;

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

{

if(page==F[i])

return i;

}

return -1;

}

int getfno(int i)

{

int fno,prev,pos=0,fpos,flag;

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

{

flag=0;

for(prev=i+1;prev<nor;prev++)

{

if(F[fno]==refstring[prev])

{

flag=1;

if(prev>pos) //2<3

{

pos=prev; //pos=2

fpos=fno; //fpos=2

}

break;

}

}

if(flag==0)

{

fpos=fno;

break;

}

}

//printf("\nfpos=%d",fpos);

return fpos;

}

void optimal()

{

int i,j,k,fno,fault=0;

for(fno=0,i=0;fno<nof && i<nor;i++)

{

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

k=search(refstring[i]);

//printf("\nk=%d",k);

if(k==-1)

{

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

fno++;

}

}

while(i<nor)

{

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

k=search(refstring[i]);

//printf("\nk=%d",k);

if(k==-1)

{

fno=getfno(i);

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

}

i++;

}

printf("\nTotal no of Page fault: %d",fault);

}

main()

{

printf("\nEnter the Length of the string: ");

scanf("%d",&nor);

printf("\nEnter no. of Frames: ");

scanf("%d",&nof);

accept();

optimal();

}

**Write the program to simulate FCFS CPU-scheduling. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.**

#include<stdio.h>

struct input

{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

int start,end;

char pname[10];

}g[50],g1[10];

int n,i,k,time,prev;

void getinput()

{

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

scanf("%d",&n);

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

{

printf("\nProcess Name: ");

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

printf("Burst Time: ");

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

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

printf("Arrival Time:" );

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

}

}

void printinput()

{

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

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

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

}

void sort()

{

struct input temp;

int j;

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

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

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

{

temp=tab[j];

tab[j]=tab[j+1];

tab[j+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;

}

void processinput()

{

int j,finish=0;

// time=tab[0].at;

while(finish!=n)

{

if(arrived(time))

{

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

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

// printinput();

prev=time;

tab[i].ft=time;

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

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

{

finish++;

break;

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

prev=time;

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

}

}

// printinput();

}

void printoutput()

{

int TTAT=0,TWT=0;

float ATAT,AWT;

printf("\n\*\*\*\*\*\*Final Table\*\*\*\*\*");

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

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

{

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

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

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

}

ATAT=(float)TTAT/n;

AWT=(float)TWT/n;

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

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

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

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

}

void printganttchart()

{

int j=0;

g1[0]=g[0];

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

{

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

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

else

{

j++;

g1[j]=g[i];

}

}

printf("\n\*\*\*\*\*\*Each unit Gantt chart\*\*\*\*\*\*");

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

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

printf("\n\*\*\*\*\*\*Final Gantt chart\*\*\*\*\*\*");

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();

printf("\nEntered data is: ");

printinput();

sort();

printf("\nData after Sorting" );

printinput();

processinput();

printoutput();

printganttchart();

}

**Slip 18**

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the LRU page replacement algorithm. Assume the memory of n frames. Reference String : 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6**

#include<stdio.h>

int nor,nof,refstring[30],F[10];

void accept()

{

int i;

printf("\nEnter the Reference String:\n ");

for(i=0;i<nor;i++)

{

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

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

}

}

int search(int page)

{

int i;

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

{

if(page==F[i])

return i;

}

return -1;

}

int getfno(int i)

{

int fno,prev,pos=99,fpos;

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

{

for(prev=i-1;prev>=0;prev--)

{

if(F[fno]==refstring[prev])

{

if(prev<pos)

{

pos=prev;

fpos=fno;

}

break;

}

}

}

;

return fpos;

}

void LRU()

{

int i,j,k,fno,fault=0;

for(fno=0,i=0;fno<nof && i<nor;i++)

{

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

k=search(refstring[i]);

if(k==-1)

{

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

fno++;

}

}

while(i<nor)

{

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

k=search(refstring[i]);

if(k==-1)

{

fno=getfno(i);

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

}

i++;

}

printf("\nTotal no of Page fault: %d",fault);

}

main()

