----------Slip No-1 ----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.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 [15]

Ans--->

```
#include<stdio.h>
int main()
{
int f,p;
int pages[50],frame[10],hit=0,count[50],time[50];
int i,j,page,flag,least,minTime,temp;
printf("Enter no of frames : ");
scanf("%d",&f);
printf("Enter no of pages : ");
```

```
scanf("%d",&p);
for(i=0;i<f;i++)
{
frame[i]=-1;
}
for(i=0;i<50;i++)
{
count[i]=0;
}
printf("Enter page no : \n");
for(i=0;i<p;i++)
{
scanf("%d",&pages[i]);
}
printf("\n");
for(i=0;i<p;i++)
{
count[pages[i]]++;
time[pages[i]]=i;
flag=1;
least=frame[0];
for(j=0;j<f;j++)
{
if(frame[j]==-1 || frame[j]==pages[i])
```

```
{
if(frame[j]!=-1)
{
hit++;
}
flag=0;

frame[j]=pages[i];
break;
}
if(count[least]>count[frame[j]])
{
least=frame[j];
}
}
if(flag)
{
minTime=50;
for(j=0;j<f;j++)
{
if(count[frame[j]]==count[least] && time[frame[j]]<minTime)
{
temp=j;
minTime=time[frame[j]];
```

```
}
}
count[frame[temp]]=0;
frame[temp]=pages[i];
}
for(j=0;j<f;j++)
{
printf("%d ",frame[j]);
}
printf("\n");
}
printf("Page Fault = %d",hit);
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. [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<unistd.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(NULL);
else
{
execvp("demo.sh",args);
}
}
}
return 0;
}
```

OUTPUT:-

hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit typeline.c

hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc typeline.c

hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out

myshell$typeline +1 demo.sh

hello!

myshell$typeline -1 demo.sh

wite a c program that behaves like a shell which display the command prompt myshells.

myshell$typeline a demo.sh

hello!

wite a c program that behaves like a shell which display the command prompt myshells.

myshell$^c

^c (control+c) use to exit from myshell prompt

*create shell file by using gedit demo.sh

demo.sh

hello!

wite a c program that behaves like a shell which display the command prompt myshells.

Q.3. Oral/Viva [05]

----------Slip No-2 ----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

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 [15]

```
#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 [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <dirent.h>
#include<unistd.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%lu\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(NULL);
else
{
execvp("Documents",args);
}
}
}
return 0;
}
```

OUTPUT:-

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit list.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc list.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$list f Documents
deshmukh
sid1
```

avi.c

aaa.c

sid2

deshmukh.c

myshell$list n Documents

3 Dir(s) 6 File(s)

myshell$list i Documents

deshmukh 2369586

sid1 2369590

avi.c 2369525

aaa.c 2369595

sid2 2369592

deshmukh.c 2369585

myshell$^C

Q.3. Oral/Viva [05]

----------Slip No-3 ----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.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 [15]

```
#include<stdio.h>

int findLRU(int time[], int n)
{
int i, minimum = time[0], pos = 0;
for(i = 1; i < n; ++i){
if(time[i] < minimum){
minimum = time[i];
pos = i;
}
}
return pos;
}

int main()
```

```
{
   int no_of_frames, no_of_pages, frames[10], pages[30], counter =
0, time[10], flag1, flag2, i, j, pos, faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);
printf("Enter reference string: ");
   for(i = 0; i < no_of_pages; ++i)
{
    scanf("%d", &pages[i]);
   }
for(i = 0; i < no_of_frames; ++i)
{
    frames[i] = -1;
   }
   for(i = 0; i < no_of_pages; ++i)
{
    flag1 = flag2 = 0;
    for(j = 0; j < no_of_frames; ++j)
{
    if(frames[j] == pages[i])
{
    counter++;
```

```
    time[j] = counter;
   flag1 = flag2 = 1;
   break;
   }
    }
    if(flag1 == 0)
{
for(j = 0; j < no_of_frames; ++j)
{
    if(frames[j] == -1)
{
    counter++;
    faults++;
    frames[j] = pages[i];
    time[j] = counter;
    flag2 = 1;
    break;
    }
    }
    }
    if(flag2 == 0)
{
    pos = findLRU(time, no_of_frames);
    counter++;
```

```
    faults++;
    frames[pos] = pages[i];
    time[pos] = counter;
    }
    printf("\n");
    for(j = 0; j < no_of_frames; ++j)
{
    printf("%d\t", frames[j]);
    }
}
printf("\n\nTotal Page Faults = %d", faults);
   return 0;
}
```

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. [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<sys/wait.h>
#include<unistd.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(NULL);
else
{
execvp("demo.h",args);
}
}
}
return 0;
}
```

OUTPUT:=

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit count.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc count.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$count c demo.sh
No.of characters:248
myshell$count w demo.sh
No.of words:42
myshell$count l demo.sh
No.of lines:7
myshell$^C
```

^c (control+c) use to exit from myshell prompt

*create shell file by using gedit demo.sh

demo.sh

hell0 !

wite a c program that behaves like a shell which display the command prompt myshells.

program logic:

main function will execute and display the command prompt as $.

accept the commaand at $ prompt from user.

system program.

system program

Q.3. Oral/Viva [05]

----------Slip No-4 ----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

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 [15]

```
//program for OTP……
#include<stdio.h>
int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);

   printf("Enter number of pages: ");
   scanf("%d", &no_of_pages);

   printf("Enter page reference string: ");

   for(i = 0; i < no_of_pages; ++i)
{
      scanf("%d", &pages[i]);
   }
```

```
   for(i = 0; i < no_of_frames; ++i)
{
      frames[i] = -1;
   }

   for(i = 0; i < no_of_pages; ++i)
{
      flag1 = flag2 = 0;

      for(j = 0; j < no_of_frames; ++j)
{
         if(frames[j] == pages[i])
{
               flag1 = flag2 = 1;
               break;
            }
      }

      if(flag1 == 0){
         for(j = 0; j < no_of_frames; ++j)
{
            if(frames[j] == -1)
{
               faults++;
```

```
            frames[j] = pages[i];
            flag2 = 1;
            break;
          }
        }
      }

      if(flag2 == 0)
{
       flag3 =0;

        for(j = 0; j < no_of_frames; ++j)
{
         temp[j] = -1;

         for(k = i + 1; k < no_of_pages; ++k)
{
         if(frames[j] == pages[k])
{
         temp[j] = k;
         break;
         }
         }
        }
```

```
        for(j = 0; j < no_of_frames; ++j)
{
         if(temp[j] == -1)
{
         pos = j;
         flag3 = 1;
         break;
         }
        }

        if(flag3 ==0)
{
         max = temp[0];
         pos = 0;
         for(j = 1; j < no_of_frames; ++j)
{
         if(temp[j] > max)
{
         max = temp[j];
         pos = j;
         }
         }
        }
```

```
frames[pos] = pages[i];
faults++;
        }
        printf("\n");
        for(j = 0; j < no_of_frames; ++j)
{
            printf("%d\t", frames[j]);
        }
    }

    printf("\n\nTotal Page Faults = %d", faults);
    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. [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<unistd.h>
#include<sys/wait.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(NULL);
else
{
execvp("demo.sh",args);
}
}
}
return 0;
}
```

OUTPUT:-

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit search.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc search.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$search c program demo.sh
Total No.of Occurrences = 4
```

myshell$search a program demo.sh

2: wite a c program that behaves like a shell which display the command prompt

myshells.

3: program logic:

6: system program.

