**AOS SLIPs**

**Slip 1**

1. **Write a C program to find whether a given file is present in current directory or not.**

Solution:

#include<stdio.h>

#include<dirent.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

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

{

DIR \*dirptr;

struct dirent \*entry;

int found = 0;

char curDir[20];

getcwd(curDir,20);

if(argc<2)

{

printf("\n Insufficient arguments\n");

exit(1);

}

dirptr = opendir(curDir);

while((entry = readdir(dirptr))!=NULL)

{

if(strcmp(entry->d\_name,argv[1])==0)

{

printf("\nFile %s present in current directory\n",argv[1]);

found=1;

break;

}

}

closedir(dirptr);

if(found==0)

printf("\nnFile %s not present in current directory\n",argv[1]);

return 0;

}

1. **Write a C program which blocks SIGOUIT signal for 5 seconds. After 5 second process checks any occurrence of quit signal during this period, if so, it unblock the signal. Now another occurrence of quit signal terminates the program. (Use sigprocmask() and sigpending() )**

Solution

#include<stdio.h>

#include<signal.h>

static void sig\_quit(int signo)

{

printf("\nCaught SIG\_QUIT");

if(signal(SIGQUIT,SIG\_DFL)==SIG\_ERR)

printf("\ncan't reset SIGQUIT");

}

int main()

{

sigset\_t newmask, oldmask, pendmask;

if(signal(SIGQUIT,sig\_quit)==SIG\_ERR)

printf("\ncant catch sigquit");

sigemptyset(&newmask);

sigaddset(&newmask, SIGQUIT);

if(sigprocmask(SIG\_BLOCK, &newmask, &oldmask)<0)

printf("\nsigblock error");

sleep(5);

printf("old signal set : %8.8ld.\n",oldmask);

if(sigpending(&pendmask)<0)

printf("\nsig-pending error");

printf("pending signal set : %8.8ld.\n",pendmask);

if(sigismember(&pendmask,SIGQUIT))

printf("\nSIGQUIT pending");

if(sigprocmask(SIG\_SETMASK, &oldmask, NULL)<0)

printf("\nsig\_setmask error");

printf("\nSIGQUIT unblocked");

printf("\nhello\n");

sleep(10);

printf("\nhello\n");

return 0;

}

**Slip 2**

1. **Write a C program that a string as an argument and return all the files that begins with that name in the current directory. For example > ./a.out foo will return all file names that begins with foo.**

Solution:

#include<stdio.h>

#include<dirent.h>

#include<stdlib.h>

#include<string.h>

#include<unistd.h>

int startsWith(char \*fileName, char \*start)

{

int i,len = strlen(start);

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

{

if(fileName[i]==start[i])

continue;

break;

}

if(i==len)

return 0;

return 1;

}

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

{

DIR \*dirptr;

struct dirent \*entry;

char curDir[20];

getcwd(curDir,20);

if(argc<2)

{

printf("\n Insufficient arguments\n");

exit(1);

}

dirptr = opendir(curDir);

while((entry = readdir(dirptr))!=NULL)

{

if(startsWith(entry->d\_name,argv[1])==0)

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

}

closedir(dirptr);

return 0;

}

1. **Write a C program to demonstrates the different behavior that can be seen with**

**automatic, global, register, static and volatile variables (Use setjmp() and longjmp() system**

**call).**

**Solution :**

#include<setjmp.h>

#include<stdio.h>

#include<stdlib.h>

static void f1(int,int,int,int);

static void f2(void);

static jmp\_buf jmpbuffer;

int globval;

int main()

{

int autoval;

register int regval;

volatile int volval;

static int statval;

globval=1;autoval=2;regval=3;volval=4;statval=5;

if(setjmp(jmpbuffer)!=0)

{

printf("\nAfter longjmp:\n");

printf("\nGlobal value = %d\n",globval);

printf("\nAuto value = %d\n",autoval);

printf("\nRegister value = %d\n",regval);

printf("\nVolatile value = %d\n",volval);

printf("\nStatic value = %d\n",statval);

exit(0);

}

globval=95;autoval=96;regval=97;volval=98;statval=99;

f1(autoval,regval,volval,statval);

return 0;

}

static void f1(int i,int j,int k,int l)

{

printf("\nIn f1():\n");

printf("\nGlobal value = %d\n",globval);

printf("\nAuto value = %d\n",i);

printf("\nRegister value = %d\n",j);

printf("\nVolatile value = %d\n",k);

printf("\nStatic value = %d\n",l);

f2();

}

static void f2(void)

{

longjmp(jmpbuffer,1);

}

**Slip 3**

1. **Write a C program to find file properties such as inode number, number of hard link, File permissions, File size, File access and modification time and so on of a given file using stat() system call.**

**Solution:**

#include<sys/stat.h>

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<sys/types.h>

#include<pwd.h>

#include<grp.h>

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

{

struct stat s;

struct tm \*timeinfo;

struct passwd \*pw;

struct group\*gr;

char filetype,perm,\*date;

int i;

memset(&s,0,sizeof(s));

if(argc<2)

{

printf("Insufficient arguments\n");

exit(1);

}

printf("\nFile size \t inode\n");