{

printf("\nEnter the Length of the string: ");

scanf("%d",&nor);

printf("\nEnter no. of Frames: ");

scanf("%d",&nof);

accept();

LRU();

}

**Write a C program to simulate FCFS CPU-scheduling. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.**

#include<stdio.h>

struct input

{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

int start,end;

char pname[10];

}g[50],g1[10];

int n,i,k,time,prev;

void getinput()

{

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

scanf("%d",&n);

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

{

printf("\nProcess Name: ");

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

printf("Burst Time: ");

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

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

printf("Arrival Time:" );

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

}

}

void printinput()

{

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

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

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

}

void sort()

{

struct input temp;

int j;

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

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

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

{

temp=tab[j];

tab[j]=tab[j+1];

tab[j+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;

}

void processinput()

{

int j,finish=0;

// time=tab[0].at;

while(finish!=n)

{

if(arrived(time))

{

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

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

// printinput();

prev=time;

tab[i].ft=time;

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

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

{

finish++;

break;

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

prev=time;

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

}

}

// printinput();

}

void printoutput()

{

int TTAT=0,TWT=0;

float ATAT,AWT;

printf("\n\*\*\*\*\*\*Final Table\*\*\*\*\*");

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

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

{

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

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

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

}

ATAT=(float)TTAT/n;

AWT=(float)TWT/n;

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

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

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

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

}

void printganttchart()

{

int j=0;

g1[0]=g[0];

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

{

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

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

else

{

j++;

g1[j]=g[i];

}

}

printf("\n\*\*\*\*\*\*Each unit Gantt chart\*\*\*\*\*\*");

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

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

printf("\n\*\*\*\*\*\*Final Gantt chart\*\*\*\*\*\*");

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();

printf("\nEntered data is: ");

printinput();

sort();

printf("\nData after Sorting" );

printinput();

processinput();

printoutput();

printganttchart();

}

**Slip 19**

**Write a C program to implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following ‘list’ commands as**

**myshell$ list f dirname :- To print names of all the files in current directory.**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<dirent.h>

char \*buff,\*t1,\*t2,\*t3,ch;

int pid;

void list(char t2,char \*t3)

{

DIR \*dir;

struct dirent \*entry;

int cnt=0;

dir=opendir(t3);

if (dir==NULL)

{

printf("Directory %s not found",t3);

return;

}

switch(t2)

{

case 'f' : while((entry=readdir(dir))!=NULL)

{

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)

{

printf("\n%s\t %d",entry->d\_name,entry->d\_ino);

}

break;

default : printf("Invalid argument");

}

closedir(dir);

}

main()

{

while(1)

{

printf("myshell$");

fflush(stdin);

t1=(char \*)malloc(80);

t2=(char \*)malloc(80);

t3=(char \*)malloc(80);

buff=(char \*)malloc(80);

fgets(buff,80,stdin);

sscanf(buff,"%s %s %s",t1,t2,t3);

if(strcmp(t1,"pause")==0)

exit(0);

else if(strcmp(t1,"list")==0)

list(t2[0],t3);

else

{

pid=fork();

if(pid<0)

printf("Child process is not created\n");

else if(pid==0)

{

execlp("/bin",NULL);

if(strcmp(t1,"exit")==0)

exit(0);

system(buff);

}

else

{

wait(NULL);

exit(0);

}

}

}

}

**Write the simulation program for Round Robin scheduling for given time quantum. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give the Gantt chart, turnaround time and waiting time for each process. Also display the average turnaround time and average waiting time.**

#include<stdio.h>

struct input

{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

char pname[10];

int start,end;

}g[30],g1[30];

int n,time,prev,k,tq;

void getinput()

{

int i;

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

scanf("%d",&n);

printf("\nEnter Time quantum: ");

scanf("%d",&tq);

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;

}

}

void printinput()

{

// int TWT=0,TTAT=0;

int i;

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

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

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

}

void printoutput()

{

int TWT=0,TTAT=0,i;

float ATAT,AWT;

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

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

{

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

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)

{

int i;

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

{

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

return 1;

}

return 0;

}

void processinput()

{

int finish=0,j;

int i=0;

k=0;

while(finish!=n)

{

if(arrived(time))

