**SLIP NO- 1**

**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>

int n,nf;

int in[100];

int p[50];

int hit=0;

int i,j,k;

int pgfaultcnt=0;

void getData()

{

printf("\nEnter length of page reference sequence:");

scanf("%d",&n);

printf("\nEnter the page reference sequence:");

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

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

printf("\nEnter no of frames:");

scanf("%d",&nf);

}

void initialize()

{

pgfaultcnt=0;

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

p[i]=9999;

}

int isHit(int data)

{

hit=0;

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

{

if(p[j]==data)

{

hit=1;

break;

}

}

return hit;

}

int getHitIndex(int data)

{

int hitind;

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

{

if(p[k]==data)

{

hitind=k;

break;

}

}

return hitind;

}

void dispPages()

{

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

{

if(p[k]!=9999)

printf(" %d",p[k]);

}

}

void dispPgFaultCnt()

{

printf("\nTotal no of page faults:%d",pgfaultcnt);

}

void lfu()

{

int usedcnt[100];

int least,repin,sofarcnt=0,bn;

initialize();

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

usedcnt[i]=0;

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

{

printf("\n For %d :",in[i]);

if(isHit(in[i]))

{

int hitind=getHitIndex(in[i]);

usedcnt[hitind]++;

printf("No page fault!");

}

else

{

pgfaultcnt++;

if(bn<nf)

{

p[bn]=in[i];

usedcnt[bn]=usedcnt[bn]+1;

bn++;

}

else

{

least=9999;

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

if(usedcnt[k]<least)

{

least=usedcnt[k];

repin=k;

}

p[repin]=in[i];

sofarcnt=0;

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

if(in[i]==in[k])

sofarcnt=sofarcnt+1;

usedcnt[repin]=sofarcnt;

}

dispPages();

}

}

dispPgFaultCnt();

}

int main()

{

getData();

lfu();

dispPgFaultCnt();

return 0;