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

{

printf("\n");

stat(argv[i],&s);

if((s.st\_mode & S\_IFMT)==S\_IFREG) filetype = 'R';

else if((s.st\_mode & S\_IFMT)==S\_IFSOCK) filetype = 'S';

else if((s.st\_mode & S\_IFMT)==S\_IFLNK) filetype = 'L';

else if((s.st\_mode & S\_IFMT)==S\_IFBLK) filetype = 'B';

else if((s.st\_mode & S\_IFMT)==S\_IFDIR) filetype = 'D';

else if((s.st\_mode & S\_IFMT)==S\_IFCHR) filetype = 'C';

else if((s.st\_mode & S\_IFMT)==S\_IFIFO) filetype = 'F';

printf("%s\t%ld\t%C\t%ld\t%ld",argv[i],s.st\_ino,filetype,s.st\_size,s.st\_nlink);

date = ctime(&s.st\_atime);

timeinfo = localtime(&s.st\_atime);

printf("\nmonth=%d\n",timeinfo->tm\_mon);

printf("\nFile access time = %s",date);

printf("\nFile access time = %s",ctime(&s.st\_mtime));

printf("\nFile access time = %s",ctime(&s.st\_ctime));

pw = getpwuid(s.st\_uid);

gr = getgrgid(s.st\_gid);

printf("\n user = %s",pw->pw\_name);

printf("\n group = %s",gr->gr\_name);

printf((s.st\_mode & S\_IRUSR)?"r":"-");

printf((s.st\_mode & S\_IWUSR)?"w":"-");

printf((s.st\_mode & S\_IXUSR)?"x":"-");

printf((s.st\_mode & S\_IRGRP)?"r":"-");

printf((s.st\_mode & S\_IWGRP)?"w":"-");

printf((s.st\_mode & S\_IXGRP)?"x":"-");

printf((s.st\_mode & S\_IROTH)?"r":"-");

printf((s.st\_mode & S\_IWOTH)?"w":"-");

printf((s.st\_mode & S\_IXOTH)?"x":"-");

}

return 0;

}

**Slip 4**

1. **Write a C program to find file properties such as inode number, number of hard link, File permissions, File size, File access and modification time and so on of a given file using fstat() system call**

Solution:

#include<sys/stat.h>

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<sys/types.h>

#include<pwd.h>

#include<grp.h>

#include<fcntl.h>

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

{

struct stat s;

struct tm \*timeinfo;

struct passwd \*pw;

struct group\*gr;

char filetype,perm,\*date;

memset(&s,0,sizeof(s));

if(argc<2)

{

printf("Insufficient arguments\n");

exit(1);

}

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

{

printf("\n");

int fd = open(argv[1],O\_RDONLY);

fstat(fd,&s);

if((s.st\_mode & S\_IFMT)==S\_IFREG) filetype = 'R';

else if((s.st\_mode & S\_IFMT)==S\_IFSOCK) filetype = 'S';

else if((s.st\_mode & S\_IFMT)==S\_IFLNK) filetype = 'L';

else if((s.st\_mode & S\_IFMT)==S\_IFBLK) filetype = 'B';

else if((s.st\_mode & S\_IFMT)==S\_IFDIR) filetype = 'D';

else if((s.st\_mode & S\_IFMT)==S\_IFCHR) filetype = 'C';

else if((s.st\_mode & S\_IFMT)==S\_IFIFO) filetype = 'F';

printf("\nfile%s\ninode=%ld\nfiletype=%c\nfilesize=%ld\nnumber of

links=%ld",argv[i],s.st\_ino,filetype,s.st\_size,s.st\_nlink);

date = ctime(&s.st\_atime);

timeinfo = localtime(&s.st\_atime);

printf("\nmonth=%d\n",timeinfo->tm\_mon);

printf("\nFile access time = %s",date);

printf("\nFile access time = %s",ctime(&s.st\_mtime));

printf("\nFile access time = %s",ctime(&s.st\_ctime));

pw = getpwuid(s.st\_uid);

gr = getgrgid(s.st\_gid);

printf("\n user = %s",pw->pw\_name);

printf("\n group = %s",gr->gr\_name);

printf((s.st\_mode & S\_IRUSR)?"r":"-");

printf((s.st\_mode & S\_IWUSR)?"w":"-");

printf((s.st\_mode & S\_IXUSR)?"x":"-");

printf((s.st\_mode & S\_IRGRP)?"r":"-");

printf((s.st\_mode & S\_IWGRP)?"w":"-");

printf((s.st\_mode & S\_IXGRP)?"x":"-");

printf((s.st\_mode & S\_IROTH)?"r":"-");

printf((s.st\_mode & S\_IWOTH)?"w":"-");

printf((s.st\_mode & S\_IXOTH)?"x":"-");

}

return 0;

}

1. **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**

Solution 

#include<stdio.h>

#include<unistd.h>

#include<signal.h>

static void sig\_handler(int signo)

{

if(signo == SIGINT)

printf("\nCaught SIG\_INT");

if(signo == SIGQUIT)

printf("\nCaught SIG\_INT");

if(signal(SIGINT,SIG\_DFL)==SIG\_ERR)

printf("\ncan't reset SIGINT");

if(signal(SIGQUIT,SIG\_DFL)==SIG\_ERR)

printf("\ncan't reset SIGQUIT");