{

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

{

for(j=0;j<tq;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;

}

if(time<tab[(i+1)%n].at)

i=0;

else

i=(i+1)%n;

}

}

void ganttchart()

{

int i,j=0;

printf("\n\*\*\*\*\*\*Each Unit Gantt chart\*\*\*\*\*\*");

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

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

{

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

}

printf("\n\*\*\*\*\*\*\*\*Final Gantt Chart\*\*\*\*\*\*\*");

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();

printf("\nData After Sorting: ");

printinput();

processinput();

printoutput();

ganttchart();

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

{

tab[i].tbt=tab[i].bt=rand()%10+1;

tab[i].at=tab[i].ft+2;

}

printinput();

processinput();

printoutput();

ganttchart();

}

**Slip 20**

**Write a C program to implement the shell which displays the command prompt “myshell$”. It accepts the command, tokenize the command line and execute it by creating the child process. Also implement the additional command ‘typeline’ as**

**typeline -a filename :- To print all lines in the file.**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h>

char \*buff,\*t1,\*t2,\*t3,ch;

FILE \*fp;

int pid;

void typeline(char \*t2,char \*t3)

{

int i,n,count=0,num;

if((fp=fopen(t3,"r"))==NULL)

printf("File not found\n");

if(strcmp(t2,"a")==0)

{

while((ch=fgetc(fp))!=EOF)

printf("%c",ch);

fclose(fp);

return;

}

n=atoi(t2);

if(n>0)

{

i=0;

while((ch=fgetc(fp))!=EOF)

{

if(ch=='\n')

i++;

if(i==n)

break;

printf("%c",ch);

}

printf("\n");

}

else

{

count=0;

while((ch=fgetc(fp))!=EOF)

if(ch=='\n')

count++;

fseek(fp,0,SEEK\_SET);

i=0;

while((ch=fgetc(fp))!=EOF)

{

if(ch=='\n')

i++;

if(i==count+n-1)

break;

}

while((ch=fgetc(fp))!=EOF)

printf("%c",ch);

}

fclose(fp);

}

main()

{

while(1)

{

printf("myshell$");

fflush(stdin);

t1=(char \*)malloc(80);

t2=(char \*)malloc(80);

t3=(char \*)malloc(80);

buff=(char \*)malloc(80);

fgets(buff,80,stdin);

sscanf(buff,"%s %s %s",t1,t2,t3);

if(strcmp(t1,"pause")==0)

exit(0);

else if(strcmp(t1,"typeline")==0)

typeline(t2,t3);

else

{

pid=fork();

if(pid<0)

printf("Child process is not created\n");

else if(pid==0)

{

execlp("/bin",NULL);

if(strcmp(t1,"exit")==0)

exit(0);

system(buff);

}

else

{

wait(NULL);

exit(0);

}

}

}

}

**Write the program to simulate Non-preemptive Shortest Job First (SJF) – scheduling. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time**

#include<stdio.h>

struct input

{

char pname[10];

int bt,at,tbt,ft;

}tab[10];

struct gantt

{

int start,end;

char pname[10];

}g[50],g1[10];

int n,i,k,time,prev;

void getinput()

{

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

scanf("%d",&n);

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

{

printf("\nProcess Name: ");

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

printf("Burst Time: ");

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

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

printf("Arrival Time:" );

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

}

}

void printinput()

{

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

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

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

}

void sort()

{

struct input temp;

int j;

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

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

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

{

temp=tab[j];

tab[j]=tab[j+1];

tab[j+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 getsmallburst(int time)

{

int min=99,mini;

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

{

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

{

min=tab[i].tbt;

mini=i;

}

}

return mini;

}

void processinput()

{

int j,finish=0;

// time=tab[0].at;

while(finish!=n)

{

if(arrived(time))

{

i=getsmallburst(time);

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

{

time++;

tab[i].tbt--;

g[k].start=prev;

g[k].end=time;

// printinput();

prev=time;

tab[i].ft=time;

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

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

{

finish++;

break;

}

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

prev=time;

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

}

}

// printinput();

}

void printoutput()

{

int TTAT=0,TWT=0;

float ATAT,AWT;

printf("\n\*\*\*\*\*\*Final Table\*\*\*\*\*");