**2.** 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>**

**#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 NO 2**

1. 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>**

**int n,nf;**

**int in[100];**

**int p[50];**

**int hit=0;**

**int i,j,k;**

**int pgfaultcnt=0;**

**void getData()**

**{**

**printf("\nEnter length of page reference sequence:");**

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

**printf("\nEnter the page reference sequence:");**

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

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

**printf("\nEnter no of frames:");**

**scanf("%d",&nf);**

**}**

**void initialize()**

**{**

**pgfaultcnt=0;**

**for(i=0; i<nf; i++)**

**p[i]=9999;**

**}**

**int isHit(int data)**

**{**

**hit=0;**

**for(j=0; j<nf; j++)**

**{**

**if(p[j]==data)**

**{**

**hit=1;**

**break;**

**}**

**}**

**return hit;**

**}**

**int getHitIndex(int data)**

**{**

**int hitind;**

**for(k=0; k<nf; k++)**

**{**

**if(p[k]==data)**

**{**

**hitind=k;**

**break;**

**}**

**}**

**return hitind;**

**}**

**void dispPages()**

**{**

**for (k=0; k<nf; k++)**

**{**

**if(p[k]!=9999)**

**printf(" %d",p[k]);**

**}**

**}**

**void dispPgFaultCnt()**

**{**

**printf("\nTotal no of page faults:%d",pgfaultcnt);**

**}**

**void lru()**

**{**

**initialize();**

**int least[50];**

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

**{**

**printf("\nFor %d :",in[i]);**

**if(isHit(in[i])==0)**

**{**

**for(j=0; j<nf; j++)**

**{**

**int pg=p[j];**

**int found=0;**

**for(k=i-1; k>=0; k--)**

**{**

**if(pg==in[k])**

**{**

**least[j]=k;**

**found=1;**

**break;**

**}**

**else**

**found=0;**

**}**

**if(!found)**

**least[j]=-9999;**

**}**

**int min=9999;**

**int repindex;**

**for(j=0; j<nf; j++)**

**{**

**if(least[j]<min)**

**{**

**min=least[j];**

**repindex=j;**

**}**

**}**

**p[repindex]=in[i];**

**pgfaultcnt++;**

**dispPages();**

**}**

**else**

**printf("No page fault!");**

**}**

**dispPgFaultCnt();**

**}**

**int main()**

**{**

**getData();**

**lru();**

**dispPgFaultCnt();**

**return 0;**

**}**

**2**.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 NO 3**

**1.**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>**

**int n,nf;**

**int in[100];**

**int p[50];**

**int hit=0;**

**int i,j,k;**

**int pgfaultcnt=0;**

**void getData()**

**{**

**printf("\nEnter length of page reference sequence:");**

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

**printf("\nEnter the page reference sequence:");**

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

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

**printf("\nEnter no of frames:");**

**scanf("%d",&nf);**

**}**

**void initialize()**

**{**

**pgfaultcnt=0;**

**for(i=0; i<nf; i++)**

**p[i]=9999;**

**}**

**int isHit(int data)**

**{**

**hit=0;**

**for(j=0; j<nf; j++)**

**{**

**if(p[j]==data)**

**{**

**hit=1;**

**break;**

**}**

**}**

**return hit;**

**}**

**int getHitIndex(int data)**

**{**

**int hitind;**

**for(k=0; k<nf; k++)**

**{**

**if(p[k]==data)**

**{**

**hitind=k;**

**break;**

**}**

**}**

**return hitind;**

**}**

**void dispPages()**

**{**

**for (k=0; k<nf; k++)**

**{**

**if(p[k]!=9999)**

**printf(" %d",p[k]);**

**}**

**}**

**void dispPgFaultCnt()**

**{**

**printf("\nTotal no of page faults:%d",pgfaultcnt);**

**}**

**void lru()**

**{**

**initialize();**

**int least[50];**

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

**{**

**printf("\nFor %d :",in[i]);**

**if(isHit(in[i])==0)**

**{**

**for(j=0; j<nf; j++)**

**{**

**int pg=p[j];**

**int found=0;**

**for(k=i-1; k>=0; k--)**

**{**

**if(pg==in[k])**

**{**

**least[j]=k;**

**found=1;**

**break;**

**}**

**else**

**found=0;**

**}**

**if(!found)**

**least[j]=-9999;**

**}**

**int min=9999;**

**int repindex;**

**for(j=0; j<nf; j++)**

**{**

**if(least[j]<min)**

**{**

**min=least[j];**

**repindex=j;**

**}**

**}**

**p[repindex]=in[i];**

**pgfaultcnt++;**

**dispPages();**

**}**

**else**

**printf("No page fault!");**

**}**

**dispPgFaultCnt();**

**}**

**int main()**

**{**

**getData();**

**lru();**

**dispPgFaultCnt();**

**return 0;**

**}**

**2.**Write a 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 <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 NO 4**

**1.**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

1. 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>

#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 NO 5

**1.**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 n,nf;

int in[100];

int p[50];

int hit=0;

int i,j,k;

int pgfaultcnt=0;

void getData()

{

printf("\nEnter length of page reference sequence:");

scanf("%d",&n);

printf("\nEnter the page reference sequence:");

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

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

printf("\nEnter no of frames:");

scanf("%d",&nf);

}

void initialize()

{

pgfaultcnt=0;

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

p[i]=9999;

}

int isHit(int data)

{

hit=0;

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

{

if(p[j]==data)

{

hit=1;

break;

}

}

return hit;

}

int getHitIndex(int data)

{

int hitind;

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

{

if(p[k]==data)

{

hitind=k;

break;

}

}

return hitind;

}

void dispPages()

{

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

{

if(p[k]!=9999)

printf(" %d",p[k]);

}

}

void dispPgFaultCnt()

{

printf("\nTotal no of page faults:%d",pgfaultcnt);

}

void optimal()

{

initialize();

int near[50];

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

{

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

if(isHit(in[i])==0)

{

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

{

int pg=p[j];

int found=0;

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

{

if(pg==in[k])

{

near[j]=k;

found=1;

break;

}

else

found=0;

}

if(!found)

near[j]=9999;

}

int max=-9999;

int repindex;

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

{

if(near[j]>max)

{

max=near[j];

repindex=j;

}

}

p[repindex]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault");

}

dispPgFaultCnt();

}

int main()

{

getData();

optimal();

dispPgFaultCnt();

return 0;

}

2.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>

#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 NO 6

1.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

2.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.

#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 NO 7

1.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 n,nf;

int in[100];

int p[50];

int hit=0;

int i,j,k;

int pgfaultcnt=0;

void getData()

{

printf("\nEnter length of page reference sequence:");

scanf("%d",&n);

printf("\nEnter the page reference sequence:");

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

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

printf("\nEnter no of frames:");

scanf("%d",&nf);

}

void initialize()

{

pgfaultcnt=0;

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

p[i]=9999;

}

int isHit(int data)

{

hit=0;

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

{

if(p[j]==data)

{

hit=1;

break;

}

}

return hit;

}

int getHitIndex(int data)

{

int hitind;

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

{

if(p[k]==data)

{

hitind=k;

break;

}

}

return hitind;

}

void dispPages()

{

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

{

if(p[k]!=9999)

printf(" %d",p[k]);

}

}

void dispPgFaultCnt()

{

printf("\nTotal no of page faults:%d",pgfaultcnt);

}

void optimal()

{

initialize();

int near[50];

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

{

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

if(isHit(in[i])==0)

{

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

{

int pg=p[j];

int found=0;

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

{

if(pg==in[k])

{

near[j]=k;

found=1;

break;

}

else

found=0;

}

if(!found)

near[j]=9999;

}

int max=-9999;

int repindex;

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

{

if(near[j]>max)

{

max=near[j];

repindex=j;