}

int main()

{

sigset\_t newmask, oldmask, pendmask;

if(signal(SIGINT,sig\_handler)==SIG\_ERR)

printf("\ncant catch sigint");

if(signal(SIGQUIT,sig\_handler)==SIG\_ERR)

printf("\ncant catch sigquit");

sigemptyset(&newmask);

sigaddset(&newmask, SIGINT);

sigaddset(&newmask, SIGQUIT);

int pipefd[2],retstatus,pid;

retstatus = pipe(pipefd);

if(retstatus==-1)

{

printf("\nfailed to create pipe\n");

return 1;

}

pid=fork();

if(pid<0)

{

printf("\nfailed to create child\n");

return 2;

}

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

{

close(pipefd[0]); //close read end of pipe

close(1); // close std.output

dup(pipefd[1]);

execlp("ls","ls","-l",(char \*)0);

}

else // parent process

{

close(pipefd[1]); // close write end of pipe

close(0); // close std. input

dup(pipefd[0]);

if(sigprocmask(SIG\_BLOCK, &newmask, &oldmask)<0)

printf("\nsigblock error");

sleep(5);

if(sigpending(&pendmask)<0)

printf("\nsig-pending error");

if(sigismember(&pendmask,SIGINT))

printf("\nSIGINT pending");

if(sigismember(&pendmask,SIGQUIT))

printf("\nSIGQUIT pending");

if(sigprocmask(SIG\_SETMASK, &oldmask, NULL)<0)

printf("\nsig\_setmask error");

printf("\nSIGINT unblocked");

printf("\nSIGQUT unblocked");

sleep(10);

execlp("wc","wc","-l",(char \*)0);

}

return 0;

}

**Slip 5**

1. **Write a C program to create an unnamed pipe. The child process will write following three messages to pipe and parent process display it.**

**Message1 = “Hello World”**

**Message2 = “Hello SPPU”**

**Message3 = “Linux is Funny”**

**Solution:**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#define MSGSIZE 16

char \*msg1 = "Hello World";

char \*msg2 = "Hello SPPU";

char \*msg3 = "Linux is funny";

int main()

{

int fd[2],i;

char buff[MSGSIZE];

if(pipe(fd)<0)

exit(1);

write(fd[1],msg1,MSGSIZE);

write(fd[1],msg2,MSGSIZE);

write(fd[1],msg3,MSGSIZE);

for(i=0;i<3;i++)

{

read(fd[0],buff,MSGSIZE);

printf("\n%s",buff);

}

return 0;

}

**Slip 6**

1. **Write a C program to map a given file in memory and display the contain of mapped file in reverse.**

**Solution:**

#include<stdio.h>

#include<stdlib.h>

#include<sys/mman.h>

#include<unistd.h>

#include<sys/stat.h>

#include<sys/types.h>

#include<fcntl.h>

#include<string.h>

#include<malloc.h>

char \*addr;

int main()

{

int fd,len,stats;

char \*rev;

struct stat st;

memset(&st,0,sizeof(st));

fd = open("dip.txt",O\_RDONLY);

stats = fstat(fd,&st);

len = st.st\_size;

if((addr = mmap(NULL, len, PROT\_READ, MAP\_PRIVATE,fd,0))==MAP\_FAILED)

printf("\nError in mmap");

printf("\nmap=\n%s",addr); // display mapped file

printf("\nmapped file in reverse order\n");

rev = addr + strlen(addr);

while(rev != addr)

{

printf("%c",\*rev);

rev--;

}

printf("%c",\*rev);

return 0;

}

1. **Write a C program that behaves like a shell (command interpreter). It has its own promp say . 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**

**Solution**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h>

#include<sys/wait.h>

#include<dirent.h>

void separate\_tokens(char \*cmd,char \*tok[])

{

int i=0;

char \*p;

p=strtok(cmd," ");

puts(p);

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void list(char \*dirName,char param)

{

DIR \*dir;

int count=0;

struct dirent \*entry;

// struct stat buff;

if((dir=opendir(dirName))==NULL)

{

printf("\n\tDirectory %s notfound\n",dirName);

return;

}

switch(param)

{

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

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

break;

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

count++;

printf("\nTotal number of entries = %d\n",count);

break;

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

printf("\n%ld:%s",entry->d\_ino,entry-

>d\_name);

break;

}

}

int main()

{

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

int pid;

system("clear");

do

{

printf("\nNewShell$ ");

fgets(cmd,80,stdin);

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

separate\_tokens(cmd,args);

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

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

else

{

pid = fork();

if(pid > 0)

wait(0);

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

printf("\n Command %s not found\n",args[0]);

}

}while(1);

return 0;

}

**Slip 7**

1. **Write a C program to create a file with hole in it.**

**Solution:**

#include<stdio.h>

#include<unistd.h>

#include<fcntl.h>

#include<sys/stat.h>

#include<stdlib.h>

int main()

{

int fd;

char \*msg1="Welcome to";

char \*msg2 = "WELCOME TO";

fd = open("p1.txt",O\_CREAT | O\_WRONLY);

if(fd<0)

{

printf("\nfailed to create file\n");

exit(1);