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

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

{

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

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

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

}

ATAT=(float)TTAT/n;

AWT=(float)TWT/n;

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

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

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

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

}

void printganttchart()

{

int j=0;

g1[0]=g[0];

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

{

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

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

else

{

j++;

g1[j]=g[i];

}

}

printf("\n\*\*\*\*\*\*Each unit Gantt chart\*\*\*\*\*\*");

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

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

printf("\n\*\*\*\*\*\*Final Gantt chart\*\*\*\*\*\*");

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();

printf("\nEntered data is: ");

printinput();

sort();

printf("\nData after Sorting" );

printinput();

processinput();

printoutput();

printganttchart();

}

**Slip 21**

**Write a C Program to create a child process using fork (), display parent and child process id. Child process will display the message “I am Child Process”and the parent process should display “I am Parent Process”.**

#include<stdio.h>

void childprocess()

{

printf("\n I am Child Process");

}

void parentprocess()

{

printf("\n I am Parent Process");

}

main()

{

int pid;

pid=fork();

if(pid==0)

{

printf("Child Process ID=%d",pid);

childprocess();

}

else

{

printf("\n Parent Process Id=%d",pid);

parentprocess();

}

}

**Write a C program to simulate Preemptive Priority scheduling. The arrival time and first CPU-burst and priority for different n number of processes should be input to the algorithm. Assume the fixed IO waiting time (2 units). The next CPU-burst should be generated randomly. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.**

#include<stdio.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 Process: ");

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("\nEnter the Priority:");

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

}

}

void printinput()

{

// int TWT=0,TTAT=0;

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

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

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

}

void printoutput()

{

int TWT=0,TTAT=0;

float ATAT,AWT;

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

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

{

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].fttab[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++)//i=0,1

{ // p1,p3 min=2

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);

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++;

}

}

else

{

time++;

g[k].start=prev;

g[k].end=time;

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

prev=time;

}

// i++;

}

}

void ganttchart()

{

int i,j=0;

printf("\n\*\*\*\*\*\*Each Unit Gantt chart\*\*\*\*\*\*");

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

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

{

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

}

printf("\n\*\*\*\*\*\*\*\*Final Gantt Chart\*\*\*\*\*\*\*");

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();

printf("\nData After Sorting: ");

printinput();

processinput();

printoutput();

ganttchart();

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

{

tab[i].tbt=tab[i].bt=rand()%10+1;

tab[i].at=tab[i].ft+2;

}

printinput();

processinput();

printoutput();

ganttchart();

}

**Slip 22**

**Write a C program that demonstrates the use of nice() system call. After a child Process is started using fork (), assign higher priority to the child usingnice () system call.**

#include<stdio.h>

main()

{

int pid,retnice,i;

pid=fork();

for(i=0;i<3;i++)

{

if(pid==0)

{

retnice=nice(-5);

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

sleep(1);

}

else

{

retnice=nice(4);

printf("Parent gets lower CPU Priority%d\n",retnice);

sleep(1);

}

}

}

**Write a C program to simulate Non preemptive priority scheduling. The arrival time and first CPU-burst of different jobs should be input to the system. Accept no. of Processes, arrival time and burst time. The output should give Gantt chart, turnaround time and waiting time for each process. Also find the average waiting time and turnaround time.**

#include<stdio.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 Process: ");

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("\nEnter the Priority:");

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

}

}

void printinput()

{

// int TWT=0,TTAT=0;

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

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

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

}

void printoutput()

{

int TWT=0,TTAT=0;

float ATAT,AWT;

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

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

{

printf("\n%s\t%d\t%d\t%d\t%d\t%d",tab[i].pname,tab[i].at,tab[i].bt,tab[i].ft,tab[i].fttab[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++)//i=0,1

{ // p1,p3 min=2

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;

}

// i++;

}

}

void ganttchart()

{

int i,j=0;

printf("\n\*\*\*\*\*\*Each Unit Gantt chart\*\*\*\*\*\*");

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

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

{

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

}

printf("\n\*\*\*\*\*\*\*\*Final Gantt Chart\*\*\*\*\*\*\*");

