**System Programming**

Submitted by: Shabbir Ahmed Rafiq

Erp: 09422

QUESTION 1

How files permission could be set through shell commands and programmatically in C.

Provide working shell commands and c source code.

ANSWER

Shell commands:

File permission can be set through the ‘chmod’ shell command.

Chmod command takes the form:

chmod *options permissions filename*

*permissions* defines the permissions for the owner of the file (the "user"), members of the group who owns the file (the "group"), and anyone else ("others"). There are two ways to represent these permissions: with symbols (alphanumeric characters), or with octal numbers (the digits **0** through **7**).

Let's say you are the owner of a file named **myfile**, and you want to set its permissions so that:

1. the **u**ser can **r**ead, **w**rite, and e**x**ecute it;
2. members of your **g**roup can **r**ead and e**x**ecute it; and
3. **o**thers may only **r**ead it.

This command will do the trick:

chmod u=rwx,g=rx,o=r myfile

This example uses symbolic permissions notation. The letters **u**, **g**, and **o** stand for "**user**", "**group**", and "**other**". The equals sign ("**=**") means "set the permissions exactly like this," and the letters "**r**", "**w**", and "**x**" stand for "read", "write", and "execute", respectively. The commas separate the different classes of permissions, and there are no spaces in between them.

In octal notation:

chmod 754 myfile

Each digit in 754 represents the permission for the user, group and others, respectively. Each digit is a combination of the numbers 4, 2, 1 and 0.

4 → read  
2 → write  
1 → execute  
0 → no permission

C Programming Language:

We can use the following function in C from man 3p chmod:

#include <sys/stat.h>  
int chmod (const char \*filename, mode\_t mode)

To read the permissions:

int stat(const char \*restrict path, struct stat \*restrict buf);

##### Setting Read Permissions for User, Group, and Others

The following example sets read permissions for the owner, group, and others.

#include <sys/stat.h>

const char \*path;

...

chmod(path, S\_IRUSR|S\_IRGRP|S\_IROTH);

##### Setting Read, Write, and Execute Permissions for the Owner Only

The following example sets read, write, and execute permissions for the owner, and no permissions for group and others.

#include <sys/stat.h>

const char \*path;

...

chmod(path, S\_IRWXU);

##### Setting Different Permissions for Owner, Group, and Other

The following example sets owner permissions for CHANGEFILE to read, write, and execute, group permissions to read and execute, and other permissions to read.

#include <sys/stat.h>

#define CHANGEFILE "/etc/myfile"

...

chmod(CHANGEFILE, S\_IRWXU|S\_IRGRP|S\_IXGRP|S\_IROTH);

##### Setting and Checking File Permissions

The following example sets the file permission bits for a file named **/home/cnd/mod1**, then calls the [*stat*()](http://pubs.opengroup.org/onlinepubs/000095399/functions/stat.html) function to verify the permissions.

#include <sys/types.h>

#include <sys/stat.h>

int status;

struct stat buffer

...

chmod("home/cnd/mod1", S\_IRWXU|S\_IRWXG|S\_IROTH|S\_IWOTH);

status = stat("home/cnd/mod1", &buffer;);

Example Source code:

#include <sys/types.h>

#include <sys/stat.h>

int main() {

char\* path;

path = "/home/file.txt";

chmod(path, S\_IRWXU|S\_IRWXG|S\_IROTH|S\_IWOTH);

return 1;

}

QUESTION 2

How in File I/O Scatter‐Gather (mode/type) is performed. Write code to read and write to a file using this method.

ANSWER

Scatter/gather I/O, is a method of input and output by which a single procedure call sequentially reads data from multiple buffers and writes it to a single data stream, or reads data from a data stream and writes it to multiple buffers. The buffers are given in a vector of buffers. Scatter/gather refers to the process of gathering data from, or scattering data into, the given set of buffers.

