**M.Sc. (Computer Science) Semester-II**

**SUBJECT: Advanced Operating System Practical Slips**

**20 Marks Program**

**Slip No.3**

**B).Write a C program to create ‘n’ child processes. When all ‘n’ child processes terminates, Display total cumulative time children spent in user and kernel mode.**

**Soln:**

#include<sys/types.h>

#include<sys/wait.h>

#include<unistd.h>

#include<time.h>

#include<sys/times.h>

#include<stdio.h>

#include<stdlib.h>

int main(void)

{

int i, status;

pid\_t pid;

time\_t currentTime;

struct tms cpuTime;

if((pid = fork())==-1) //start child process

{

perror("\nfork error");

exit(EXIT\_FAILURE);

}

else if(pid==0) //child process

{

time(&currentTime);

printf("\nChild process started at %s",ctime(&currentTime));

for(i=0;i<5;i++)

{

printf("\nCounting= %dn",i); //count for 5 seconds

sleep(1);

}

time(&currentTime);

printf("\nChild process ended at %s",ctime(&currentTime));

exit(EXIT\_SUCCESS);

}

else

{ //Parent process

time(&currentTime); // gives normal time

printf("\nParent process started at %s ",ctime(&currentTime));

if(wait(&status)== -1) //wait for child process

perror("\n wait error");

if(WIFEXITED(status))

printf("\nChild process ended normally");

else

printf("\nChild process did not end normally");

if(times(&cpuTime)<0) //Get process time

perror("\nTimes error");

else

{ // \_SC\_CLK\_TCK: system configuration time: seconds clock tick

printf("\nParent process user time= %fn",((double)

cpuTime.tms\_utime));

printf("\nParent process system time = %fn",((double)

cpuTime.tms\_stime));

printf("\nChild process user time = %fn",((double)

cpuTime.tms\_cutime));

printf("\nChild process system time = %fn",((double)

cpuTime.tms\_cstime));

}

time(&currentTime);

printf("\nParent process ended at %s",ctime(&currentTime));

exit(EXIT\_SUCCESS);

}

}

**Slip No.6**

**B) Write a C program that behaves like a shell (command interpreter). It has its own prompt say “NewShell$”. Any normal shell command is executed from your shell by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following command.**

**i) list f<dirname> - print name of all files in directory**

**ii) list n <dirname> - print number of all entries**

**iii) list i<dirname> - print name and inode of all files**

**Soln:**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<dirent.h>

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

int pid;

void list(char t2,char \*t3)

{

DIR \*dir;

struct dirent \*entry;

int cnt=0;

dir=opendir(t3);

if (dir==NULL)

{

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

return;

}

switch(t2)

{

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

{

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

}

break;

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

cnt++;

printf("Total No of Entries: %d\n",cnt);

break;

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

{

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

}

break;

default : printf("Invalid argument");

}

closedir(dir);

}

main()

{

while(1)

{

printf("myshell$");

fflush(stdin);

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

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

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

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

fgets(buff,80,stdin);

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

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

exit(0);

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

list(t2[0],t3);

else

{

pid=fork();

if(pid<0)

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

else if(pid==0)

{

execlp("/bin",NULL);

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

exit(0);

system(buff);

}

else

{

wait(NULL);

exit(0);

}

}

}

}

**Slip No.8**

**B) Write a C program which receives file names as command line arguments and display those filenames in ascending order according to their sizes.**

**(e.g $ a.out a.txt b.txt c.txt, …)**

**Soln:**

#include<stdio.h>

#include<dirent.h>

#include<string.h>

#include<sys/stat.h>

#include<time.h>

#include<stdlib.h>

structfilelist

{

charfname[100];

intfsize;

};

int main(intargc,char \*argv[])

{

DIR \*dp;

inti,j,k;

structdirent \*ep;

struct stat sb;

charmon[100];

structfilelist f1[100],temp;

j=0;

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

{

dp=opendir("./");

if (dp!=NULL)

{

while(ep=readdir(dp))

{

if((strcmp(ep->d\_name,argv[i]))==0)

{

stat(ep->d\_name,&sb);

strcpy(f1[j].fname,ep->d\_name);

f1[j].fsize=sb.st\_size;

j++;

break;

}

}

}

(void)closedir(dp);

}

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

{

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

{

if(f1[i].fsize< f1[k].fsize)