}

chmod("p1.txt",0777);

write(fd,msg1,10);

lseek(fd,1024L,SEEK\_CUR);

write(fd,msg2,10);

close(fd);

return 0;

}

1. **Write a C program which display the information of a given file similar to given by the unix /linux command on current directory (l.e File Access permission, file name, file type, User id, group id, file size, file access and modified time and so on) ls –l DO NOT simply exec ls -l or system command from the program.**

**Solution**

#include<sys/stat.h>

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<sys/types.h>

#include<pwd.h>

#include<grp.h>

#include<dirent.h>

#include<unistd.h>

#include<fcntl.h>

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

{

DIR \*dirptr;

struct dirent \*entry;

char curDir[80];

getcwd(curDir,80);

printf("%s\n",curDir);

struct stat s;

struct tm \*timeinfo;

struct passwd \*pw;

struct group\*gr;

int fd;

char filetype,perm,\*date;

memset(&s,0,sizeof(s));

fd=open("a.txt",O\_CREAT | O\_WRONLY);

chmod("a.txt",0777);

close(1);

dup(fd);

dirptr = opendir(curDir);

while((entry = readdir(dirptr))!=NULL)

{

printf("\n");

stat(entry->d\_name,&s);

if((s.st\_mode & S\_IFMT)==S\_IFREG) filetype = '-';

else if((s.st\_mode & S\_IFMT)==S\_IFSOCK) filetype = 'S';

else if((s.st\_mode & S\_IFMT)==S\_IFLNK) filetype = 'L';

else if((s.st\_mode & S\_IFMT)==S\_IFBLK) filetype = 'B';

else if((s.st\_mode & S\_IFMT)==S\_IFDIR) filetype = 'D';

else if((s.st\_mode & S\_IFMT)==S\_IFCHR) filetype = 'C';

else if((s.st\_mode & S\_IFMT)==S\_IFIFO) filetype = 'F';

date = ctime(&s.st\_atime);

timeinfo = localtime(&s.st\_atime);

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

pw = getpwuid(s.st\_uid);

gr = getgrgid(s.st\_gid);

printf("%c",filetype);

printf((s.st\_mode & S\_IRUSR)?"r":"-");

printf((s.st\_mode & S\_IWUSR)?"w":"-");

printf((s.st\_mode & S\_IXUSR)?"x":"-");

printf((s.st\_mode & S\_IRGRP)?"r":"-");

printf((s.st\_mode & S\_IWGRP)?"w":"-");

printf((s.st\_mode & S\_IXGRP)?"x":"-");

printf((s.st\_mode & S\_IROTH)?"r":"-");

printf((s.st\_mode & S\_IWOTH)?"w":"-");

printf((s.st\_mode & S\_IXOTH)?"x":"-");

printf(" %ld %s %s %ld\t%s %s",s.st\_nlink,pw->pw\_name,pw-

>pw\_name,s.st\_size,date,entry->d\_name);

}

close(fd);

return 0;

}

**slip 8**

1. **Write a C program to get and set the resource limits such as files, memory associated**

**with a process.\*/**

**Solution:**

#include<stdio.h>

#include<stdlib.h>

#include<sys/time.h>

#include<sys/resource.h>

int main()

{

struct rlimit limit;

if(getrlimit(RLIMIT\_NPROC,&limit)<0)

printf("getrlimit error\n");

printf("No. of extant process = [%10ld][%10ld]\n",limit.rlim\_max,limit.rlim\_cur);

if(getrlimit(RLIMIT\_CPU,&limit)<0)

printf("getrlimit error\n");

printf("limit on amount of CPU time that process can consume =

[%ld][%ld]\n",limit.rlim\_max,limit.rlim\_cur);

if(getrlimit(RLIMIT\_DATA,&limit)<0)

printf("getrlimit error\n");

printf("max.size of process's data segment =

[%ld][%ld]\n",limit.rlim\_max,limit.rlim\_cur);

if(getrlimit(RLIMIT\_FSIZE,&limit)<0)

printf("getrlimit error\n");

printf("max. size in bytes of files that process may create =

[%ld][%ld]\n",limit.rlim\_max,limit.rlim\_cur);

if(getrlimit(RLIMIT\_LOCKS,&limit)<0)

printf("getrlimit error\n");

printf("limit on locks = [%ld][%ld]\n",limit.rlim\_max,limit.rlim\_cur);

if(getrlimit(RLIMIT\_MEMLOCK,&limit)<0)

printf("getrlimit error\n");

printf("max. no. of bytes of memory that can be locked in RAM =

[%ld][%ld]\n",limit.rlim\_max,limit.rlim\_cur);

if(getrlimit(RLIMIT\_MSGQUEUE,&limit)<0)

printf("getrlimit error\n");

printf("msg queue = [%ld][%ld]\n",limit.rlim\_max,limit.rlim\_cur);

return 0;

}

1. **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, ...) \*/**

**Solution**

#include<sys/stat.h>

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<sys/types.h>

#include<pwd.h>

#include<grp.h>

#include<dirent.h>

#include<unistd.h>

struct fileinfo

{

char fileName[20];

int size;

}files[20],temp;

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

{

struct stat s;

memset(&s,0,sizeof(s));

int i,j,n;

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

{

printf("\n");

stat(argv[i],&s);

strcpy(files[i-1].fileName,argv[i]);

files[i-1].size = s.st\_size;