CODE:

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <unistd.h>

#include <sys/uio.h>

#define NUMBUFS 3

int

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

{

const char \*buf1 = "Hello ";

const char \*buf2 = "SP ";

const char \*buf3 = "Class!\n";

struct iovec bufs[NUMBUFS];

bufs[0].iov\_base = (void\*) buf1;

bufs[0].iov\_len = strlen(buf1);

bufs[1].iov\_base = (void\*) buf2;

bufs[1].iov\_len = strlen(buf2);

bufs[2].iov\_base = (void\*) buf3;

bufs[2].iov\_len = strlen(buf3);

if (-1 == writev(STDOUT\_FILENO, bufs, NUMBUFS))

{

perror("writev()");

exit(EXIT\_FAILURE);

}

return 0;

}

QUESTION 3

List all signals you can find online and their relevant attributes in a table  
  
  
ANSWER

Linux Signals are:

Signal Name Number Description

SIGHUP 1 Hangup (POSIX)

SIGINT 2 Terminal interrupt (ANSI)

SIGQUIT 3 Terminal quit (POSIX)

SIGILL 4 Illegal instruction (ANSI)

SIGTRAP 5 Trace trap (POSIX)

SIGIOT 6 IOT Trap (4.2 BSD)

SIGBUS 7 BUS error (4.2 BSD)

SIGFPE 8 Floating point exception (ANSI)

SIGKILL 9 Kill (can't be caught or ignored) (POSIX)

SIGUSR1 10 User defined signal 1 (POSIX)

SIGSEGV 11 Invalid memory segment access (ANSI)

SIGUSR2 12 User defined signal 2 (POSIX)

SIGPIPE 13 Write on a pipe with no reader, Broken pipe (POSIX)

SIGALRM 14 Alarm clock (POSIX)

SIGTERM 15 Termination (ANSI)

SIGSTKFLT 16 Stack fault

SIGCHLD 17 Child process has stopped or exited, changed (POSIX)

SIGCONT 18 Continue executing, if stopped (POSIX)

SIGSTOP 19 Stop executing (can't be caught or ignored) (POSIX)

SIGTSTP 20 Terminal stop signal (POSIX)

SIGTTIN 21 Background process trying to read, from TTY (POSIX)

SIGTTOU 22 Background process trying to write, to TTY (POSIX)

SIGURG 23 Urgent condition on socket (4.2 BSD)

SIGXCPU 24 CPU limit exceeded (4.2 BSD)

SIGXFSZ 25 File size limit exceeded (4.2 BSD)

SIGVTALRM 26 Virtual alarm clock (4.2 BSD)

SIGPROF 27 Profiling alarm clock (4.2 BSD)

SIGWINCH 28 Window size change (4.3 BSD, Sun)

SIGIO 29 I/O now possible (4.2 BSD)

SIGPWR 30 Power failure restart (System V)

QUESTION 4

How sigevent structure is used to deliver signals to a process. Give working code

ANSWER

The sigevent structure is used to describe the way a process is to be notified about an event (e.g., completion of an asynchronous request, expiration of a timer, or the arrival of a message).

CODE

union sigval { /\* Data passed with notification \*/

int sival\_int; /\* Integer value \*/

void \*sival\_ptr; /\* Pointer value \*/

};

struct sigevent {

int sigev\_notify; /\* Notification method \*/

int sigev\_signo; /\* Notification signal \*/

union sigval sigev\_value; /\* Data passed with

notification \*/

void (\*sigev\_notify\_function) (union sigval);

/\* Function used for thread

notification (SIGEV\_THREAD) \*/

void \*sigev\_notify\_attributes;

/\* Attributes for notification thread

(SIGEV\_THREAD) \*/

pid\_t sigev\_notify\_thread\_id;

/\* ID of thread to signal (SIGEV\_THREAD\_ID) \*/

};

QUESTION 5

How multiple related process (parent/child) uses a pipe. Provide code?

ANSWER:

A pipe is a communication device that permits unidirectional communication. Typically, a pipe is used to communicate between two threads in a single process or between parent and child processes. Pipes are serial devices; the data is always read from the pipe in the same order it was written.

The child is created by a fork () call that is executed by the parent. The pipe is inherited by the child and may be passed on to the grand‐children by the child process or other children by the parent.