7: system program.

myshell$search f program demo.sh

2: wite a c program that behaves like a shell which display the command prompt

myshells.

myshell$^C

^c (control+c) use to exit from myshell prompt

*create shell file by using gedit demo.sh

demo.sh

hell0 !

wite a c program that behaves like a shell which display the command prompt myshells.

program logic:

main function will execute and display the command prompt as $.

accept the commaand at $ prompt from user.

system program.

system program

Q.3. Oral/Viva [05]

----------Slip No-5 ----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

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 : 7, 5, 4, 8, 5, 7, 2, 3, 1, 3, 5, 9, 4, 6, 2 [15]

//program for OTP……

```
#include<stdio.h>
int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);
```

```
    printf("Enter number of pages: ");
    scanf("%d", &no_of_pages);

    printf("Enter page reference string: ");

    for(i = 0; i < no_of_pages; ++i)
{
        scanf("%d", &pages[i]);
    }

    for(i = 0; i < no_of_frames; ++i)
{
        frames[i] = -1;
    }

    for(i = 0; i < no_of_pages; ++i)
{
        flag1 = flag2 = 0;

        for(j = 0; j < no_of_frames; ++j)
{
            if(frames[j] == pages[i])
{
```

```
            flag1 = flag2 = 1;
            break;
          }
      }

      if(flag1 == 0){
         for(j = 0; j < no_of_frames; ++j)
{
           if(frames[j] == -1)
{
              faults++;
              frames[j] = pages[i];
              flag2 = 1;
              break;
           }
         }
      }

      if(flag2 == 0)
{
       flag3 =0;

         for(j = 0; j < no_of_frames; ++j)
{
```

```
            temp[j] = -1;

            for(k = i + 1; k < no_of_pages; ++k)
{
            if(frames[j] == pages[k])
{
            temp[j] = k;
            break;
            }
            }
           }

           for(j = 0; j < no_of_frames; ++j)
{
            if(temp[j] == -1)
{
            pos = j;
            flag3 = 1;
            break;
            }
           }

           if(flag3 ==0)
{
```

```
            max = temp[0];
            pos = 0;
            for(j = 1; j < no_of_frames; ++j)
{
            if(temp[j] > max)
{
            max = temp[j];
            pos = j;
            }
            }
           }
frames[pos] = pages[i];
faults++;
        }
        printf("\n");
        for(j = 0; j < no_of_frames; ++j)
{
           printf("%d\t", frames[j]);
        }
    }

    printf("\n\nTotal Page Faults = %d", faults);
    return 0;
}
```

Q.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. [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<unistd.h>
#include<sys/wait.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(NULL);
else
{
execvp("demo.sh",args);
}
```

}

}

return 0;

}

OUTPUT:-

hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit search.c

hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc search.c

hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out

myshell$search c program demo.sh

Total No.of Occurrences = 4

myshell$search a program demo.sh

2: wite a c program that behaves like a shell which display the command prompt

myshells.

3: program logic:

6: system program.

7: system program.

myshell$search f program demo.sh

2: wite a c program that behaves like a shell which display the command prompt

myshells.

myshell$^C

^c (control+c) use to exit from myshell prompt

*create shell file by using gedit demo.sh

demo.sh

hell0 !

wite a c program that behaves like a shell which display the command prompt myshells.

program logic:

main function will execute and display the command prompt as $.

accept the commaand at $ prompt from user.

system program.

system program

Q.3. Oral/Viva [05]

----------Slip No-6----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.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 [15]

```
#include<stdio.h>

int findLRU(int time[], int n)
{
int i, minimum = time[0], pos = 0;
for(i = 1; i < n; ++i){
if(time[i] < minimum){
minimum = time[i];
pos = i;
}
}
return pos;
}

int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], counter = 0, time[10], flag1, flag2, i, j, pos, faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);
```

```
printf("Enter reference string: ");
    for(i = 0; i < no_of_pages; ++i)
{
     scanf("%d", &pages[i]);
    }
for(i = 0; i < no_of_frames; ++i)
{
     frames[i] = -1;
    }
    for(i = 0; i < no_of_pages; ++i)
{
     flag1 = flag2 = 0;
     for(j = 0; j < no_of_frames; ++j)
{
     if(frames[j] == pages[i])
{
     counter++;
     time[j] = counter;
   flag1 = flag2 = 1;
   break;
   }
     }
     if(flag1 == 0)
{
```

```
for(j = 0; j < no_of_frames; ++j)
{
    if(frames[j] == -1)
{
    counter++;
    faults++;
    frames[j] = pages[i];
    time[j] = counter;
    flag2 = 1;
    break;
    }
    }
    }
    if(flag2 == 0)
{
    pos = findLRU(time, no_of_frames);
    counter++;
    faults++;
    frames[pos] = pages[i];
    time[pos] = counter;
    }
    printf("\n");
    for(j = 0; j < no_of_frames; ++j)
{
```

```
    printf("%d\t", frames[j]);
    }
}
printf("\n\nTotal Page Faults = %d", faults);
   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 c filename pattern :- To count the number of occurrence

of pattern in the file. [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<unistd.h>
#include<sys/wait.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(NULL);
else
{
execvp("demo.sh",args);
}
}
}
return 0;
}
```

OUTPUT:-

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit search.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc search.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$search c program demo.sh
Total No.of Occurrences = 4
myshell$search a program demo.sh
2: wite a c program that behaves like a shell which display the
command prompt
myshells.
3: program logic:
6: system program.
7: system program.
myshell$search f program demo.sh
```

2: wite a c program that behaves like a shell which display the command prompt

myshells.

myshell$^C

^c (control+c) use to exit from myshell prompt

*create shell file by using gedit demo.sh

demo.sh

hell0 !

wite a c program that behaves like a shell which display the command prompt myshells.

program logic:

main function will execute and display the command prompt as $.

accept the commaand at $ prompt from user.

system program.

system program

Q.3. Oral/Viva [05]

----------Slip No-7 ----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

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 : 8, 5, 7, 8, 5, 7, 2, 3, 7, 3, 5, 9, 4, 6, 2 [15]

```
#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];
  }
 }
```

}

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 a filename pattern :- To search all the occurrence of

pattern in the file. [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<unistd.h>
#include<sys/wait.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(NULL);
else
{
execvp("demo.sh",args);
```

```
}
}
}
return 0;
}
```

OUTPUT:-

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit search.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc search.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$search c program demo.sh
Total No.of Occurrences = 4
myshell$search a program demo.sh
2: wite a c program that behaves like a shell which display the command prompt
myshells.
3: program logic:
6: system program.
7: system program.
myshell$search f program demo.sh
2: wite a c program that behaves like a shell which display the command prompt
myshells.
myshell$^C
^c (control+c) use to exit from myshell prompt
```

*create shell file by using gedit demo.sh

demo.sh

hell0 !

wite a c program that behaves like a shell which display the command prompt myshells.

program logic:

main function will execute and display the command prompt as $.

accept the commaand at $ prompt from user.

system program.

system program

Q.3. Oral/Viva [05]

----------Slip No-8----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

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 : 2, 4, 5, 6, 9, 4, 7, 3, 4, 5, 6, 7, 2, 4, 7, 1
[15]

```
#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];
  }
 }
}
```

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 i dirname :- To print names and inodes of the files in the

current directory. [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <dirent.h>
#include<unistd.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%lu\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(NULL);
else
{
execvp("Documents",args);
}
}
}
return 0;
```

}

OUTPUT:-

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit list.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc list.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$list f Documents
deshmukh
sid1
avi.c
aaa.c
sid2
deshmukh.c
myshell$list n Documents
3 Dir(s) 6 File(s)
myshell$list i Documents
deshmukh 2369586
sid1 2369590
avi.c 2369525
aaa.c 2369595
sid2 2369592
deshmukh.c 2369585
myshell$^C
```

Q.3. Oral/Viva [05]

----------Slip No-9----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.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 : 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6 [15]

```
#include<stdio.h>

int findLRU(int time[], int n)
{
int i, minimum = time[0], pos = 0;
for(i = 1; i < n; ++i){
if(time[i] < minimum){
minimum = time[i];
pos = i;
```

```
}
}
return pos;
}

int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], counter =
0, time[10], flag1, flag2, i, j, pos, faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);
printf("Enter reference string: ");
   for(i = 0; i < no_of_pages; ++i)
{
    scanf("%d", &pages[i]);
   }
for(i = 0; i < no_of_frames; ++i)
{
    frames[i] = -1;
   }
   for(i = 0; i < no_of_pages; ++i)
{
```

```
    flag1 = flag2 = 0;
    for(j = 0; j < no_of_frames; ++j)
{
    if(frames[j] == pages[i])
{
    counter++;
    time[j] = counter;
   flag1 = flag2 = 1;
   break;
   }
    }
    if(flag1 == 0)
{
for(j = 0; j < no_of_frames; ++j)
{
    if(frames[j] == -1)
{
    counter++;
    faults++;
    frames[j] = pages[i];
    time[j] = counter;
    flag2 = 1;
    break;
    }
```

```
    }
    }
    if(flag2 == 0)
{
    pos = findLRU(time, no_of_frames);
    counter++;
    faults++;
    frames[pos] = pages[i];
    time[pos] = counter;
    }
    printf("\n");
    for(j = 0; j < no_of_frames; ++j)
{
    printf("%d\t", frames[j]);
    }
}
printf("\n\nTotal Page Faults = %d", faults);
   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 [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <dirent.h>
#include<unistd.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%lu\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(NULL);
else
{
execvp("Documents",args);
}
}
}
return 0;
}
```

