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

**SUBJECT: Advanced Operating System Practical Slips**

**10 Marks Program**

**Slip No:1**

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

**Soln:**

#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;

}

**Slip No: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.**

**Soln:**

#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);

}

}

**Slip No:3**

**A)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.**

**Soln:**

#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');

}

**Slip No:4**

**A)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.**

**Soln:**

#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;

}

**Slip No:5**

**A) 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”**

**Soln:**

#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

{

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Parent Process is Reading from pipe – Message 1 is %s\n", readmessage);

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Parent Process is Reading from pipe – Message 2 is %s\n", readmessage);

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Parent Process is Reading from pipe – Message 3 is %s\n", readmessage);

}

}

**Slip No:6**

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

**Soln:**

#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;

}

**Slip No:7**

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

**Soln:**

#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);

}

**Slip No:8**

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

**Soln:**

#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

**Slip No:9**

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

**Soln:**

#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");

}

**Slip No:10**

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

**Soln:**

#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);

}

**Slip No:11**

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

**Soln:**

#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;

}

**Slip No:12**

**A)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)**

**Soln:**

#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");

}

}

}

**Slip No:13**

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

**Soln:**

#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;

}

**Slip No:14**

**A) 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”**

**Soln:**

#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

{

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Parent Process is Reading from pipe – Message 1 is %s\n", readmessage);

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Parent Process is Reading from pipe – Message 2 is %s\n", readmessage);

read(pipefds[0], readmessage, sizeof(readmessage));

printf("Parent Process is Reading from pipe – Message 3 is %s\n", readmessage);

}

}

**Slip No:15**

1. **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.**

**Soln:**

**Slip No: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.**

**Soln:**

#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);

}

**Slip No:17**

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

**Soln:**

#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

}

**Slip No:18**

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

**Soln:**

#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;

}

**Slip No:19**

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

**Soln:**

#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;

}

**Slip No:20**

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

**Soln:**

#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;

}