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

#include <stdio.h> #include <unistd.h>

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

{

if (access(argv[1],F\_OK)==0) printf("File %s exists", argv[1]);

else

printf("File %s doesn't exist.", argv[1]);

return 0;

}

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.

#include<stdio.h> #include<dirent.h> #include<string.h>

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

{

DIR \*d;

char \*position; struct dirent \*dir; int i=0;

if(argc!=2){

printf("Provide suffiecient args");

}

else {

d = opendir("."); if (d)

{

while ((dir = readdir(d)) != NULL)

{

position=strstr(dir->d\_name,argv[1]); i=position-dir->d\_name;

if(i==0)

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

}

closedir(d);

}

return(0);

}

}

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.

#include <sys/types.h> #include <sys/stat.h> #include <time.h> #include <stdio.h> #include <stdlib.h>

int main(int argc, char \*argv[]) { struct stat info;

if (argc != 2) {

printf("Enter filename\n");

}

if (stat(argv[1], &info) == -1) { printf("stat erro"); exit(EXIT\_FAILURE);

}

printf("I-node number: %ld\n", (long) info.st\_ino); printf("File size: %lld bytes\n",(long long) info.st\_size); printf("Last file access: %s", ctime(&info.st\_atime)); printf("Last file modification: %s", ctime(&info.st\_mtime));

printf("No of hard links: %d\n",info.st\_nlink); printf("File Permissions: \t");

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

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

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

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

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

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

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

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

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

putchar('\n');

}

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.

#include <unistd.h> #include <fcntl.h> #include <stdio.h> #include <sys/stat.h> #include <sys/types.h>

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

{

if(argc != 2)

return 1;

int file=0; if((file=open(argv[1],O\_RDONLY)) < -1)

return 1;

struct stat fileStat; if(fstat(file,&fileStat) < 0)

return 1;

printf("Information for %s\n",argv[1]); printf("- \n");

printf("File Size: \t\t%d bytes\n",fileStat.st\_size); printf("Number of Hard Links: \t%d\n",fileStat.st\_nlink); printf("File inode: \t\t%d\n",fileStat.st\_ino);

//printf("Last file access: %s", ctime(&fileStat.st\_atime));

//printf("Last file modification: %s", ctime(&fileStat.st\_mtime));

printf("File Permissions: \t");

printf( (S\_ISDIR(fileStat.st\_mode)) ? "d" : "-");

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

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

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

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

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

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

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

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

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

close(file); return 0;

}

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”

#include<stdio.h> #include<unistd.h>

int main() {

int pipefds[2]; int returnstatus;

char writemessages[3][20]={"Hello World", "Hello SPPU","Linux is Funny"}; char readmessage[20];

returnstatus = pipe(pipefds);

if (returnstatus == -1) {

printf("Unable to create pipe\n"); return 1;

}

int child = fork(); if(child==0){

printf("Child is Writing to pipe - Message 1 is %s\n", writemessages[0]); write(pipefds[1], writemessages[0], sizeof(writemessages[0])); printf("Child is Writing to pipe - Message 2 is %s\n", writemessages[1]); write(pipefds[1], writemessages[1], sizeof(writemessages[1])); printf("Child is Writing to pipe - Message 3 is %s\n", writemessages[2]); write(pipefds[1], writemessages[2], sizeof(writemessages[2]));

}

else

{

readmessage); readmessage);

readmessage);

}

read(pipefds[0], readmessage, sizeof(readmessage)); printf("Parent Process is Reading from pipe – Message 1 is %s\n",

read(pipefds[0], readmessage, sizeof(readmessage)); printf("Parent Process is Reading from pipe – Message 2 is %s\n",

read(pipefds[0], readmessage, sizeof(readmessage)); printf("Parent Process is Reading from pipe – Message 3 is %s\n",

}

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

#include <stdio.h> #include <stdlib.h> #include <string.h> #include <unistd.h> #include <sys/stat.h> #include <fcntl.h> #include <sys/types.h>