}

}

p[repindex]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault");

}

dispPgFaultCnt();

}

int main()

{

getData();

optimal();

dispPgFaultCnt();

return 0;

}

2.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>

#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 NO 8

1. 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 n,nf;

int in[100];

int p[50];

int hit=0;

int i,j,k;

int pgfaultcnt=0;

void getData()

{

printf("\nEnter length of page reference sequence:");

scanf("%d",&n);

printf("\nEnter the page reference sequence:");

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

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

printf("\nEnter no of frames:");

scanf("%d",&nf);

}

void initialize()

{

pgfaultcnt=0;

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

p[i]=9999;

}

int isHit(int data)

{

hit=0;

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

{

if(p[j]==data)

{

hit=1;

break;

}

}

return hit;

}

int getHitIndex(int data)

{

int hitind;

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

{

if(p[k]==data)

{

hitind=k;

break;

}

}

return hitind;

}

void dispPages()

{

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

{

if(p[k]!=9999)

printf(" %d",p[k]);

}

}

void dispPgFaultCnt()

{

printf("\nTotal no of page faults:%d",pgfaultcnt);

}

void lru()

{

initialize();

int least[50];

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

{

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

if(isHit(in[i])==0)

{

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

{

int pg=p[j];

int found=0;

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

{

if(pg==in[k])

{

least[j]=k;

found=1;

break;

}

else

found=0;

}

if(!found)

least[j]=-9999;

}

int min=9999;

int repindex;

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

{

if(least[j]<min)

{

min=least[j];

repindex=j;

}

}

p[repindex]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault!");

}

dispPgFaultCnt();

}

int main()

{

getData();

lru();

dispPgFaultCnt();

return 0;

}

1. 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

#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 NO 9

1.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 n,nf;

int in[100];

int p[50];

int hit=0;

int i,j,k;

int pgfaultcnt=0;

void getData()

{

printf("\nEnter length of page reference sequence:");

scanf("%d",&n);

printf("\nEnter the page reference sequence:");

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

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

printf("\nEnter no of frames:");

scanf("%d",&nf);

}

void initialize()

{

pgfaultcnt=0;

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

p[i]=9999;

}

int isHit(int data)

{

hit=0;

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

{

if(p[j]==data)

{

hit=1;

break;

}

}

return hit;

}

int getHitIndex(int data)

{

int hitind;

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

{

if(p[k]==data)

{

hitind=k;

break;

}

}

return hitind;

}

void dispPages()

{

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

{

if(p[k]!=9999)

printf(" %d",p[k]);

}

}

void dispPgFaultCnt()

{

printf("\nTotal no of page faults:%d",pgfaultcnt);

}

void lru()

{

initialize();

int least[50];

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

{

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

if(isHit(in[i])==0)

{

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

{

int pg=p[j];

int found=0;

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

{

if(pg==in[k])

{

least[j]=k;

found=1;

break;

}

else

found=0;

}

if(!found)

least[j]=-9999;

}

int min=9999;

int repindex;

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

{

if(least[j]<min)

{

min=least[j];

repindex=j;

}

}

p[repindex]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault!");

}

dispPgFaultCnt();

}

int main()

{

getData();

lru();

dispPgFaultCnt();

return 0;

}

2.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.

#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 NO 10

1.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 n,nf;

int in[100];

int p[50];

int hit=0;

int i,j,k;

int pgfaultcnt=0;

void getData()

{

printf("\nEnter length of page reference sequence:");

scanf("%d",&n);

printf("\nEnter the page reference sequence:");

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

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

printf("\nEnter no of frames:");

scanf("%d",&nf);

}

void initialize()

{

pgfaultcnt=0;

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

p[i]=9999;

}

int isHit(int data)

{

hit=0;

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

{

if(p[j]==data)

{

hit=1;

break;

}

}

return hit;

}

int getHitIndex(int data)

{

int hitind;

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

{

if(p[k]==data)

{

hitind=k;

break;

}

}

return hitind;

}

void dispPages()

{

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

{

if(p[k]!=9999)

printf(" %d",p[k]);

}

}

void dispPgFaultCnt()

{

printf("\nTotal no of page faults:%d",pgfaultcnt);

}

void lru()

{

initialize();

int least[50];

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

{

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

if(isHit(in[i])==0)

{

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

{

int pg=p[j];

int found=0;

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

{

if(pg==in[k])

{

least[j]=k;

found=1;

break;

}

else

found=0;

}

if(!found)

least[j]=-9999;

}

int min=9999;

int repindex;

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

{

if(least[j]<min)

{

min=least[j];

repindex=j;

}

}

p[repindex]=in[i];

pgfaultcnt++;

dispPages();

}

else

printf("No page fault!");

}

dispPgFaultCnt();

}

int main()

{

getData();

lru();

dispPgFaultCnt();

return 0;

}

2.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;

}

}