Code:

#include <stdlib.h>

#include <stdio.h>

void runpipe();

int

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

{

int pid, status;

int fd[2];

pipe(fd);

switch (pid = fork()) {

case 0: /\* child \*/

runpipe(fd);

exit(0);

default: /\* parent \*/

while ((pid = wait(&status)) != -1)

fprintf(stderr, "process %d exits with %d\n", pid, WEXITSTATUS(status));

break;

case -1:

perror("fork");

exit(1);

}

exit(0);

}

char \*cmd1[] = { "/bin/ls", "-al", "/", 0 };

char \*cmd2[] = { "/usr/bin/tr", "a-z", "A-Z", 0 };

void

runpipe(int pfd[])

{

int pid;

switch (pid = fork()) {

case 0: /\* child \*/

dup2(pfd[0], 0);

close(pfd[1]); /\* the child does not need this end of the pipe \*/

execvp(cmd2[0], cmd2);

perror(cmd2[0]);

default: /\* parent \*/

dup2(pfd[1], 1);

close(pfd[0]); /\* the parent does not need this end of the pipe \*/

execvp(cmd1[0], cmd1);

perror(cmd1[0]);

case -1:

perror("fork");

exit(1);

}

}

QUESTION 6:

How popen () and pclose () is used for managing pipes operation?

ANSWER

Since a common operation is to create a pipe to another process, to either read its output or send it input, the standard I/O library has historically provided the popen and pclose functions. These two functions handle all the dirty work that we've been doing ourselves: creating a pipe, forking a child, closing the unused ends of the pipe, executing a shell to run the command, and waiting for the command to terminate.

QUESTION 7

How message notification for queue is handled by a thread function. Give working code

for reading messages from queue?

ANSWER

#include <pthread.h>

#include <mqueue.h>

#include <fcntl.h> /\* For definition of O\_NONBLOCK \*/

#include "tlpi\_hdr.h"

/\* This program does not handle the case where a message already exists on

the queue by the time the first attempt is made to register for message

notification. In that case, the program would never receive a notification.

\*/

static void notifySetup(mqd\_t \*mqdp);

static void /\* Thread notification function \*/

threadFunc(union sigval sv)

{

ssize\_t numRead;

mqd\_t \*mqdp;

void \*buffer;

struct mq\_attr attr;

mqdp = sv.sival\_ptr;

/\* Determine mq\_msgsize for message queue, and allocate an input buffer

of that size \*/

if (mq\_getattr(\*mqdp, &attr) == -1)

errExit("mq\_getattr");

buffer = malloc(attr.mq\_msgsize);

if (buffer == NULL)

errExit("malloc");

+ /\* Reregister for message notification \*/

+

notifySetup(mqdp);

while ((numRead = mq\_receive(\*mqdp, buffer, attr.mq\_msgsize, NULL)) >= 0)

printf("Read %ld bytes\n", (long) numRead);

if (errno != EAGAIN) /\* Unexpected error \*/

errExit("mq\_receive");

free(buffer);

}

static void

notifySetup(mqd\_t \*mqdp)

{

struct sigevent sev;

sev.sigev\_notify = SIGEV\_THREAD; /\* Notify via thread \*/

sev.sigev\_notify\_function = threadFunc;

sev.sigev\_notify\_attributes = NULL;

/\* Could be pointer to pthread\_attr\_t structure \*/

sev.sigev\_value.sival\_ptr = mqdp; /\* Argument to threadFunc() \*/

if (mq\_notify(\*mqdp, &sev) == -1)

errExit("mq\_notify");

}

int

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

{

mqd\_t mqd;

if (argc != 2 || strcmp(argv[1], "--help") == 0)

usageErr("%s mq-name\n", argv[0]);

mqd = mq\_open(argv[1], O\_RDONLY | O\_NONBLOCK);

if (mqd == (mqd\_t) -1)

errExit("mq\_open");

notifySetup(&mqd);

pause(); /\* Wait for notifications via thread function \*/

}