#include <sys/stat.h> #include <sys/io.h> #include <sys/mman.h>

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

{

unsigned char \*f, \*g; int size;

struct stat s;

const char \* file\_name = argv[1]; int fd = open(argv[1], O\_RDONLY);

int status = fstat(fd, &s); size = s.st\_size;

int i;

f = (char \*) mmap (0, size, PROT\_READ, MAP\_PRIVATE, fd, 0);

//g = (char \*) mmap (0, size, PROT\_READ, MAP\_PRIVATE, fd, 0);

for(i = 0; i < size; i++) { char c;

c = f[i]; putchar(c);

}

//ABOVE THIS WORKS

// int z = 0;

//while(f[z] != NULL) {

//z++;

// printf("%d", z);

// } int x;

int y = 0; close(fd);

FILE \*f1;

f1 = fopen(argv[2], "w+");

for(x = size - 1; x >= 0; x--)

{

char c;

c = f[x];

fputc(c, f1);

}

return 0;

}

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

#include <fcntl.h> #include<stdio.h> #include<stdlib.h>

char buf1[] = "welcome";

char buf2[] = "Computer science";

int main(void)

{

int fd;

if ((fd = creat("file\_with\_hole.txt",0777 )) < 0) printf("creat error");

if (write(fd, buf1, 7) != 7)

printf("buf1 write error"); lseek(fd,100,SEEK\_CUR);

if (write(fd, buf2, 16) != 16)

printf("buf2 write error");

exit(0);

}

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

#include <stdio.h> #include <sys/resource.h> #include <string.h> #include <errno.h> #include <unistd.h> #include <sys/types.h> #include <sys/stat.h> #include <fcntl.h>

int main() {

struct rlimit old\_lim, lim, new\_lim;

// Get old limits

if( getrlimit(RLIMIT\_NOFILE, &old\_lim) == 0) printf("Old limits -> soft limit= %ld \t"

" hard limit= %ld \n", old\_lim.rlim\_cur,

old\_lim.rlim\_max);

else

fprintf(stderr, "%s\n", strerror(errno));

// Set new value lim.rlim\_cur = 3;

lim.rlim\_max = 1024;

// Set limits if(setrlimit(RLIMIT\_NOFILE, &lim) == -1)

fprintf(stderr, "%s\n", strerror(errno));

// Get new limits

if( getrlimit(RLIMIT\_NOFILE, &new\_lim) == 0) printf("New limits -> soft limit= %ld "

"\t hard limit= %ld \n", new\_lim.rlim\_cur,

new\_lim.rlim\_max);

else

fprintf(stderr, "%s\n", strerror(errno));

return 0;

}

Output:

Old limits -> soft limit= 1048576 hard limit= 1048576 New limits -> soft limit= 3 hard limit= 1024

Write a C program to display as well as resets the environment variable such as path, home, root etc.

/\* C Program to Print Environment variables \*/ #include <stdio.h>

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

{

int i;

for (i = 0; envp[i] != NULL; i++)

{

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

}

/\* set environment variable \_EDC\_ANSI\_OPEN\_DEFAULT to "Y" \*/ setenv("\_EDC\_ANSI\_OPEN\_DEFAULT","Y",1);

/\* set x to the current value of the \_EDC\_ANSI\_OPEN\_DEFAULT\*/ x = getenv("\_EDC\_ANSI\_OPEN\_DEFAULT");

printf("program1 \_EDC\_ANSI\_OPEN\_DEFAULT = %s\n", (x != NULL) ? x : "undefined");

}

Write a C program that will only list all subdirectories in alphabetical order from current directory. #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);

}

}

Write a C program to display statistics related to memory allocation system. (Use mallinfo() system call).