OUTPUT:-

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit list.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc list.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$list f Documents
deshmukh
sid1
avi.c
aaa.c
sid2
deshmukh.c
myshell$list n Documents
3 Dir(s) 6 File(s)
myshell$list i Documents
deshmukh 2369586
sid1 2369590
avi.c 2369525
aaa.c 2369595
sid2 2369592
deshmukh.c 2369585
myshell$^C
```

Q.3. Oral/Viva [05]

----------Slip No-10----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.1 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. [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<unistd.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(NULL);
else
{
execvp("demo.sh",args);
}
}
}
return 0;
}
```

OUTPUT:-

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit typeline.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc typeline.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$typeline +1 demo.sh
hello!
myshell$typeline -1 demo.sh
wite a c program that behaves like a shell which display the
command prompt myshells.
myshell$typeline a demo.sh
```

hello!

wite a c program that behaves like a shell which display the command prompt myshells.

myshell$^c

^c (control+c) use to exit from myshell prompt

*create shell file by using gedit demo.sh

demo.sh

hello!

wite a c program that behaves like a shell which display the command prompt myshells

Q.2 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.

[15]

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>
```

```
typedef struct process_info
{
char pname[20];
int at,bt,ct,bt1;
struct process_info *next;
}NODE;
int n,ts;
NODE *first,*last;
void accept_info()
{
NODE *p;
int i;
printf("Enter no.of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p = (NODE*)malloc(sizeof(NODE));
printf("Enter process name:");
scanf("%s",p->pname);
printf("Enter arrival time:");
scanf("%d",&p->at);
printf("Enter first CPU burst time:");
scanf("%d",&p->bt);
p->bt1 = p->bt;
```

```
p->next = NULL;
if(first==NULL)
first=p;
else
last->next=p;
last = p;
}
printf("Enter time slice:");
scanf("%d",&ts);
}
void print_output()
{
NODE *p;
float avg_tat=0,avg_wt=0;
printf("pname\tat\tbt\tct\ttat\twt\n");
p = first;
while(p!=NULL)
{
int tat = p->ct-p->at;
int wt = tat-p->bt;
avg_tat+=tat;
avg_wt+=wt;
printf("%s\t%d\t%d\t%d\t%d\t%d\n",
p->pname,p->at,p->bt,p->ct,tat,wt);
```

```
p=p->next;
}
printf("Avg TAT=%f\tAvg WT=%f\n",
avg_tat/n,avg_wt/n);
}
void print_input()
{
NODE *p;
p = first;
printf("pname\tat\tbt\n");
while(p!=NULL)
{
printf("%s\t%d\t%d\n",
p->pname,p->at,p->bt1);
p = p->next;
}
}
void sort()
{
NODE *p,*q;
int t;
char name[20];
p = first;
while(p->next!=NULL)
```

```
{
q=p->next;
while(q!=NULL)
{
if(p->at > q->at)
{
strcpy(name,p->pname);
strcpy(p->pname,q->pname);
strcpy(q->pname,name);
t = p->at;
p->at = q->at;
q->at = t;
t = p->bt;
p->bt = q->bt;
q->bt = t;
t = p->ct;
p->ct = q->ct;
q->ct = t;
t = p->bt1;
p->bt1 = q->bt1;
q->bt1 = t;
}
q=q->next;
}
```

```
p=p->next;
}
}
int time;
int is_arrived()
{
NODE *p;
p = first;
while(p!=NULL)
{
if(p->at<=time && p->bt1!=0)
return 1;
p=p->next;
}
return 0;
}
NODE * delq()
{
NODE *t;
t = first;
first = first->next;
t->next=NULL;
return t;
}
```

```
void addq(NODE *t)
{
last->next = t;
last = t;
}
struct gantt_chart
{
int start;
char pname[30];
int end;
}s[100],s1[100];
int k;
void rr()
{
int prev=0,n1=0;
NODE *p;
while(n1!=n)
{
if(!is_arrived())
{
time++;
s[k].start = prev;
strcpy(s[k].pname,"*");
s[k].end = time;
```

```
k++;
prev=time;
}
else
{
p = first;
while(1)
{
if(p->at<=time && p->bt1!=0)
break;
p = delq();
addq(p);
p = first;
}
if(p->bt1<=ts)
{
time+=p->bt1;
p->bt1=0;
}
else
{
time+=ts;
p->bt1-=ts;
}
```

```
p->ct = time;
s[k].start = prev;
strcpy(s[k].pname,p->pname);
s[k].end = time;
k++;
prev = time;
if(p->bt1==0) n1++;
p = delq();
addq(p);
}
print_input();
}
}
void print_gantt_chart()
{
int i,j,m;
s1[0] = s[0];
for(i=1,j=0;i<k;i++)
{
if(strcmp(s[i].pname,s1[j].pname)==0)
s1[j].end = s[i].end;
else
s1[++j] = s[i];
}
```

```
printf("%d",s1[0].start);
for(i=0;i<=j;i++)
{
m = (s1[i].end - s1[i].start);
for(k=0;k<m/2;k++)
printf("-");
printf("%s",s1[i].pname);
for(k=0;k<(m+1)/2;k++)
printf("-");
printf("%d",s1[i].end);
}
}
int main()
{
accept_info();
sort();
rr();
print_output();
print_gantt_chart();
return 0;
}
```

Q.3. Oral/Viva [05]

----------Slip No-11----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.1 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. [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<unistd.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(NULL);
else
{
execvp("demo.sh",args);
}
}
}
return 0;
}
```

OUTPUT:-

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit typeline.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc typeline.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$typeline +1 demo.sh
hello!
myshell$typeline -1 demo.sh
```

wite a c program that behaves like a shell which display the command prompt myshells.

myshell$typeline a demo.sh

hello!

wite a c program that behaves like a shell which display the command prompt myshells.

myshell$^c

^c (control+c) use to exit from myshell prompt

*create shell file by using gedit demo.sh

demo.sh

hello!

wite a c program that behaves like a shell which display the command prompt myshells

Q.2 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 [15]

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>
```

```
typedef struct process_info
{
char pname[20];
int at,bt,ct,bt1;
struct process_info *next;
}NODE;
int n;
NODE *first,*last;
void accept_info()
{
NODE *p;
int i;
printf("Enter no.of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p = (NODE*)malloc(sizeof(NODE));
printf("Enter process name:");
scanf("%s",p->pname);
printf("Enter arrival time:");
scanf("%d",&p->at);
printf("Enter first CPU burst time:");
scanf("%d",&p->bt);
p->bt1 = p->bt;
```

```
p->next = NULL;
if(first==NULL)
first=p;
else
last->next=p;
last = p;
}
}
void print_output()
{
NODE *p;
float avg_tat=0,avg_wt=0;
printf("pname\tat\tbt\tct\ttat\twt\n");
p = first;
while(p!=NULL)
{
int tat = p->ct-p->at;
int wt = tat-p->bt;
avg_tat+=tat;
avg_wt+=wt;
printf("%s\t%d\t%d\t%d\t%d\t%d\n",
p->pname,p->at,p->bt,p->ct,tat,wt);
p=p->next;
}
```

```
printf("Avg TAT=%f\tAvg WT=%f\n",
avg_tat/n,avg_wt/n);
}
void print_input()
{
NODE *p;
p = first;
printf("pname\tat\tbt\n");
while(p!=NULL)
{
printf("%s\t%d\t%d\n",
p->pname,p->at,p->bt1);
p = p->next;
}
}
void sort()
{
NODE *p,*q;
int t;
char name[20];
p = first;
while(p->next!=NULL)
{
q=p->next;
```

```
while(q!=NULL)
{
if(p->at > q->at)
{
strcpy(name,p->pname);
strcpy(p->pname,q->pname);
strcpy(q->pname,name);
t = p->at;
p->at = q->at;
q->at = t;
t = p->bt;
p->bt = q->bt;
q->bt = t;
t = p->ct;
p->ct = q->ct;
q->ct = t;
t = p->bt1;
p->bt1 = q->bt1;
q->bt1 = t;
}
q=q->next;
}
p=p->next;
}
```

