Slip 1

Q.1 Write a simulation program to implement demand paging using LFU page replacement algorithm. Assume the memory of “n” frames. Show the contents of page after every page replacement in a frame and it at end we told number of page faults accordingly. Input the following page reference string: Page 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;

}

Q.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>  
  
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

Q.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>  
#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;  
}

Q.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 3

Q.2 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

Q.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

#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;  
}

Q.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 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

Q.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>  
#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;  
}

Q.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>  
  
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

Q.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

#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;  
}

Q.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>  
  
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