#include <[malloc.h](https://linux.die.net/include/malloc.h)> #include "tlpi\_hdr.h"

static void display\_mallinfo(void)

{

struct mallinfo mi; mi = mallinfo();

printf("Total non-mmapped bytes (arena): %d\n", mi.arena); printf("# of free chunks (ordblks): %d\n", mi.ordblks); printf("# of free fastbin blocks (smblks): %d\n", mi.smblks); printf("# of mapped regions (hblks): %d\n", mi.hblks); printf("Bytes in mapped regions (hblkhd): %d\n", mi.hblkhd); printf("Max. total allocated space (usmblks): %d\n", mi.usmblks); printf("Free bytes held in fastbins (fsmblks): %d\n", mi.fsmblks); printf("Total allocated space (uordblks): %d\n", mi.uordblks); printf("Total free space (fordblks): %d\n", mi.fordblks); printf("Topmost releasable block (keepcost): %d\n", mi.keepcost);

}

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

{

#define MAX\_ALLOCS 2000000

char \*alloc[MAX\_ALLOCS];

int numBlocks, j, freeBegin, freeEnd, freeStep; size\_t blockSize;

if (argc < 3 || strcmp(argv[1], "--help") == 0)

usageErr("%s num-blocks block-size [free-step [start-free " "[end-free]]]\n", argv[0]);

numBlocks = atoi(argv[1]); blockSize = atoi(argv[2]);

freeStep = (argc > 3) ? atoi(argv[3]) : 1; freeBegin = (argc > 4) ? atoi(argv[4]) : 0; freeEnd = (argc > 5) ? atoi(argv[5]) : numBlocks;

printf("============== Before allocating blocks ==============\n"); display\_mallinfo();

for (j = 0; j < numBlocks; j++) { if (numBlocks >= MAX\_ALLOCS)

fatal("Too many allocations");

alloc[j] = malloc(blockSize); if (alloc[j] == NULL)

errExit("malloc");

}

printf("\n============== After allocating blocks ==============\n"); display\_mallinfo();

for (j = freeBegin; j < freeEnd; j += freeStep) free(alloc[j]);

printf("\n============== After freeing blocks ==============\n"); display\_mallinfo();

exit(EXIT\_SUCCESS);

}

Write a C program to create variable length arrays using alloca() system call.

// C program for variable length members in structures in

// GCC before C99 #include <stdio.h> #include <stdlib.h> #include <string.h>

// Structure of type student struct student {

int stud\_id; int name\_len;

int struct\_size; char stud\_name[0];

// variable length array must be

// last.

};

// Memory allocation and initialisation of structure struct student\* createStudent(struct student\* s, int id,

char a[])

{

s = alloca(sizeof(\*s) + sizeof(char) \* strlen(a));

s->stud\_id = id;

s->name\_len = strlen(a); strcpy(s->stud\_name, a);

s->struct\_size

= (sizeof(\*s)

+ sizeof(char) \* strlen(s->stud\_name));

return s;

}

// Print student details

void printStudent(struct student\* s)

{

printf("Student\_id : %d\n" "Stud\_Name : %s\n" "Name\_Length: %d\n"

"Allocated\_Struct\_size: %d\n\n",

s->stud\_id, s->stud\_name, s->name\_len, s->struct\_size);

// Value of Allocated\_Struct\_size here is in bytes.

}

// Driver Code int main()

{

struct student \*s1, \*s2;

s1 = createStudent(s1, 523, "Sanjayulsha");

s2 = createStudent(s2, 535, "Cherry");

printStudent(s1); printStudent(s2);

// size in bytes

printf("Size of Struct student: %lu\n", sizeof(struct student));

// size in bytes

printf("Size of Struct pointer: %lu", sizeof(s1));

return 0;

}

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)

#include <fcntl.h> #include <unistd.h> #include <stdio.h> #include<signal.h> #include<sys/types.h> #include<sys/wait.h> #include <stdlib.h>

void Dingdong()

{

printf("Ding!"); exit(1);

}

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

{

if(argc!=3)

{

printf("How much seconds you want to sleep the child process\n");

}

int PauseSecond=(argv[1]);

{

if(fork()==0)

{

printf("waiting for alarm to go off\n"); printf("%d second pause",PauseSecond); sleep(PauseSecond); kill(getpid(),SIGALRM);

}

else {

printf("Alarm application starting\n", getpid());

signal(SIGALRM,Dingdong); printf("done");

}

}

}

Write a C program that redirects standard output to a file output.txt. (use of dup and open system call).