```
}
int time;
NODE * get_sjf()
{
NODE *p,*min_p=NULL;
int min=9999;
p = first;
while(p!=NULL)
{
if(p->at<=time && p->bt1!=0 &&
p->bt1<min)
{
min = p->bt1;
min_p = p;
}
p=p->next;
}
return min_p;
}
struct gantt_chart
{
int start;
char pname[30];
int end;
```

```
}s[100],s1[100];
int k;
void sjfnp()
{
int prev=0,n1=0;
NODE *p;
while(n1!=n)
{
p = get_sjf();
if(p==NULL)
{
time++;
s[k].start = prev;
strcpy(s[k].pname,"*");
s[k].end = time;
prev = time;
k++;
}
else
{
time+=p->bt1;
s[k].start = prev;
strcpy(s[k].pname, p->pname);
s[k].end = time;
```

```
prev = time;
k++;
p->ct = time;
p->bt1 = 0;
n1++;
}
print_input();
sort();
}
}
void print_gantt_chart()
{
int i,j,m;
s1[0] = s[0];
for(i=1,j=0;i<k;i++)
{
if(strcmp(s[i].pname,s1[j].pname)==0)
s1[j].end = s[i].end;
else
s1[++j] = s[i];
}
printf("%d",s1[0].start);
for(i=0;i<=j;i++)
{
```

```
m = (s1[i].end - s1[i].start);
for(k=0;k<m/2;k++)
printf("-");
printf("%s",s1[i].pname);
for(k=0;k<(m+1)/2;k++)
printf("-");
printf("%d",s1[i].end);
}
}
int main()
{
accept_info();
sort();
sjfnp();
print_output();
print_gantt_chart();
return 0;
}
```

Q.3. Oral/Viva [05]

----------Slip No-12----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.1 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.

[15]

PROGRAM:-

```
#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/wait.h>

#include <dirent.h>
```

```
#include<unistd.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%lu\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(NULL);
else
{
execvp("Documents",args);
}
```

```
}
}
return 0;
}
```

OUTPUT:-

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit list.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc list.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$list f Documents
deshmukh
sid1
avi.c
aaa.c
sid2
deshmukh.c
myshell$list n Documents
3 Dir(s) 6 File(s)
myshell$list i Documents
deshmukh 2369586
sid1 2369590
avi.c 2369525
aaa.c 2369595
sid2 2369592
deshmukh.c 2369585
```

myshell$^C

Q.2 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. The

output should give Gantt chart, turnaround time and waiting time for each

process. Also find the average waiting time and turnaround time [15]

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct process_info
{
char pname[20];
int at,bt,ct,bt1;
struct process_info *next;
}NODE;
int n;
NODE *first,*last;
void accept_info()
```

```
{
NODE *p;
int i;
printf("Enter no.of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p = (NODE*)malloc(sizeof(NODE));
printf("Enter process name:");
scanf("%s",p->pname);
printf("Enter arrival time:");
scanf("%d",&p->at);
printf("Enter first CPU burst time:");
scanf("%d",&p->bt);
p->bt1 = p->bt;
p->next = NULL;
if(first==NULL)
first=p;
else
last->next=p;
last = p;
}
}
void print_output()
```

```
{
NODE *p;
float avg_tat=0,avg_wt=0;
printf("pname\tat\tbt\tct\ttat\twt\n");
p = first;
while(p!=NULL)
{
int tat = p->ct-p->at;
int wt = tat-p->bt;
avg_tat+=tat;
avg_wt+=wt;
printf("%s\t%d\t%d\t%d\t%d\t%d\n",
p->pname,p->at,p->bt,p->ct,tat,wt);
p=p->next;
}
printf("Avg TAT=%f\tAvg WT=%f\n",
avg_tat/n,avg_wt/n);
}
void print_input()
{
NODE *p;
p = first;
printf("pname\tat\tbt\n");
while(p!=NULL)
```

```
{
printf("%s\t%d\t%d\n",
p->pname,p->at,p->bt1);
p = p->next;
}
}
void sort()
{
NODE *p,*q;
int t;
char name[20];
p = first;
while(p->next!=NULL)
{
q=p->next;
while(q!=NULL)
{
if(p->at > q->at)
{
strcpy(name,p->pname);
strcpy(p->pname,q->pname);
strcpy(q->pname,name);
t = p->at;
p->at = q->at;
```

```
q->at = t;
t = p->bt;
p->bt = q->bt;
q->bt = t;
t = p->ct;
p->ct = q->ct;
q->ct = t;
t = p->bt1;
p->bt1 = q->bt1;
q->bt1 = t;
}
q=q->next;
}
p=p->next;
}
}
int time;
NODE * get_sjf()
{
NODE *p,*min_p=NULL;
int min=9999;
p = first;
while(p!=NULL)
{
```

```
if(p->at<=time && p->bt1!=0 &&
p->bt1<min)
{
min = p->bt1;
min_p = p;
}
p=p->next;
}
return min_p;
}
struct gantt_chart
{
int start;
char pname[30];
int end;
}s[100],s1[100];
int k;
void sjfp()
{
int prev=0,n1=0;
NODE *p;
while(n1!=n)
{
p = get_sjf();
```

```
if(p==NULL)
{
time++;
s[k].start = prev;
strcpy(s[k].pname,"*");
s[k].end = time;
prev = time;
k++;
}
else
{
time++;
s[k].start = prev;
strcpy(s[k].pname, p->pname);
s[k].end = time;
prev = time;
k++;
p->ct = time;
p->bt1--;
if(p->bt1==0)
n1++;
}
print_input();
sort();
```

```
}
}
void print_gantt_chart()
{
int i,j,m;
s1[0] = s[0];
for(i=1,j=0;i<k;i++)
{
if(strcmp(s[i].pname,s1[j].pname)==0)
s1[j].end = s[i].end;
else
s1[++j] = s[i];
}
printf("%d",s1[0].start);
for(i=0;i<=j;i++)
{
m = (s1[i].end - s1[i].start);
for(k=0;k<m/2;k++)
printf("-");
printf("%s",s1[i].pname);
for(k=0;k<(m+1)/2;k++)
printf("-");
printf("%d",s1[i].end);
}
```

```
}
int main()
{
accept_info();
sort();
sjfp();
print_output();
print_gantt_chart();
return 0;
}
```

Q.3. Oral/Viva [05]

----------Slip No-13----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

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 : 7, 5, 4, 8, 5, 7, 2, 3, 1, 3, 5, 9, 4, 6, [15]

//program for OTP……

```
#include<stdio.h>
int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);

   printf("Enter number of pages: ");
   scanf("%d", &no_of_pages);

   printf("Enter page reference string: ");

   for(i = 0; i < no_of_pages; ++i)
{
      scanf("%d", &pages[i]);
   }
```

```
   for(i = 0; i < no_of_frames; ++i)
{
      frames[i] = -1;
   }

   for(i = 0; i < no_of_pages; ++i)
{
      flag1 = flag2 = 0;

      for(j = 0; j < no_of_frames; ++j)
{
         if(frames[j] == pages[i])
{
               flag1 = flag2 = 1;
               break;
            }
      }

      if(flag1 == 0){
         for(j = 0; j < no_of_frames; ++j)
{
            if(frames[j] == -1)
{
```

```
            faults++;
            frames[j] = pages[i];
            flag2 = 1;
            break;
          }
        }
      }

      if(flag2 == 0)
{
       flag3 =0;

         for(j = 0; j < no_of_frames; ++j)
{
          temp[j] = -1;

          for(k = i + 1; k < no_of_pages; ++k)
{
          if(frames[j] == pages[k])
{
          temp[j] = k;
          break;
          }
          }
```

```
        }

        for(j = 0; j < no_of_frames; ++j)
{
         if(temp[j] == -1)
{
         pos = j;
         flag3 = 1;
         break;
         }
        }

        if(flag3 ==0)
{
         max = temp[0];
         pos = 0;
         for(j = 1; j < no_of_frames; ++j)
{
         if(temp[j] > max)
{
         max = temp[j];
         pos = j;
         }
         }
```

```
            }
frames[pos] = pages[i];
faults++;
        }
        printf("\n");
        for(j = 0; j < no_of_frames; ++j)
{
            printf("%d\t", frames[j]);
        }
    }

    printf("\n\nTotal Page Faults = %d", faults);
    return 0;
}
```

Q.2 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. [15]