{

temp=f1[k];

f1[k]=f1[i];

f1[i]=temp;

}

}

}

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

{

printf("%s\t%d\n",f1[i].fname,f1[i].fsize);

}

return 0;

}

**Slip No.9**

1. **Write a C program that will only list all subdirectories in alphabetical order from current directory.**

**Soln:**

#include <stdio.h>

#include <stdlib.h>

#include <dirent.h>

int

main(void)

{

struct dirent \*\*namelist;

int n;

int i=0;

n = scandir(".", &namelist, 0, alphasort);

if (n < 0)

perror("scandir");

else {

while (i<n) {

printf("%s\n", namelist[i]->d\_name);

free(namelist[i]);

i++;

}

free(namelist);

}

}

**Slip No.11**

**B) Write a C program that behaves like a shell (command interpreter). It has its own prompt say “NewShell$”. Any normal shell command is executed from your shell by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following command.**

**i) count c <filename> - print number of characters in file**

**ii) count w <filename> - print number of words in file**

**iii) count l <filename> - print number of lines in file**

**Soln:**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

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

FILE \*fp;

int pid;

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

{

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

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

printf("File not found");

else

{

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

{

if(ch==' ')

wordcount++;

else if(ch=='\n')

{

linecount++;

wordcount++;

}

else

charcount++;

}

fclose(fp);

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

printf("The total no. of characters :%d\n",charcount);

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

printf("The total no. of words :%d\n",wordcount);

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

printf("The total no. of lines :%d\n",linecount);

else

printf("Command not found");

}

}

main()

{

while(1)

{

printf("myshell$");

fflush(stdin);

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

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

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

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

fgets(buff,80,stdin);

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

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

exit(0);

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

count(t2,t3);

else

{

pid=fork();

if(pid<0)

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

else if(pid==0)

{

execlp("/bin",NULL);

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

exit(0);

system(buff);

}

else

{

wait(NULL);

exit(0);

}

}

}

}

**Slip No.13**

**B) Write a C program that behaves like a shell (command interpreter). It has its own prompt say “NewShell$”. Any normal shell command is executed from your shell by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following command.**

**i) typeline +10 <filename> - print first 10 lines of file**

**ii) typeline -20 <filename> - print last 20 lines of file**

**iii) typeline a <filename> - print all lines of file**

**Soln:**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h>

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

FILE \*fp;

int pid;

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

{

int i,n,count=0,num;

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

printf("File not found\n");

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

{

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

printf("%c",ch);

fclose(fp);

return;

}

n=atoi(t2);

if(n>0)

{

i=0;

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

{

if(ch=='\n')

i++;

if(i==n)

break;

printf("%c",ch);

}

printf("\n");

}

else

{

count=0;

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

if(ch=='\n')

count++;

fseek(fp,0,SEEK\_SET);

i=0;

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

{

if(ch=='\n')

i++;

if(i==count+n-1)

break;

}

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

printf("%c",ch);

}

fclose(fp);

}

main()

{

while(1)

{

printf("myshell$");

fflush(stdin);

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

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

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

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

fgets(buff,80,stdin);

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

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

exit(0);

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

typeline(t2,t3);

else

{

pid=fork();

if(pid<0)

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

else if(pid==0)

{

execlp("/bin",NULL);

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

exit(0);

system(buff);

}

else

{

wait(NULL);

exit(0);

}

}

}

}

**Slip No.14**

**B) Write a C program that behaves like a shell (command interpreter). It has its own prompt say “NewShell$”.Any normal shell command is executed from your shell by starting a child process to execute the system program corresponding to the command. It should additionally interpret the following command.**

**i) search f <pattern><filename> - search first occurrence of pattern in filename**

**ii) search c <pattern><filename> - count no. of occurrences of pattern in filename**

**iii) search a <pattern><filename> - search all occurrences of pattern in filename**

**Soln:**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h>

char \*buff,\*t1,\*t2,\*t3,\*t4,ch;

FILE \*fp;

int pid;

void search(char \*t2,char \*t3,char \*t4)

{

int i=1,count=0;

char \*p;

if((fp=fopen(t4,"r"))==NULL)

printf("File not found\n");

else

{

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

{

while(fgets(buff,80,fp))

{

if((strstr(buff,t3))!=NULL)

{

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

break;

}

}

i++;

}

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

{

while(fgets(buff,80,fp))

{

if((strstr(buff,t3))!=NULL)