#include <stdio.h> #include <stdlib.h> #include <unistd.h> #include <fcntl.h>

int main(void) {

int number1, number2, sum;

int input\_fds = open("./input.txt", O\_RDONLY);

if(dup2(input\_fds, STDIN\_FILENO) < 0) {

[printf](https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("Unable to duplicate file descriptor."); [exit](https://www.opengroup.org/onlinepubs/009695399/functions/exit.html)(EXIT\_FAILURE);

}

[scanf](https://www.opengroup.org/onlinepubs/009695399/functions/scanf.html)("%d %d",

&number1,

&number2);

sum = number1 + number2;

[printf](https://www.opengroup.org/onlinepubs/009695399/functions/printf.html)("%d + %d = %d\n", number1, number2, sum);

return EXIT\_SUCCESS;

}

Write a C program to create an unnamed pipe. Write following three messages to pipe and display it.

Message1 = “Hello World” Message2 = “Hello SPPU” Message3 = “Linux is Funny” .

#include<stdio.h> #include<unistd.h>

int main() {

int pipefds[2]; int returnstatus;

char writemessages[3][20]={"Hello World", "Hello SPPU","Linux is Funny"}; char readmessage[20];

returnstatus = pipe(pipefds);

if (returnstatus == -1) {

printf("Unable to create pipe\n"); return 1;

}

int child = fork(); if(child==0){

printf("Child is Writing to pipe - Message 1 is %s\n", writemessages[0]); write(pipefds[1], writemessages[0], sizeof(writemessages[0])); printf("Child is Writing to pipe - Message 2 is %s\n", writemessages[1]);

}

else

{

readmessage); readmessage);

readmessage);

}

write(pipefds[1], writemessages[1], sizeof(writemessages[1])); printf("Child is Writing to pipe - Message 3 is %s\n", writemessages[2]); write(pipefds[1], writemessages[2], sizeof(writemessages[2]));

read(pipefds[0], readmessage, sizeof(readmessage)); printf("Parent Process is Reading from pipe – Message 1 is %s\n",

read(pipefds[0], readmessage, sizeof(readmessage)); printf("Parent Process is Reading from pipe – Message 2 is %s\n",

read(pipefds[0], readmessage, sizeof(readmessage)); printf("Parent Process is Reading from pipe – Message 3 is %s\n",

}

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.

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.

#include <stdio.h> #include <unistd.h> #include <stdlib.h> #include <signal.h>

void sigfun(int sig)

{

printf("You have presses Ctrl-C , please press again to exit"); (void) signal(SIGINT, SIG\_DFL);

}

int main()

{

(void) signal(SIGINT, sigfun);

");

while(1) { printf("Hello World!

sleep(1);

}

return(0);

}

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

#include <stdio.h> #include <setjmp.h> jmp\_buf buf; main() {

int x = 1,n;

setjmp(buf); //set the jump position using buf printf("Hello"); // Prints a msg

x++;

printf(“Enter the number”); scanf(“%d”, &n);

if (x <= n)

longjmp(buf, 1); // Jump to the point located by setjmp

}

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

#include <time.h> #include <sys/types.h> #include <sys/stat.h> #include <stdio.h>

int main()

{

char filename[] = "c:\\test.txt"; char timeStr[ 100 ] = "";

struct stat buf; time\_t ltime; char datebuf [9];

char timebuf [9];

if (!stat(filename, &buf))

{

strftime(timeStr, 100, "%d-%m-%Y %H:%M:%S", localtime( &buf.st\_mtime)); printf("\nLast modified date and time = %s\n", timeStr);

}

else

{

printf("error getting atime\n");