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();

printf("\nData After Sorting: ");

printinput();

processinput();

printoutput();

ganttchart();

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

{

tab[i].tbt=tab[i].bt=rand()%10+1;

tab[i].at=tab[i].ft+2;

}

printinput();

processinput();

printoutput();

ganttchart();

}

**Slip 23**

**Write a C program to illustrate the concept of orphan process. Parent process creates a child and terminates before child has finished its task. So child process becomes orphan process. (Use fork(), sleep(), getpid(), getppid()).**

#include<stdio.h>

main()

{

int pid,a;

printf("\n the process id is%d",getpid());

pid=fork();

if(pid<0)

printf("\n Fork failed!");

else if(pid==0)

{

printf("\n I am Child process");

printf("\n Child process ID=%d",getpid());

}

else

{

wait(NULL);

sleep(5);

printf("\n I am Parent process");

printf("\n Child Parent ID=%d",getpid());

}

}

**Write the simulation program for demand paging and show the page scheduling and total number of page faults according the Optimal page replacement algorithm. Assume the memory of n frames. Reference String : 7, 5, 4, 8, 5, 7, 2, 3, 1, 3, 5, 9, 4, 6,**

#include<stdio.h>

int nor,nof,refstring[30],F[10];

void accept()

{

int i;

printf("\nEnter the Reference String:\n ");

for(i=0;i<nor;i++)

{

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

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

}

}

int search(int page)

{

int i;

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

{

if(page==F[i])

return i;

}

return -1;

}

int getfno(int i)

{

int fno,prev,pos=0,fpos,flag;

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

{

flag=0;

for(prev=i+1;prev<nor;prev++)

{

if(F[fno]==refstring[prev])

{

flag=1;

if(prev>pos) //2<3

{

pos=prev; //pos=2

fpos=fno; //fpos=2

}

break;

}

}

if(flag==0)

{

fpos=fno;

break;

}

}

//printf("\nfpos=%d",fpos);

return fpos;

}

void optimal()

{

int i,j,k,fno,fault=0;

for(fno=0,i=0;fno<nof && i<nor;i++)

{

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

k=search(refstring[i]);

//printf("\nk=%d",k);

if(k==-1)

{

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

fno++;

}

}

while(i<nor)

{

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

k=search(refstring[i]);

//printf("\nk=%d",k);

if(k==-1)

{

fno=getfno(i);

F[fno]=refstring[i];

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

{

if(F[j])

printf("\t%d",F[j]);

}

fault++;

}

i++;

}

printf("\nTotal no of Page fault: %d",fault);

}

main()

{

printf("\nEnter the Length of the string: ");

scanf("%d",&nor);

printf("\nEnter no. of Frames: ");

scanf("%d",&nof);

accept();

optimal();

}

**Slip 24**

**Write a C program to accept n integers to be sorted. Main function creates child process using fork system call. Parent process sorts the integers using bubble sort and waits for child process using wait system call. Child process sorts the integers using insertion sort.**

#include<stdio.h>

int n;

void display(int a[20])

{

int i;

printf("\n Array Element:");

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

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

}

void insertionsort(int a[20])

{

int i,key,j;

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

{

key=a[i];

for(j=i-1;j>=0 && a[j]>key;j--)

{

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

}

a[j+1]=key;

}

}

void bubblesort(int a[20])

{

int pass,i,temp;

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

{

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

{

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

{

temp=a[i];

a[i]=a[i+1];

a[i+1]=temp;

}

}

}

}

main()

{

int i,a[20],pid,Barr[20],Iarr[20];

printf("\n Enter value of n");

scanf("%d",&n);

printf("\n Enter arrray Elements");

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

{

printf("\n Enter Element:",i);

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

Barr[i]=a[i];

Iarr[i]=a[i];

}

display(a);

pid=fork();

if(pid==0)

{

printf("\n Child Process id=%d",getpid());

insertionsort(Iarr);

printf("\n Insertion Sort");

display(Iarr);

}

else if(pid>0)

{

wait(NULL);

sleep(5);

printf("\n Parent Process id=%d",getpid());

bubblesort(Barr);

printf("\nBubbleSort");

display(Barr);