}

n=i-1;

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

{

for(j=i+1;j<n;j++)

{

if(files[i].size > files[j].size)

{

temp = files[i];

files[i]=files[j];

files[j]=temp;

}

}

}

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

printf("\n%s\t%d",files[i].fileName,files[i].size);

return 0;

}

**Slip 9**

1. **Write a C program to display as well as resets the environment variable such as path,**

**home, root etc.**

**Solution:**

#include<stdio.h>

#include<stdlib.h>

int main()

{

char \*path = getenv("PATH");

if(path)

printf("\npath=%s\n",path);

else printf("\nvar not found\n");

char \*home = getenv("HOME");

if(home)

printf("\nhome=%s\n",home);

else printf("\nvar not found\n");

char \*shell = getenv("SHELL");

if(shell)

printf("\nshell=%s\n",shell);

else printf("\nvar not found\n");

setenv("HOME","/home/AOS",1);

home = getenv("HOME");

if(home)

printf("\nhome=%s\n",home);

else printf("\nvar not found\n");

return 0;

}

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

**Solution**

#include<sys/stat.h>

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<sys/types.h>

#include<pwd.h>

#include<grp.h>

#include<dirent.h>

#include<unistd.h>

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

{

DIR \*dirptr;

struct dirent \*entry;

char curDir[80];

getcwd(curDir,80);

printf("%s\n",curDir);

struct stat s;

char \*files[10],temp[10];

int i=0,j,n;

memset(&s,0,sizeof(s));

dirptr = opendir(curDir);

while((entry = readdir(dirptr))!=NULL)

{

stat(entry->d\_name,&s);

if((s.st\_mode & S\_IFMT)==S\_IFDIR)

{

files[i]=malloc(20);

strcpy(files[i++],entry->d\_name);

}

}

n=i-1;

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

{

for(j=i+1;j<n;j++)

{

if(strcmp(files[i],files[j])>0)

{

strcpy(temp,files[i]);

strcpy(files[i],files[j]);

strcpy(files[j],temp);

}

}

}

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

printf("\n%s",files[i]);

return 0;

}

**Slip 11**

**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 - print number of characters in file** ii) count w - print number of **words in file iii) count l - print number of lines in file**

**Solution:**

#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);

        }

    }

    }

}

/\*

 [root@localhost Desktop]# ./a.out

myshell$count c a.txt

The total no. of characters :36

myshell$count w a.txt

The total no. of words :10

myshell$count l a.txt

The total no. of lines :4

myshell$ls

a.c  a.out  a.txt  b.c  count.c  list.c  search.c  typeline.c

myshell$pause

\*/

**Slip 12**

1. **Write a C program to send SIGALRM signal by child process to parent process and parent process make a provision to catch the signal and display alarm is fired.(Use Kill, fork, signal and sleep system call)\*/**

**Solution:**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<signal.h>

#include<string.h>

#include<sys/wait.h>

static void my\_alarm(int signo)

{

printf("\n in signal handler");

alarm(1);

}

int main()

{

int i;

pid\_t pid;

signal(SIGALRM,my\_alarm);

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

printf("\nfork error");

if(pid==0)

{

printf("\n child");

alarm(2);

kill(getppid(),SIGALRM);

}

// alarm(2);

else

{

printf("\nparent");

for(i=1;;i++)

{

printf("\n inside main");

sleep(1);

}

}

return 0;

}

1. **Write a C program to display all the files from current directory and its subdirectory whose size is greater than ’n’ Bytes Where n is accepted from user through command line.**

**Solution**

#include<sys/stat.h>

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<sys/types.h>

#include<pwd.h>

#include<grp.h>

#include<dirent.h>

#include<unistd.h>

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

{

DIR \*dirptr,\*subdirptr;

struct dirent \*entry,\*subentry;

struct stat s;

memset(&s,0,sizeof(s));

int n = atoi(argv[1]);

char curDir[80];

getcwd(curDir,80);

if(argc<2)

{

printf("\n Insufficient arguments\n");

exit(1);

}

dirptr = opendir(curDir);

while((entry = readdir(dirptr))!=NULL)

{

stat(entry->d\_name,&s);

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

if(((s.st\_mode & S\_IFMT)==S\_IFREG) && s.st\_size > n)

printf("\n%s : %ld",entry->d\_name,s.st\_size);

if((s.st\_mode & S\_IFMT)==S\_IFDIR)

{

subdirptr = opendir(entry->d\_name);

while((subentry = readdir(subdirptr))!=NULL)

{

stat(subentry->d\_name,&s);

if(((s.st\_mode & S\_IFMT)==S\_IFREG) && s.st\_size > n)

printf("\n%s : %ld",subentry->d\_name,s.st\_size);

}

}

}

closedir(dirptr);

return 0;

}

**Slip 13**

**B) Write a C program that behaves like a shell (command interpreter). It has its own prompt**

**say . 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.**

**Solution**

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<string.h>

#include<sys/wait.h>

#include<dirent.h>

#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

void separate\_tokens(char \*cmd,char \*tok[])

{

int i=0;

char \*p;

p=strtok(cmd," ");

puts(p);

while(p!=NULL)

{

tok[i++]=p;

p=strtok(NULL," ");