```
#include<stdio.h>
```

```
#include<stdlib.h>
#include<string.h>
typedef struct process_info
{
char pname[20];
int at,bt,ct,bt1;
struct process_info *next;
}NODE;
int n;
NODE *first,*last;
void accept_info()
{
NODE *p;
int i;
printf("Enter no.of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p = (NODE*)malloc(sizeof(NODE));
printf("Enter process name:");
scanf("%s",p->pname);
printf("Enter arrival time:");
scanf("%d",&p->at);
printf("Enter first CPU burst time:");
```

```
scanf("%d",&p->bt);
p->bt1 = p->bt;
p->next = NULL;
if(first==NULL)
first=p;
else
last->next=p;
last = p;
}
}
void print_output()
{
NODE *p;
float avg_tat=0,avg_wt=0;
printf("pname\tat\tbt\tct\ttat\twt\n");
p = first;
while(p!=NULL)
{
int tat = p->ct-p->at;
int wt = tat-p->bt;
avg_tat+=tat;
avg_wt+=wt;
printf("%s\t%d\t%d\t%d\t%d\t%d\n",
p->pname,p->at,p->bt,p->ct,tat,wt);
```

```
p=p->next;
}
printf("Avg TAT=%f\tAvg WT=%f\n",
avg_tat/n,avg_wt/n);
}
void print_input()
{
NODE *p;
p = first;
printf("pname\tat\tbt\n");
while(p!=NULL)
{
printf("%s\t%d\t%d\n",
p->pname,p->at,p->bt1);
p = p->next;
}
}
void sort()
{
NODE *p,*q;
int t;
char name[20];
p = first;
while(p->next!=NULL)
```

```
{
q=p->next;
while(q!=NULL)
{
if(p->at > q->at)
{
strcpy(name,p->pname);
strcpy(p->pname,q->pname);
strcpy(q->pname,name);
t = p->at;
p->at = q->at;
q->at = t;
t = p->bt;
p->bt = q->bt;
q->bt = t;
t = p->ct;
p->ct = q->ct;
q->ct = t;
t = p->bt1;
p->bt1 = q->bt1;
q->bt1 = t;
}
q=q->next;
}
```

```
p=p->next;
}
}
int time;
NODE * get_fcfs()
{
NODE *p;
p = first;
while(p!=NULL)
{
if(p->at<=time && p->bt1!=0)
return p;
p=p->next;
}
return NULL;
}
struct gantt_chart
{
int start;
char pname[30];
int end;
}s[100],s1[100];
int k;
void fcfs()
```

```
{
int prev=0,n1=0;
NODE *p;
while(n1!=n)
{
p = get_fcfs();
if(p==NULL)
{
time++;
s[k].start = prev;
strcpy(s[k].pname,"*");
s[k].end = time;
prev = time;
k++;
}
else
{
time+=p->bt1;
s[k].start = prev;
strcpy(s[k].pname, p->pname);
s[k].end = time;
prev = time;
k++;
p->ct = time;
```

```
p->bt1 = 0;
n1++;
}
print_input();
sort();
}
}
void print_gantt_chart()
{
int i,j,m;
s1[0] = s[0];
for(i=1,j=0;i<k;i++)
{
if(strcmp(s[i].pname,s1[j].pname)==0)
s1[j].end = s[i].end;
else
s1[++j] = s[i];
}
printf("%d",s1[0].start);
for(i=0;i<=j;i++)
{
m = (s1[i].end - s1[i].start);
for(k=0;k<m/2;k++)
printf("-");
```

```
printf("%s",s1[i].pname);
for(k=0;k<(m+1)/2;k++)
printf("-");
printf("%d",s1[i].end);
}
}
int main()
{
accept_info();
sort();
fcfs();
print_output();
print_gantt_chart();
return 0;
}
```

Q.3. Oral/Viva [05]

----------Slip No-14----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.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 : 3, 4, 5, 6, 3, 4, 7, 3, 4, 5, 6, 7, 2, 4, 6 [15]

```
#include<stdio.h>

int findLRU(int time[], int n)
{
int i, minimum = time[0], pos = 0;
for(i = 1; i < n; ++i){
if(time[i] < minimum){
minimum = time[i];
pos = i;
}
}
return pos;
}

int main()
{
```

```
   int no_of_frames, no_of_pages, frames[10], pages[30], counter =
0, time[10], flag1, flag2, i, j, pos, faults = 0;
printf("Enter number of frames: ");
scanf("%d", &no_of_frames);
printf("Enter number of pages: ");
scanf("%d", &no_of_pages);
printf("Enter reference string: ");
   for(i = 0; i < no_of_pages; ++i)
{
    scanf("%d", &pages[i]);
   }
for(i = 0; i < no_of_frames; ++i)
{
    frames[i] = -1;
   }
   for(i = 0; i < no_of_pages; ++i)
{
    flag1 = flag2 = 0;
    for(j = 0; j < no_of_frames; ++j)
{
    if(frames[j] == pages[i])
{
    counter++;
    time[j] = counter;
```

```
   flag1 = flag2 = 1;
   break;
   }
    }
    if(flag1 == 0)
{
for(j = 0; j < no_of_frames; ++j)
{
    if(frames[j] == -1)
{
    counter++;
    faults++;
    frames[j] = pages[i];
    time[j] = counter;
    flag2 = 1;
    break;
    }
    }
    }
    if(flag2 == 0)
{
    pos = findLRU(time, no_of_frames);
    counter++;
    faults++;
```

```
    frames[pos] = pages[i];
    time[pos] = counter;
    }
    printf("\n");
    for(j = 0; j < no_of_frames; ++j)
{
    printf("%d\t", frames[j]);
    }
}
printf("\n\nTotal Page Faults = %d", faults);
   return 0;
}
```

Q.2 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. [15]

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct process_info
{
char pname[20];
int at,bt,ct,bt1;
struct process_info *next;
}NODE;
int n;
NODE *first,*last;
void accept_info()
{
NODE *p;
int i;
printf("Enter no.of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p = (NODE*)malloc(sizeof(NODE));
printf("Enter process name:");
scanf("%s",p->pname);
printf("Enter arrival time:");
scanf("%d",&p->at);
```

```
printf("Enter first CPU burst time:");
scanf("%d",&p->bt);
p->bt1 = p->bt;
p->next = NULL;
if(first==NULL)
first=p;
else
last->next=p;
last = p;
}
}
void print_output()
{
NODE *p;
float avg_tat=0,avg_wt=0;
printf("pname\tat\tbt\tct\ttat\twt\n");
p = first;
while(p!=NULL)
{
int tat = p->ct-p->at;
int wt = tat-p->bt;
avg_tat+=tat;
avg_wt+=wt;
printf("%s\t%d\t%d\t%d\t%d\t%d\n",
```

```
p->pname,p->at,p->bt,p->ct,tat,wt);
p=p->next;
}
printf("Avg TAT=%f\tAvg WT=%f\n",
avg_tat/n,avg_wt/n);
}
void print_input()
{
NODE *p;
p = first;
printf("pname\tat\tbt\n");
while(p!=NULL)
{
printf("%s\t%d\t%d\n",
p->pname,p->at,p->bt1);
p = p->next;
}
}
void sort()
{
NODE *p,*q;
int t;
char name[20];
p = first;
```

```
while(p->next!=NULL)
{
q=p->next;
while(q!=NULL)
{
if(p->at > q->at)
{
strcpy(name,p->pname);
strcpy(p->pname,q->pname);
strcpy(q->pname,name);
t = p->at;
p->at = q->at;
q->at = t;
t = p->bt;
p->bt = q->bt;
q->bt = t;
t = p->ct;
p->ct = q->ct;
q->ct = t;
t = p->bt1;
p->bt1 = q->bt1;
q->bt1 = t;
}
q=q->next;
```

```
}
p=p->next;
}
}
int time;
NODE * get_fcfs()
{
NODE *p;
p = first;
while(p!=NULL)
{
if(p->at<=time && p->bt1!=0)
return p;
p=p->next;
}
return NULL;
}
struct gantt_chart
{
int start;
char pname[30];
int end;
}s[100],s1[100];
int k;
```

```
void fcfs()
{
int prev=0,n1=0;
NODE *p;
while(n1!=n)
{
p = get_fcfs();
if(p==NULL)
{
time++;
s[k].start = prev;
strcpy(s[k].pname,"*");
s[k].end = time;
prev = time;
k++;
}
else
{
time+=p->bt1;
s[k].start = prev;
strcpy(s[k].pname, p->pname);
s[k].end = time;
prev = time;
k++;
```

```
p->ct = time;
p->bt1 = 0;
n1++;
}
print_input();
sort();
}
}
void print_gantt_chart()
{
int i,j,m;
s1[0] = s[0];
for(i=1,j=0;i<k;i++)
{
if(strcmp(s[i].pname,s1[j].pname)==0)
s1[j].end = s[i].end;
else
s1[++j] = s[i];
}
printf("%d",s1[0].start);
for(i=0;i<=j;i++)
{
m = (s1[i].end - s1[i].start);
for(k=0;k<m/2;k++)
```

```
printf("-");
printf("%s",s1[i].pname);
for(k=0;k<(m+1)/2;k++)
printf("-");
printf("%d",s1[i].end);
}
}
int main()
{
accept_info();
sort();
fcfs();
print_output();
print_gantt_chart();
return 0;
}
```