}

}

**Write a C program to implement the toy shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands.**

**count c filename :- To print number of characters in the file.**

**count w filename :- To print number of words in the file.**

**count l filename :- To print number of lines in the file.**

#include<stdio.h>

char \*buff,\*t1,\*t2,\*t3,ch;

FILE \*fp;

int pid;

void count(char \*t2,char \*t3)

{

int charcount=0,wordcount=0,linecount=0;

fp=fopen(t3,"r");

if(fp==NULL)

printf("\nError in opening the file");

else

{

while((ch=fgetc(fp))!=EOF)

{

if(ch==' ')

wordcount++;

else if(ch=='\n')

{

linecount++;

wordcount++;

}

else

charcount++;

}

}

fclose(fp);

if(strcmp(t2,"c")==0)

printf("\nTotal no of characters: %d",charcount);

else if(strcmp(t2,"w")==0)

printf("\nTotal no of Words: %d",wordcount);

else if(strcmp(t2,"l")==0)

printf("\nTotal no of lines: %d",linecount);

else

printf("Command not found");

}

main()

{

while(1)

{

printf("\nMyshell$");

buff=(char\*)malloc(80);

t1=(char\*)malloc(80);

t2=(char\*)malloc(80);

t3=(char\*)malloc(80);

fgets(buff,80,stdin);

sscanf(buff,"%s %s %s",t1,t2,t3);

if(strcmp(t1, "pause")==0)

exit(0);

if(strcmp(t1,"count")==0)

count(t2,t3);

else

{

pid=fork();

if(pid<0)

printf("\nchild Process is not craeted\n");

else if(pid==0)

{

execlp("/bin",NULL);

system(buff);

}

else

{

wait(NULL);

exit(0);

}

}

}

}

**Slip 25**

**Write a C program that accepts an integer array. Main function forks child process. Parent process sorts an integer array and passes the sorted array to child process through the command line arguments of execve() system call. The child process uses execve() system call to load new program that uses this sorted array for performing the binary search to search the particular item in the array.**

**Write a programto implement the shell. It should display the command prompt “myshell$”. Tokenize the command line and execute the given command by creating the child process. Additionally it should interpret the following commands. myshell$ search f filename pattern :- To display first occurrence of pattern in the file.**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h>

char \*buff,\*t1,\*t2,\*t3,\*t4,ch;

FILE \*fp;

int pid;

void search(char \*t2,char \*t3,char \*t4)

{

int i=1,count=0;

char \*p;

if((fp=fopen(t3,"r"))==NULL)

printf("File not found\n");

else

{

if(strcmp(t2,"f")==0)

{

while(fgets(buff,80,fp))

{

if((strstr(buff,t4))!=NULL)

{

printf("%d: %s\n",i,buff);

break;

}

}

i++;

}

else if(strcmp(t2,"c")==0)

{

while(fgets(buff,80,fp))

{

if((strstr(buff,t4))!=NULL)

{

count++;

}

}

printf("No of occurences of %s= %d\n",t3,count);

}

else if(strcmp(t2,"a")==0)

{

while(fgets(buff,80,fp))

{

if((strstr(buff,t4))!=NULL)

{

printf("%d: %s\n",i,buff);

}

i++;

}

}

else

printf("Command not found\n");

fclose(fp);

}

}

main()

{

while(1)

{

printf("myshell$");

fflush(stdin);

t1=(char \*)malloc(80);

t2=(char \*)malloc(80);

t3=(char \*)malloc(80);

t4=(char \*)malloc(80);

buff=(char \*)malloc(80);

fgets(buff,80,stdin);

sscanf(buff,"%s %s %s %s",t1,t2,t3,t4);

if(strcmp(t1,"pause")==0)

exit(0);

else if(strcmp(t1,"search")==0)

search(t2,t3,t4);

else

{

pid=fork();

if(pid<0)

printf("Child process is not created\n");

else if(pid==0)

{

execlp("/bin",NULL);

if(strcmp(t1,"exit")==0)

exit(0);

system(buff);

}

else

{

wait(NULL);

exit(0);

}

}

}

}