}

\_strtime(timebuf);

\_strdate(datebuf);

printf("\nThe Current time is %s\n",timebuf); printf("\nThe Current Date is %s\n",datebuf); time( &ltime );

printf("\nThe Current time is %s\n",ctime( &ltime ));

printf("\Diff between current and last modified time ( in seconds ) %f\n", difftime(ltime ,buf.st\_mtime )

);

return 0;

}

Write a C program to move the content of file1.txt to file2.txt and remove the file1.txt from directory. #include<stdio.h>

#include<fcntl.h> #include<unistd.h> #define buffersize 10000 int main()

{

char source[25],destination[25]; //Source and destination filename

char buffer[buffersize]; //Character buffer

ssize\_t read\_in,write\_out; //Number of bytes returned by single read and write operation printf(“Enter source file name”);

scanf(“%s”,&source); printf(“%s”,source);

int sourcefiledesc = open(source,O\_RDONLY); //Source file open in read only mode if(sourcefiledesc < 0 )

{

printf(“Source file not Exist”); //Source file not found

}

else

{

printf(“Enter destination file name”); scanf(“%s”,&destination);

/\* Destination file open in write mode and if not found then create\*/ int destfiledesc = open(destination,O\_WRONLY | O\_CREAT); while((read\_in = read(sourcefiledesc,&buffer,buffersize))>0)

{

write\_out = write(destfiledesc,&buffer,read\_in);

}

if (remove(sourcefiledesc) == 0) printf("File Deleted successfully");

else

printf("Unable to delete the file"); close(sourcefiledesc); close(destfiledesc);

}

return 0;

}

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

// C code to find the exit status of child process #include <stdio.h>

#include <stdlib.h> #include <unistd.h> #include <sys/types.h> #include <sys/wait.h>

// Driver code int main(void)

{

pid\_t pid = fork();

if ( pid == 0 )

{

/\* The pathname of the file passed to execl() is not defined \*/

execl("/bin/sh", "bin/sh", "-c", "./nopath", "NULL");

}

int status;

waitpid(pid, &status, 0); if ( WIFEXITED(status) )

{

int exit\_status = WEXITSTATUS(status); printf("Exit status of the child was %d\n",

exit\_status);

}

return 0;

}

**Second Question(20)**

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() )

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).

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.

#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);

}

}

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

Write a C program to read all txt files (that is files that ends with .txt) in the current directory and merge them all to one txt file and returns a file descriptor for the new file

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

#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)

{

}

else

{

}

}

}

}

execlp("/bin",NULL); if(strcmp(t1,"exit")==0)

exit(0); system(buff);

wait(NULL); exit(0);

/\*

[root@localhost ass1]# ./a.out

myshell$list f ass2 rr.C

NPSJP.C PSJF.C PRIORITY.BAK

..

. PP.BAK

PRIORITY.C PP.C

myshell$list n ass2 Total No of Entries: 9

myshell$list i ass2

rr.C 1452033

NPSJP.C 1452013

PSJF.C 1452032

PRIORITY.BAK 1452016

.. 1451875

. 1452001

PP.BAK 1452014

PRIORITY.C 1452017

PP.C 1452015

myshell$ls

#a.c# a.out ass2 a.txt b.c count.c list.c search.c s.txt typeline.c myshell$pause

\*/

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 <filename>

DO NOT simply exec ls -l <filename> or system command from the program

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, …)

#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;

}

/\*[root@localhostsppuslipquestions]# cc slip20.c [root@localhostsppuslipquestions]# ./a.out slip12.c a1.c q12.c a1.c 465

q12.c 596

slip12.c 803

\*/

Write a C program which creates two files. The first file should have read and write permission to owner, group of owner and other users whereas second file has read and write permission to owner(use umask() function). Now turn on group-id and turn off group execute permission of first file. Set the read permission to all user for second file (use chmod() function).