{

count++;

}

}

printf("No of occurences of %s= %d\n",t3,count);

}

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

{

while(fgets(buff,80,fp))

{

if((strstr(buff,t3))!=NULL)

{

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

}

i++;

}

}

else

printf("Command not found\n");

fclose(fp);

}

}

main()

{

while(1)

{

printf("myshell$");

fflush(stdin);

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

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

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

t4=(char \*)malloc(80);

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

fgets(buff,80,stdin);

sscanf(buff,"%s %s %s %s",t1,t2,t3,t4);

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

exit(0);

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

search(t2,t3,t4);

else

{

pid=fork();

if(pid<0)

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

else if(pid==0)

{

execlp("/bin",NULL);

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

exit(0);

system(buff);

}

else

{

wait(NULL);

exit(0);

}

}

}}

**Slip No.15**

1. **Write a C program which creates a child process and child process catches a signal SIGHUP, SIGINT and SIGQUIT. The Parent process send a SIGHUP or SIGINT signal after every 3 seconds, at the end of 15 second parent send SIGQUIT signal to child and child terminates by displaying message "My Papa has Killed me!!!”.**

**Soln:**

#include<stdio.h>

#include<signal.h>

#include<stdlib.h>

void sighup();

void sigint();

void sigquit();

main()

{

int pid,i,j,k;

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

{

perror("fork");

exit(1);

}

if ( pid == 0)

{

signal(SIGHUP,sighup);

signal(SIGINT,sigint);

signal(SIGQUIT,sigquit);

for(;;);

}

else

{

j=0;

for(i=1;i<=5;i++)

{

j++;

printf("PARENT: sending SIGHUP Signal : %d\n",j);

kill(pid,SIGHUP);

sleep(3);

printf("PARENT: sending signal : %d\n",j);

kill (pid,SIGINT);

sleep(3);

}

sleep(3);

printf("Parent sending SIGQUIT\n");

kill(pid,SIGQUIT);

}

}

void sighup()

{

signal(SIGHUP,sighup);

printf("Child: I have received sighup\n");

}

void sigint()

{

signal(SIGINT,sigint);

printf("Child: I have received sighINT\n");

}

void sigquit()

{

printf("My daddy has killed me\n");

exit(0);

}

**Slip No.17**

1. **Write a C program to display all the files from current directory which are created in a particular month.**

**Soln:**

#include<stdio.h>

#include<dirent.h>

#include<string.h>

#include<sys/stat.h>

#include<time.h>

#include<stdlib.h>

int main(intargc, char \*argv[])

{

char in[100],st[100],\*ch,\*ch1,c,buff[512];

DIR \*dp;

int i;

structdirent \*ep;

struct stat sb;

charmon[100];

dp=opendir("./");

if (dp != NULL)

{

while(ep =readdir(dp))

{

if(stat(ep->d\_name,&sb) == -1)

{

perror("stat");

exit(EXIT\_SUCCESS);

}

strcpy(mon,ctime(&sb.st\_ctime));

ch=strtok(mon," ");

ch=strtok(NULL,",");

ch1=strtok(ch," ");

if((strcmp(ch1,argv[1]))==0)

{

printf("%s\t\t%s",ep->d\_name,ctime(&sb.st\_ctime));

}

}

(void)closedir(dp);

}

return 0;

}

**Slip No.18**

**B) Write a C program to implement the following unix/linux command (use fork, pipe and exec system call). Your program should block the signal Ctrl-C and Ctrl-\ signal during the execution.**

**ls –l | wc –l**

**Soln:**

#include <stdio.h>

#include <stdlib.h>

#include <fcntl.h>

#include<errno.h>

#include<sys/wait.h>

#include <unistd.h>

int main(){

// array of 2 size a[0] is for

// reading and a[1] is for

// writing over a pipe

int a[2];

// using pipe for inter process communication

pipe(a);

if(!fork())

{

// closing normal stdout

close(1);

// making stdout same as a[1]

dup(a[1]);

// closing reading part of pipe

// we don't need of it at this time

close(a[0]);

// executing ls

execlp("ls","ls",NULL);

}

else

{

// closing normal stdin

close(0);

// making stdin same as a[0]

dup(a[0]);

// closing writing part in parent,

// we don't need of it at this time

close(a[1]);

// executing wc

execlp("wc","wc",NULL);

}

}