}

tok[i]=NULL;

}

void typeline(char \*fileName,char\* count)

{

int handle,n,i=0,cnt=0;

char ch;

if((handle=open(fileName,O\_RDONLY))==-1)

{

printf("\n\file %s notfound\n",fileName);

return;

}

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

{

while((read(handle,&ch,1)!=0))

printf("%c",ch);

close(handle);

return;

}

n=atoi(count);

if(n>0)

{

while((read(handle,&ch,1)!=0))

{

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

if(i==n) break;

printf("%c",ch);

}

printf("\n");

close(handle);

return;

}

if(n<0)

{

while((read(handle,&ch,1)!=0))

if(ch=='\n') cnt++;

lseek(handle,0,SEEK\_SET);

while((read(handle,&ch,1)!=0))

{

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

if(i==(cnt+n))

break;

}

while((read(handle,&ch,1)!=0))

printf("%c",ch);

printf("\n");

close(handle);

return;

}

}

int main()

{

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

int pid;

system("clear");

do

{

printf("\nNewShell$ ");

fgets(cmd,80,stdin);

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

separate\_tokens(cmd,args);

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

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

else

{

pid = fork();

if(pid > 0)

wait(0);

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

printf("\n Command %s not found\n",args[0]);

}

}while(1);

return 0;

}

**Slip 14**

1. **Write a C program to create an unnamed pipe. The child process will write following three messages to pipe and parent process display it.**

**Message1 = “Hello World”**

**Message2 = “Hello SPPU”**

**Message3 = “Linux is Funny”**

**Solution:**

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#define MSGSIZE 16

char \*msg1 = "Hello World";

char \*msg2 = "Hello SPPU";

char \*msg3 = "Linux is funny";

int main()

{

int fd[2],i;

char buff[MSGSIZE];

if(pipe(fd)<0)

exit(1);

write(fd[1],msg1,MSGSIZE);

write(fd[1],msg2,MSGSIZE);

write(fd[1],msg3,MSGSIZE);

for(i=0;i<3;i++)

{

read(fd[0],buff,MSGSIZE);

printf("\n%s",buff);

}

return 0;

}

1. **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 - search first occurrence of pattern in filename**

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

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

**solution:**

#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);

            }

        }

    }

}

/\*

Output:

 [root@localhost Desktop]# cc shellsearch.c

[root@localhost Desktop]# ./a.out

myshell$ search f aaa f1.txt

myshell$ search f aaa f1.txt

1: aaa

myshell$ search f 123 f1.txt

1: 123456

myshell$ search c a f1.txt

No of occurences of a= 1

myshell$ search c a f1.txt

No of occurences of a= 2

myshell$ search a aa f1.txt

1: aaa

2: aa

myshell$ search a 12 f1.txt

3: 123456

5: 1112

myshell$ pause

[root@localhost Desktop]#

\*/

**Slip 15**

**A)Write a C program to Identify the type (Directory, character device, Block device, Regular**

**file, FIFO or pipe, symbolic link or socket) of given file using stat() system call. \*/**

**Solution:**

#include<sys/stat.h>

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<sys/types.h>

#include<pwd.h>

#include<grp.h>

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

{

struct stat s;

char filetype;

memset(&s,0,sizeof(s));

if(argc<2)

{

printf("Insufficient arguments\n");

exit(1);

}

stat(argv[1],&s);

if((s.st\_mode & S\_IFMT)==S\_IFREG) filetype = 'R';

else if((s.st\_mode & S\_IFMT)==S\_IFSOCK) filetype = 'S';

else if((s.st\_mode & S\_IFMT)==S\_IFLNK) filetype = 'L';

else if((s.st\_mode & S\_IFMT)==S\_IFBLK) filetype = 'B';

else if((s.st\_mode & S\_IFMT)==S\_IFDIR) filetype = 'D';

else if((s.st\_mode & S\_IFMT)==S\_IFCHR) filetype = 'C';

else if((s.st\_mode & S\_IFMT)==S\_IFIFO) filetype = 'F';

printf("%s\t%c\n",argv[1],filetype);

return 0;

}

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!!!”.**

**Solution**

#include<stdio.h>

#include<signal.h>

#include<unistd.h>

#include<stdlib.h>

void sighup(int signo)

{

signal(SIGHUP,sighup);

printf("\nCHILD : I have received SIGHUP");

}

void sigint(int signo)

{

signal(SIGINT,sigint);

printf("\nCHILD : I have received SIGINT");

}

void sigquit(int signo)

{

// signal(SIGQUIT,sigquit);

printf("\nCHILD : My daddy has killed me");

exit(0);

}

int main()

{

int pid;

struct sigaction sigact;

sigact.sa\_flags=0;

sigemptyset(&sigact.sa\_mask);

sigact.sa\_handler = sighup;

if(sigaction(SIGHUP,&sigact,NULL)<0)

{

printf("\nsigaction error");

exit(1);

}

sigact.sa\_handler = sigint;

if(sigaction(SIGINT,&sigact,NULL)<0)

{

printf("\nsigaction error");

exit(1);

}

sigact.sa\_handler = sigquit;

if(sigaction(SIGQUIT,&sigact,NULL)<0)

{

printf("\nsigaction error");

exit(1);