Q.3. Oral/Viva [05]

----------Slip No-15----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.1 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. [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#include <dirent.h>
#include<unistd.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%lu\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(NULL);
else
{
execvp("Documents",args);
}
}
}
```

```
return 0;
}
```

OUTPUT:-

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit list.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc list.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$list f Documents
deshmukh
sid1
avi.c
aaa.c
sid2
deshmukh.c
myshell$list n Documents
3 Dir(s) 6 File(s)
myshell$list i Documents
deshmukh 2369586
sid1 2369590
avi.c 2369525
aaa.c 2369595
sid2 2369592
deshmukh.c 2369585
myshell$^
```

Q.2 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.

[15]

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct process_info
{
char pname[20];
int at,bt,ct,bt1;
struct process_info *next;
}NODE;
int n,ts;
NODE *first,*last;
void accept_info()
{
```

```
NODE *p;
int i;
printf("Enter no.of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p = (NODE*)malloc(sizeof(NODE));
printf("Enter process name:");
scanf("%s",p->pname);
printf("Enter arrival time:");
scanf("%d",&p->at);
printf("Enter first CPU burst time:");
scanf("%d",&p->bt);
p->bt1 = p->bt;
p->next = NULL;
if(first==NULL)
first=p;
else
last->next=p;
last = p;
}
printf("Enter time slice:");
scanf("%d",&ts);
}
```

```
void print_output()
{
NODE *p;
float avg_tat=0,avg_wt=0;
printf("pname\tat\tbt\tct\ttat\twt\n");
p = first;
while(p!=NULL)
{
int tat = p->ct-p->at;
int wt = tat-p->bt;
avg_tat+=tat;
avg_wt+=wt;
printf("%s\t%d\t%d\t%d\t%d\t%d\n",
p->pname,p->at,p->bt,p->ct,tat,wt);
p=p->next;
}
printf("Avg TAT=%f\tAvg WT=%f\n",
avg_tat/n,avg_wt/n);
}
void print_input()
{
NODE *p;
p = first;
printf("pname\tat\tbt\n");
```

```
while(p!=NULL)
{
printf("%s\t%d\t%d\n",
p->pname,p->at,p->bt1);
p = p->next;
}
}
void sort()
{
NODE *p,*q;
int t;
char name[20];
p = first;
while(p->next!=NULL)
{
q=p->next;
while(q!=NULL)
{
if(p->at > q->at)
{
strcpy(name,p->pname);
strcpy(p->pname,q->pname);
strcpy(q->pname,name);
t = p->at;
```

```
p->at = q->at;
q->at = t;
t = p->bt;
p->bt = q->bt;
q->bt = t;
t = p->ct;
p->ct = q->ct;
q->ct = t;
t = p->bt1;
p->bt1 = q->bt1;
q->bt1 = t;
}
q=q->next;
}
p=p->next;
}
}
int time;
int is_arrived()
{
NODE *p;
p = first;
while(p!=NULL)
{
```

```
if(p->at<=time && p->bt1!=0)
return 1;
p=p->next;
}
return 0;
}
NODE * delq()
{
NODE *t;
t = first;
first = first->next;
t->next=NULL;
return t;
}
void addq(NODE *t)
{
last->next = t;
last = t;
}
struct gantt_chart
{
int start;
char pname[30];
int end;
```

```
}s[100],s1[100];
int k;
void rr()
{
int prev=0,n1=0;
NODE *p;
while(n1!=n)
{
if(!is_arrived())
{
time++;
s[k].start = prev;
strcpy(s[k].pname,"*");
s[k].end = time;
k++;
prev=time;
}
else
{
p = first;
while(1)
{
if(p->at<=time && p->bt1!=0)
break;
```

```
p = delq();
addq(p);
p = first;
}
if(p->bt1<=ts)
{
time+=p->bt1;
p->bt1=0;
}
else
{
time+=ts;
p->bt1-=ts;
}
p->ct = time;
s[k].start = prev;
strcpy(s[k].pname,p->pname);
s[k].end = time;
k++;
prev = time;
if(p->bt1==0) n1++;
p = delq();
addq(p);
}
```

```
print_input();
}
}
void print_gantt_chart()
{
int i,j,m;
s1[0] = s[0];
for(i=1,j=0;i<k;i++)
{
if(strcmp(s[i].pname,s1[j].pname)==0)
s1[j].end = s[i].end;
else
s1[++j] = s[i];
}
printf("%d",s1[0].start);
for(i=0;i<=j;i++)
{
m = (s1[i].end - s1[i].start);
for(k=0;k<m/2;k++)
printf("-");
printf("%s",s1[i].pname);
for(k=0;k<(m+1)/2;k++)
printf("-");
printf("%d",s1[i].end);
```

```
}
}
int main()
{
accept_info();
sort();
rr();
print_output();
print_gantt_chart();
return 0;
}
```

Q.3. Oral/Viva [05]

----------Slip No-16----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.1 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. [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<unistd.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(NULL);
else
{
```

```
execvp("demo.sh",args);
}
}
}
return 0;
}
```

OUTPUT:-

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit typeline.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc typeline.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$typeline +1 demo.sh
hello!
myshell$typeline -1 demo.sh
wite a c program that behaves like a shell which display the
command prompt myshells.
myshell$typeline a demo.sh
hello!
wite a c program that behaves like a shell which display the
command prompt myshells.
myshell$^c
```

^c (control+c) use to exit from myshell prompt

*create shell file by using gedit demo.sh

demo.sh

```
hello!
```

wite a c program that behaves like a shell which display the command prompt myshells

Q.2 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 [15]

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct process_info
{
char pname[20];
int at,bt,ct,bt1;
struct process_info *next;
}NODE;
int n;
```

```
NODE *first,*last;
void accept_info()
{
NODE *p;
int i;
printf("Enter no.of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p = (NODE*)malloc(sizeof(NODE));
printf("Enter process name:");
scanf("%s",p->pname);
printf("Enter arrival time:");
scanf("%d",&p->at);
printf("Enter first CPU burst time:");
scanf("%d",&p->bt);
p->bt1 = p->bt;
p->next = NULL;
if(first==NULL)
first=p;
else
last->next=p;
last = p;
}
```

```
}
void print_output()
{
NODE *p;
float avg_tat=0,avg_wt=0;
printf("pname\tat\tbt\tct\ttat\twt\n");
p = first;
while(p!=NULL)
{
int tat = p->ct-p->at;
int wt = tat-p->bt;
avg_tat+=tat;
avg_wt+=wt;
printf("%s\t%d\t%d\t%d\t%d\t%d\n",
p->pname,p->at,p->bt,p->ct,tat,wt);
p=p->next;
}
printf("Avg TAT=%f\tAvg WT=%f\n",
avg_tat/n,avg_wt/n);
}
void print_input()
{
NODE *p;
p = first;
```

```
printf("pname\tat\tbt\n");
while(p!=NULL)
{
printf("%s\t%d\t%d\n",
p->pname,p->at,p->bt1);
p = p->next;
}
}
void sort()
{
NODE *p,*q;
int t;
char name[20];
p = first;
while(p->next!=NULL)
{
q=p->next;
while(q!=NULL)
{
if(p->at > q->at)
{
strcpy(name,p->pname);
strcpy(p->pname,q->pname);
strcpy(q->pname,name);
```

```
t = p->at;
p->at = q->at;
q->at = t;
t = p->bt;
p->bt = q->bt;
q->bt = t;
t = p->ct;
p->ct = q->ct;
q->ct = t;
t = p->bt1;
p->bt1 = q->bt1;
q->bt1 = t;
}
q=q->next;
}
p=p->next;
}
}
int time;
NODE * get_sjf()
{
NODE *p,*min_p=NULL;
int min=9999;
p = first;
```

```
while(p!=NULL)
{
if(p->at<=time && p->bt1!=0 &&
p->bt1<min)
{
min = p->bt1;
min_p = p;
}
p=p->next;
}
return min_p;
}
struct gantt_chart
{
int start;
char pname[30];
int end;
}s[100],s1[100];
int k;
void sjfnp()
{
int prev=0,n1=0;
NODE *p;
while(n1!=n)
```

```
{
p = get_sjf();
if(p==NULL)
{
time++;
s[k].start = prev;
strcpy(s[k].pname,"*");
s[k].end = time;
prev = time;
k++;
}
else
{
time+=p->bt1;
s[k].start = prev;
strcpy(s[k].pname, p->pname);
s[k].end = time;
prev = time;
k++;
p->ct = time;
p->bt1 = 0;
n1++;
}
print_input();
```

```
sort();
}
}
void print_gantt_chart()
{
int i,j,m;
s1[0] = s[0];
for(i=1,j=0;i<k;i++)
{
if(strcmp(s[i].pname,s1[j].pname)==0)
s1[j].end = s[i].end;
else
s1[++j] = s[i];
}
printf("%d",s1[0].start);
for(i=0;i<=j;i++)
{
m = (s1[i].end - s1[i].start);
for(k=0;k<m/2;k++)
printf("-");
printf("%s",s1[i].pname);
for(k=0;k<(m+1)/2;k++)
printf("-");
printf("%d",s1[i].end);
```

```
}
}
int main()
{
accept_info();
sort();
sjfnp();
print_output();
print_gantt_chart();
return 0;
}
```