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

#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);

}

else

{

}

}

}

}

/\*

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

system(buff);

wait(NULL); exit(0);

[root@localhost ass1]# ./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

\*/

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.

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.

1. typeline +10 <filename> - print first 10 lines of file
2. typeline -20 <filename> - print last 20 lines of file
3. typeline a <filename> - print all lines of file #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);

}

}

else

{

printf("\n");

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)

{

}

else

{

}

}

}

}

/\*

execlp("/bin",NULL); if(strcmp(t1,"exit")==0)

exit(0); system(buff);

wait(NULL); exit(0);

[root@localhost ass1]# cc typeline.c [root@localhost ass1]# ./a.out

myshell$typeline a s.txt hello aa welcome

dyp bb tybcs aa dyp

good morning dyp gm

myshell$typeline -2 s.txt good morning

dyp gm

myshell$typeline +3 s.txt hello aa welcome

dyp bb tybcs aa dyp

myshell$ls

#a.c# a.out ass2 a.txt b.c count.c list.c search.c s.txt typeline.c

myshell$pause

\*/

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.

1. search f <pattern><filename> - search first occurrence of pattern in filename
2. search c <pattern><filename> - count no. of occurrences of pattern in filename iii) search a <pattern><filename> - search all occurrences of pattern in filename

#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)

{

}

} i++;

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

}

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)

{

}

else

{

}

}

execlp("/bin",NULL); if(strcmp(t1,"exit")==0)

exit(0); system(buff);

wait(NULL); exit(0);

}

}

/\*

[root@localhost ass1]# cc search.c [root@localhost ass1]# ./a.out myshell$search f aa s.txt

1: hello aa welcome

myshell$search c dyp s.txt No of occurences of dyp= 3

myshell$search a dyp s.txt 2: dyp bb

3: tybcs aa dyp

5: dyp gm myshell$ls

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

myshell$pause

\*/

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!!!”.

#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);

}

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.

#include<stdlib.h> #include<stdio.h> #include<string.h> main(int argc, char \*argv[])

{

char d[50]; if(argc==2)

{

bzero(d,sizeof(d));

strcat(d,"ls ");

strcat(d,"> ");

strcat(d,argv[1]); system(d);

}

else

printf("\nInvalid No. of inputs");

}

/\*output:-

[root@localhost unix]# cc slip23.c [root@localhost unix]# ls

exitdemo.c hello orphan.c signaldemo.c~ x.out exitprg.c hello1 orphan.c~ slip10.c zombie.c exitprg.c~ hello1.c p.c slip10.c~ zombie.c~ f1.txt hello.c p.c~ slip16.c

f1.txt~ hello.txt pipedemo.c slip16.c~ f3 hello.txt~ pipedemo.c~ slip19.c [root@localhost unix]# cat >file3

[6]+ Stopped cat > file3 [root@localhost unix]# ./a.out file3 [root@localhost unix]# cat file3

|  |  |  |  |
| --- | --- | --- | --- |
| exitdemo.c | hello | orphan.c | signaldemo.c~ x.out |
| exitprg.c | hello1 | orphan.c~ | slip10.c zombie.c |
| exitprg.c~ | hello1.c | p.c | slip10.c~ zombie.c~ |

f1.txt hello.c p.c~ slip16.c f1.txt~ hello.txt pipedemo.c slip16.c~ f3 hello.txt~ pipedemo.c~ slip19.c

\*/

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

#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;

}

/\*

[root@localhostUnix]# cc month.c [root@localhostUnix]# ./a.out Mar a.out Fri Mar 20 22:15:23 2020

. Fri Mar 20 22:15:23 2020

.. Fri Mar 20 22:14:29 2020

\*/

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

// C code to implement ls | wc command #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);

}

}

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.

Write a C program that will read a directory containing a set of files, create a new sub directory called “backup” in same directory and copy all the files from directory to the new sub directory “backup”. You are supposed to use exec() with “cp” command. ( Create child processes that can execute each of the cp commands to copy the file from directory to subdirectory “backup”)