}

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

{

printf("\nfork error");

exit(1);

}

if(pid == 0) //child

{

for(;;) ;

}

else //parent

{

sigact.sa\_handler = SIG\_DFL;

sigaction(SIGHUP,&sigact,NULL);

sigaction(SIGINT,&sigact,NULL);

sigaction(SIGQUIT,&sigact,NULL);

printf("\nparent sending SIGHUP");

kill(pid,SIGHUP);

sleep(3);

printf("\nparent sending SIGINT");

kill(pid,SIGINT);

sleep(3);

printf("\nparent sending SIGHUP");

kill(pid,SIGHUP);

sleep(3);

printf("\nparent sending SIGINT");

kill(pid,SIGINT);

sleep(3);

printf("\nparent sending SIGINT");

kill(pid,SIGINT);

sleep(3);

printf("\nparent sending SIGQUIT");

kill(pid,SIGQUIT);

sleep(3);

}

return 0;

}

**Slip 16**

1. **Write a C program that catches the ctrl-c (SIGINT) signal for the first time and display the appropriate message and exits on pressing ctrl-c again.**

**Solution:**

#include<stdio.h>

#include<signal.h>

void handle\_sigint(int sig)

{

printf("\ncaught signal %d\n",sig);

signal(SIGINT, SIG\_DFL);

}

int main()

{

signal(SIGINT,handle\_sigint);

while(1)

{

printf("hello world\n");

sleep(1);

}

return 0;

}

1. **Write a C program to implement the following unix/linux command on current directory ls –l > output.txt DO NOT simply exec ls -l > output.txt or system command from the program.**

**Solution**

#include<sys/stat.h>

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<sys/types.h>

#include<pwd.h>

#include<grp.h>

#include<dirent.h>

#include<unistd.h>

#include<fcntl.h>

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

{

DIR \*dirptr;

struct dirent \*entry;

char curDir[80];

getcwd(curDir,80);

printf("%s\n",curDir);

struct stat s;

struct tm \*timeinfo;

struct passwd \*pw;

struct group\*gr;

int fd;

char filetype,perm,\*date;

memset(&s,0,sizeof(s));

fd=open("a.txt",O\_CREAT | O\_WRONLY);

chmod("a.txt",0777);

close(1);

dup(fd);

dirptr = opendir(curDir);

while((entry = readdir(dirptr))!=NULL)

{

printf("\n");

stat(entry->d\_name,&s);

if((s.st\_mode & S\_IFMT)==S\_IFREG) filetype = '-';

else if((s.st\_mode & S\_IFMT)==S\_IFSOCK) filetype = 'S';

else if((s.st\_mode & S\_IFMT)==S\_IFLNK) filetype = 'L';

else if((s.st\_mode & S\_IFMT)==S\_IFBLK) filetype = 'B';

else if((s.st\_mode & S\_IFMT)==S\_IFDIR) filetype = 'D';

else if((s.st\_mode & S\_IFMT)==S\_IFCHR) filetype = 'C';

else if((s.st\_mode & S\_IFMT)==S\_IFIFO) filetype = 'F';

date = ctime(&s.st\_atime);

timeinfo = localtime(&s.st\_atime);

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

pw = getpwuid(s.st\_uid);

gr = getgrgid(s.st\_gid);

printf("%c",filetype);

printf((s.st\_mode & S\_IRUSR)?"r":"-");

printf((s.st\_mode & S\_IWUSR)?"w":"-");

printf((s.st\_mode & S\_IXUSR)?"x":"-");

printf((s.st\_mode & S\_IRGRP)?"r":"-");

printf((s.st\_mode & S\_IWGRP)?"w":"-");

printf((s.st\_mode & S\_IXGRP)?"x":"-");

printf((s.st\_mode & S\_IROTH)?"r":"-");

printf((s.st\_mode & S\_IWOTH)?"w":"-");

printf((s.st\_mode & S\_IXOTH)?"x":"-");

printf(" %ld %s %s %ld\t%s %s",s.st\_nlink,pw->pw\_name,pw-

>pw\_name,s.st\_size,date,entry->d\_name);

}

close(fd);

return 0;

}

**Slip 17**

1. **Write a C program to display the given message ‘n’ times. (make a use of setjmp and longjmp system call)**

**Solution**

#include<setjmp.h>

#include<stdio.h>

#include<stdlib.h>

static jmp\_buf jmpbuffer;

static void f1()

{

printf("\nInside f1(): before longjmp");

longjmp(jmpbuffer,1);

printf("\nInside f1(): after longjmp");

}

int main()

{

if(setjmp(jmpbuffer)!=0)

printf("\nInside main():after longjmp\n");

else

{

printf("\nInside main: calling f1()");

f1();

}

return 0;

}

**Slip 18**

1. **Write a C program to display the last access and modified time of a given file.**

**Solution:**

#include<sys/stat.h>

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<sys/types.h>

#include<pwd.h>

#include<grp.h>

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

{

struct stat s;

struct tm \*timeinfo;

memset(&s,0,sizeof(s));

char \*date;

if(argc<2)

{

printf("Insufficient arguments\n");

exit(1);