Q.3. Oral/Viva [05]

----------Slip No-17 ----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.1 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”.
[15]

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
#include<unistd.h>
int main(int argc,int argv[])
{
pid_t pid=fork();
if(pid==0)
{
printf("\n i am child process");
printf("\n process id=%d\n",getpid());
return 0;
}
else if(pid>0)
{
printf("\n i am parent process");
printf("\n process id =%d",getpid());
}
```

```
else
{
printf("\n unable to create child process...");
}
return 0;
}
```

**********output**********

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc setA1.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
i am parent process
process id =3787
i am child process
process id=3788
```

Q.2 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. [15]

preemptive priority scheduling algorithm

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct process_info
{
char pname[20];
int at,bt,ct,bt1,p;
struct process_info *next;
}NODE;
int n;
NODE *first,*last;
void accept_info()
{
NODE *p;
int i;
printf("Enter no.of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p = (NODE*)malloc(sizeof(NODE));
printf("Enter process name:");
scanf("%s",p->pname);
printf("Enter arrival time:");
```

```
scanf("%d",&p->at);
printf("Enter first CPU burst time:");
scanf("%d",&p->bt);
printf("Enter priority:");
scanf("%d",&p->p);
p->bt1 = p->bt;
p->next = NULL;
if(first==NULL)
first=p;
else
last->next=p;
last = p;
}
}
void print_output()
{
NODE *p;
float avg_tat=0,avg_wt=0;
printf("pname\tat\tbt\tp\ttct\ttat\twt\n");
p = first;
while(p!=NULL)
{
int tat = p->ct-p->at;
int wt = tat-p->bt;
```

```
avg_tat+=tat;
avg_wt+=wt;
printf("%s\t%d\t%d\t%d\t%d\t%d\t%d\n",
p->pname,p->at,p->bt,p->p,p->ct,tat,wt);
p=p->next;
}
printf("Avg TAT=%f\tAvg WT=%f\n",
avg_tat/n,avg_wt/n);
}
void print_input()
{
NODE *p;
p = first;
printf("pname\tat\tbt\tp\n");
while(p!=NULL)
{
printf("%s\t%d\t%d\t%d\n",
p->pname,p->at,p->bt1,p->p);
p = p->next;
}
}
void sort()
{
NODE *p,*q;
```

```
int t;
char name[20];
p = first;
while(p->next!=NULL)
{
q=p->next;
while(q!=NULL)
{
if(p->at > q->at)
{
strcpy(name,p->pname);
strcpy(p->pname,q->pname);
strcpy(q->pname,name);
t = p->at;
p->at = q->at;
q->at = t;
t = p->bt;
p->bt = q->bt;
q->bt = t;
t = p->ct;
p->ct = q->ct;
q->ct = t;
t = p->bt1;
p->bt1 = q->bt1;
```

```
q->bt1 = t;
t = p->p;
p->p = q->p;
q->p = t;
}
q=q->next;
}
p=p->next;
}
}
int time;
NODE * get_p()
{
NODE *p,*min_p=NULL;
int min=9999;
p = first;
while(p!=NULL)
{
if(p->at<=time && p->bt1!=0 &&
p->p<min)
{
min = p->p;
min_p = p;
}
```

```
p=p->next;
}
return min_p;
}
struct gantt_chart
{
int start;
char pname[30];
int end;
}s[100],s1[100];
int k;
void pnp()
{
int prev=0,n1=0;
NODE *p;
while(n1!=n)
{
p = get_p();
if(p==NULL)
{
time++;
s[k].start = prev;
strcpy(s[k].pname,"*");
s[k].end = time;
```

```
prev = time;
k++;
}
else
{
time++;
s[k].start = prev;
strcpy(s[k].pname, p->pname);
s[k].end = time;
prev = time;
k++;
p->ct = time;
p->bt1--;
if(p->bt1==0) n1++;
}
print_input();
sort();
}
}
void print_gantt_chart()
{
int i,j,m;
s1[0] = s[0];
for(i=1,j=0;i<k;i++)
```

```
{
if(strcmp(s[i].pname,s1[j].pname)==0)
s1[j].end = s[i].end;
else
s1[++j] = s[i];
}
printf("%d",s1[0].start);
for(i=0;i<=j;i++)
{
m = (s1[i].end - s1[i].start);
for(k=0;k<m/2;k++)
printf("-");
printf("%s",s1[i].pname);
for(k=0;k<(m+1)/2;k++)
printf("-");
printf("%d",s1[i].end);
}
}
int main()
{
accept_info();
sort();
pnp();
print_output();
```

```
print_gantt_chart();
return 0;
}
```

Q.3. Oral/Viva [05]

----------Slip No-18----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.1 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 using

nice () system call. [15]

```
#include<stdio.h>
#include<stdlib.h>
#include<sys/types.h>
int main()
```

```
{
int pid, retnice;
printf(“press DEL to stop process \n”);
pid=fork();
for(;;)
{
if(pid==0)
{
retnice=nice(-5);
printf(“\n child gets higher CPU priority %d”,retnice);
sleep(1);
}
else
{
retnice=nice(4);
printf(“ \n parent gets lower CPU priority %d”,retnice);
sleep(1);
}
}
return 0; }
```

OUTPUT:

```
Press DEL to stop process.
parent gets lower CPU priority -6
child gets higher CPU priority -15
```

parent gets lower CPU priority -2

child gets higher CPU priority -20

parent gets lower CPU priority -1

Q.2 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. [15]

non preemptive priority

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct process_info
{
char pname[20];
int at,bt,ct,bt1,p;
struct process_info *next;
}NODE;
```

```
int n;
NODE *first,*last;
void accept_info()
{
NODE *p;
int i;
printf("Enter no.of process:");
scanf("%d",&n);
for(i=0;i<n;i++)
{
p = (NODE*)malloc(sizeof(NODE));
printf("Enter process name:");
scanf("%s",p->pname);
printf("Enter arrival time:");
scanf("%d",&p->at);
printf("Enter first CPU burst time:");
scanf("%d",&p->bt);
printf("Enter priority:");
scanf("%d",&p->p);
p->bt1 = p->bt;
p->next = NULL;
if(first==NULL)
first=p;
else
```

```
last->next=p;
last = p;
}
}
void print_output()
{
NODE *p;
float avg_tat=0,avg_wt=0;
printf("pname\tat\tbt\tp\ttct\ttat\twt\n");
p = first;
while(p!=NULL)
{
int tat = p->ct-p->at;
int wt = tat-p->bt;
avg_tat+=tat;
avg_wt+=wt;
printf("%s\t%d\t%d\t%d\t%d\t%d\t%d\n",
p->pname,p->at,p->bt,p->p,p->ct,tat,wt);
p=p->next;
}
printf("Avg TAT=%f\tAvg WT=%f\n",
avg_tat/n,avg_wt/n);
}
void print_input()
```

```
{
NODE *p;
p = first;
printf("pname\tat\tbt\tp\n");
while(p!=NULL)
{
printf("%s\t%d\t%d\t%d\n",
p->pname,p->at,p->bt1,p->p);
p = p->next;
}
}
void sort()
{
NODE *p,*q;
int t;
char name[20];
p = first;
while(p->next!=NULL)
{
q=p->next;
while(q!=NULL)
{
if(p->at > q->at)
{
```

```
strcpy(name,p->pname);
strcpy(p->pname,q->pname);
strcpy(q->pname,name);
t = p->at;
p->at = q->at;
q->at = t;
t = p->bt;
p->bt = q->bt;
q->bt = t;
t = p->ct;
p->ct = q->ct;
q->ct = t;
t = p->bt1;
p->bt1 = q->bt1;
q->bt1 = t;
t = p->p;
p->p = q->p;
q->p = t;
}
q=q->next;
}
p=p->next;
}
}
```

```
int time;
NODE * get_p()
{
NODE *p,*min_p=NULL;
int min=9999;
p = first;
while(p!=NULL)
{
if(p->at<=time && p->bt1!=0 &&
p->p<min)
{
min = p->p;
min_p = p;
}
p=p->next;
}
return min_p;
}
struct gantt_chart
{
int start;
char pname[30];
int end;
}s[100],s1[100];
```

```
int k;
void pnp()
{
int prev=0,n1=0;
NODE *p;
while(n1!=n)
{
p = get_p();
if(p==NULL)
{
time++;
s[k].start = prev;
strcpy(s[k].pname,"*");
s[k].end = time;
prev = time;
k++;
}
else
{
time+=p->bt1;
s[k].start = prev;
strcpy(s[k].pname, p->pname);
s[k].end = time;
prev = time;
```

```
k++;
p->ct = time;
p->bt1 = 0;
n1++;
}
print_input();
sort();
}
}
void print_gantt_chart()
{
int i,j,m;
s1[0] = s[0];
for(i=1,j=0;i<k;i++)
{
if(strcmp(s[i].pname,s1[j].pname)==0)
s1[j].end = s[i].end;
else
s1[++j] = s[i];
}
printf("%d",s1[0].start);
for(i=0;i<=j;i++)
{
m = (s1[i].end - s1[i].start);
```

```
for(k=0;k<m/2;k++)
printf("-");
printf("%s",s1[i].pname);
for(k=0;k<(m+1)/2;k++)
printf("-");
printf("%d",s1[i].end);
}
}
int main()
{
accept_info();
sort();
pnp();
print_output();
print_gantt_chart();
return 0;
}
```

Q.3. Oral/Viva [05]

----------Slip No-19----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.1 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()).

[15]

Ans----->

```
#include<stdio.h>
#include<sys/types.h>
#include<unistd.h>
int main ()
{
int pid=fork();
if(pid>0)
{
printf("\n parent process");
printf("process id=%d \n",getpid());
}
else if( pid==0)
{
```

```
printf(“\n child process” );
printf(“process id=%d\n”, getpid());
printf(“parent ID=%d\n”,getppid());
Sleep(10);
printf(“\n child process” );
printf(“process id=%d\n”, getpid());
printf(“parent ID=%d\n”,getppid());
}
Else
{
Printf(“failed to create child process..”);
}
Return 0;
}
```

OUTPUT:

```
Parent process
Process id=10669
Child process
Process id=10675
Parent id =22641
```

Q.2 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, [15]

```
//program for OTP……
#include<stdio.h>
int main()
{
   int no_of_frames, no_of_pages, frames[10], pages[30], temp[10], flag1, flag2, flag3, i, j, k, pos, max, faults = 0;
   printf("Enter number of frames: ");
   scanf("%d", &no_of_frames);

   printf("Enter number of pages: ");
   scanf("%d", &no_of_pages);

   printf("Enter page reference string: ");

   for(i = 0; i < no_of_pages; ++i)
{
      scanf("%d", &pages[i]);
   }
```

```
   for(i = 0; i < no_of_frames; ++i)
{
      frames[i] = -1;
   }

   for(i = 0; i < no_of_pages; ++i)
{
      flag1 = flag2 = 0;

      for(j = 0; j < no_of_frames; ++j)
{
         if(frames[j] == pages[i])
{
               flag1 = flag2 = 1;
               break;
            }
      }

      if(flag1 == 0){
         for(j = 0; j < no_of_frames; ++j)
{
            if(frames[j] == -1)
{
               faults++;
```

```
            frames[j] = pages[i];
            flag2 = 1;
            break;
          }
        }
      }

      if(flag2 == 0)
{
       flag3 =0;

        for(j = 0; j < no_of_frames; ++j)
{
         temp[j] = -1;

         for(k = i + 1; k < no_of_pages; ++k)
{
         if(frames[j] == pages[k])
{
         temp[j] = k;
         break;
         }
         }
        }
```

```
        for(j = 0; j < no_of_frames; ++j)
{
         if(temp[j] == -1)
{
         pos = j;
         flag3 = 1;
         break;
         }
        }

        if(flag3 ==0)
{
         max = temp[0];
         pos = 0;
         for(j = 1; j < no_of_frames; ++j)
{
         if(temp[j] > max)
{
         max = temp[j];
         pos = j;
         }
         }
        }
```

```
frames[pos] = pages[i];
faults++;
        }
        printf("\n");
        for(j = 0; j < no_of_frames; ++j)
{
            printf("%d\t", frames[j]);
        }
    }

    printf("\n\nTotal Page Faults = %d", faults);
    return 0;
}
```

Q.3. Oral/Viva [05]

----------Slip No-20----------

SAVITRIBAI PHULE PUNE UNIVERSITY

T.Y.B.Sc.(Computer Science) Practical Examination, March/October

(2019 Pattern)

CS-357 Lab Course-I Operating System-I

Duration: 3 Hrs Max. Marks: 35

Q.1 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. [15]

Program:

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
void bubblesort()
{
int i,j,a[50],n,temp;
printf("\n bubble sort \n");
printf("enter size of array\t");
scanf("%d",&n);
printf("\n enter %d element",n);
for(i=0;i<n;i++)
{
scanf("%d",&a[i]);
```

```
}
for(i=0;i<n;i++)
{
for(j=i+1;j<n;j++)
{
if(a[i]>a[j])
{
int temp;
temp=a[i];
a[i]=a[j];
a[j]=temp;
}
}
}
printf("\n sorted array is \t");
for(i=0;i<n;i++)
{
printf("%d\t",a[i]);
}
}
int insertionsort()
{
int n,i,j,temp;
int arr[64];
```

```
printf("enter number of element");
scanf("%d",&n);
printf("enter %d element\n",n);
for(i=0;i<n;i++)
{
scanf("%d",&arr[i]);
}
for(i=1;i<=n-1;i++)
{
j=i;
while(j>0&&arr[j-1]>arr[j])
{
temp=arr[j];
arr[j]=arr[j-1];
arr[j-1]=temp;
j--;
}
}
printf("sorted list in ascending order\n");
for(i=0;i<=n-1;i++)
{
printf("%d\n",arr[i]);
}
return 0;
```

```
}
int main()
{
if(fork()==0)
{insertionsort();
}
else
{
wait(NULL);
bubblesort();
}
return 0;
}
**********output**********
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc avi.chp@hp-HP-EliteDesk-800-G2-SFF:~$
./a.out
enter number of element 4
enter 4 element
23
11
45
33
sorted list in ascending order
```

11

23

33

45

bubble sort

enter size of array 4

enter 4 element

23

11

45

33

sorted array is

11

23

33

45

Q.2 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. [15]

PROGRAM:-

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include<sys/wait.h>
#include<unistd.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(NULL);
else
{
execvp("demo.h",args);
}
}
}
return 0;
}
```

OUTPUT:=

```
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gedit count.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ gcc count.c
hp@hp-HP-EliteDesk-800-G2-SFF:~$ ./a.out
myshell$count c demo.sh
No.of characters:248
myshell$count w demo.sh
No.of words:42
myshell$count l demo.sh
No.of lines:7
myshell$^C
^c (control+c) use to exit from myshell prompt
```

*create shell file by using gedit demo.sh

demo.sh

hell0 !

wite a c program that behaves like a shell which display the command prompt myshells.

program logic:

main function will execute and display the command prompt as $.

accept the commaand at $ prompt from user.

system program.

system program

Q.3. Oral/Viva [05]