}

stat(argv[1],&s);

date = ctime(&s.st\_atime);

timeinfo = localtime(&s.st\_atime);

printf("\nFile modification time = %s",ctime(&s.st\_mtime));

printf("\nFile access time = %s",ctime(&s.st\_atime));

return 0;

}

1. **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**

**Solution**

#include<stdio.h>

#include<unistd.h>

#include<signal.h>

static void sig\_handler(int signo)

{

if(signo == SIGINT)

printf("\nCaught SIG\_INT");

if(signo == SIGQUIT)

printf("\nCaught SIG\_INT");

if(signal(SIGINT,SIG\_DFL)==SIG\_ERR)

printf("\ncan't reset SIGINT");

if(signal(SIGQUIT,SIG\_DFL)==SIG\_ERR)

printf("\ncan't reset SIGQUIT");

}

int main()

{

sigset\_t newmask, oldmask, pendmask;

if(signal(SIGINT,sig\_handler)==SIG\_ERR)

printf("\ncant catch sigint");

if(signal(SIGQUIT,sig\_handler)==SIG\_ERR)

printf("\ncant catch sigquit");

sigemptyset(&newmask);

sigaddset(&newmask, SIGINT);

sigaddset(&newmask, SIGQUIT);

int pipefd[2],retstatus,pid;

retstatus = pipe(pipefd);

if(retstatus==-1)

{

printf("\nfailed to create pipe\n");

return 1;

}

pid=fork();

if(pid<0)

{

printf("\nfailed to create child\n");

return 2;

}

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

{

close(pipefd[0]); //close read end of pipe

close(1); // close std.output

dup(pipefd[1]);

execlp("ls","ls","-l",(char \*)0);

}

else // parent process

{

close(pipefd[1]); // close write end of pipe

close(0); // close std. input

dup(pipefd[0]);

if(sigprocmask(SIG\_BLOCK, &newmask, &oldmask)<0)

printf("\nsigblock error");

sleep(5);

if(sigpending(&pendmask)<0)

printf("\nsig-pending error");

if(sigismember(&pendmask,SIGINT))

printf("\nSIGINT pending");

if(sigismember(&pendmask,SIGQUIT))

printf("\nSIGQUIT pending");

if(sigprocmask(SIG\_SETMASK, &oldmask, NULL)<0)

printf("\nsig\_setmask error");

printf("\nSIGINT unblocked");

printf("\nSIGQUT unblocked");

sleep(10);

execlp("wc","wc","-l",(char \*)0);

}

return 0;

}

**Slip 19**

1. **Write a C program to move the content of file1.txt to file2.txt and remove the file1.txt**

**from directory.**

**Solution:**

#include<stdio.h>

#include<fcntl.h>

#include<unistd.h>

#include<stdlib.h>

int main()

{

char ch;

int fd1 = open("dip.txt",O\_RDONLY);

int fd2 = creat("dip1.txt",O\_CREAT | O\_WRONLY);

while((read(fd1,&ch,1)!=0))

write(fd2,&ch,1);

close(fd1);

close(fd2);

unlink("dip.txt");

return 0;

}

1. **Write a C program which creates a child process to run linux/ unix command or any user defined program. The parent process set the signal handler for death of child signal and Alarm signal. If a child process does not complete its execution in 5 second then parent process kills child process.**

**Solution**

#include<stdio.h>

#include<unistd.h>

#include<sys/wait.h>

#include<signal.h>

pid\_t pid;

static void sig\_handler(int signo)

{

// if(signo==SIG\_ERR)

// printf("\n sig err");

if(signo == SIGCHLD)

printf("\nchild signal");

if(signo == SIGALRM)

{

printf("\n alarm signal");

kill(pid,SIGKILL);

}

}

int main()

{

signal(SIGCHLD,sig\_handler);

signal(SIGALRM,sig\_handler);

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

printf("\nfork error");

if(pid==0)

{

sleep(5);

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

}

alarm(5);

wait(NULL);

return 0;

}

**Slip 20**

1. **Write a C program that print the exit status of a terminated child process**

**solution**

#include<stdio.h>

#include<sys/wait.h>

#include<stdlib.h>

#include<sys/types.h>

#include<unistd.h>

void pr\_exit(int status)

{

if(WIFEXITED(status))

printf("\nnormal termination\nexit status = %d\n",WEXITSTATUS(status));

else if(WIFSIGNALED(status))

printf("\nabnormal termination\nsignal number =

%d%s\n",WTERMSIG(status),

#ifdef WCOREDUMP

WCOREDUMP(status)? "(Core file generated)":"");

#else

"");

#endif

else if(WIFSTOPPED(status))

printf("\nchild stopped \nsignal number = %d\n",WSTOPSIG(status));

}

int main()

{

pid\_t pid;

int status;

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

printf("fork error");

else if(pid==0) //child

exit(7);

if(wait(&status) != pid) //wait for child

printf("wait error");

pr\_exit(status); // & print its status

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

printf("fork error");

else if(pid==0) //child

abort(); // generates SIGABRT

if(wait(&status) != pid) //wait for child

printf("wait error");

pr\_exit(status);

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

printf("fork error");

else if(pid==0) //child

status/=0; // divide by 0 generates SIGFPE

if(wait(&status) != pid) //wait for child

printf("wait error");

pr\_exit(status);

return 0;

}