

S228/417P

DUBLIN INSTITUTE OF TECHNOLOGY
KEVIN STREET, DUBLIN 8.

BSc Computer Science

Year 4

SUMMER EXAMINATIONS 2005

SYSTEMS PROGRAMMING

Mr. C. O'Leary
Dr. B. O'Shea
Mr. D. Yeates

May 2005, 2:30 – 5:30

Answer **FOUR** questions

All questions carry equal marks

1. (a) Consider the following code, stored in file `q1a.c`:

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

int main() {
    int filedes;
    if((filedes = open("file", O_CREAT, 0777)) == -1) {
        printf ("Couldn't open %s\n", "file");
        exit(1);
    }
    exit(0);
}
```

Now consider the following commands and their output:

```
[examiner@host q1]$ cc -o q1a q1a.c
[examiner@host q1]$ ls -l
-rwxr-xr-x  1 examiner  grp          11832 Jan  1 10:00 q1a
-rw-----  1 examiner  grp           198 Jan  1 10:00 q1a.c
[examiner@host q1]$ q1a
[examiner@host q1]$ ls -l
-rwxr-xr-x  1 examiner  grp           0 Jan  1 10:00 file
-rwxr-xr-x  1 examiner  grp          11832 Jan  1 10:00 q1a
-rw-----  1 examiner  grp           198 Jan  1 10:00 q1a.c
[examiner@host q1]$
```

Observe the permissions of the newly created file. Explain why the permissions that are granted are not those included in the program code. Demonstrate how this problem may be fixed.

(7 marks)

- (b) Explain clearly what is meant by the *effective user ID* of a file and demonstrate, using an example, a situation where the effective user ID of a file would be different to the *real user ID* for the same file.

What effect does a call to `chown(...)` have on the effective user ID for a file? Explain why this effect is required.

(8 marks)

Continued on the next page

- (c) Consider the following code:

```

#include <stdlib.h>
#include <sys/stat.h>
#include <dirent.h>

int main(int argc, char **argv) {
    int ftw2(const char *path, int (*func)());
    int list(const char *name, const struct stat *sptr, int type);
    ftw2(argv[1], list);
}

int ftw2(const char *path, int (*func)()) {
    struct stat statbuf;
    stat(path, &statbuf);
    // call the function whose name is passed as an argument
    (*func)(path, &statbuf, 1);
}

// type = 0 for directory, type = 1 for file
int list(const char *name, const struct stat *status, int type) {
    if(type == 1)
        printf("%-40s\t0%-3o\n", name, status->st_mode & 0777);
    else
        printf("%-40s*\t0%-3o\n", name, status->st_mode & 0777);
}

```

The above code contains a function named `ftw2(...)`, which is an attempt at re-implementing the `ftw(...)` function available in the C library `ftw.h`.

- i. Describe the behaviour of the above program.
- ii. Rewrite the `ftw2(...)` function so that it has the same behaviour as the `ftw(...)` function.

(10 marks)

- 2 (a) Consider the following code:

```

#include <stdio.h>
int main() {
    char buffer[10];
    FILE* file = fopen("f", "r");
    read(3, buffer, 10);
    buffer[10] = '\0';
    printf("%s\n", buffer);
    fgets(buffer, 10, file);
    printf("%s\n", buffer);
    fclose(file);
}

```

If you assume the existence of a file named `f`, of size 100 bytes, describe the output you would expect from the above program. What is unsafe about this code?

(6 marks)

- (b) Describe each of the following process attributes, and for each, explain what effect a call to `fork()` or `exec()` has on the attribute, and also how the attributes can be changed from within the program code:
- i. User ID
 - ii. Process ID
 - iii. Process Group ID
 - iv. Session Group ID
 - v. Environment Variables
 - vi. Current Working Directory
 - vii. Current Root Directory
 - viii. Priority
 - ix. File Descriptors

(9 marks)

- (c) Consider the following two programs:

q2c_A.c	q2c_B.c
<pre>#include <sys/wait.h> #include <unistd.h> #include <stdlib.h> int main() { int i=0; if(fork()==0) { sleep(10); exit(0); } while(wait(NULL)==0){ printf("%d\n",i++); sleep(1); } }</pre>	<pre>#include <sys/wait.h> #include <unistd.h> #include <stdlib.h> int main() { pid_t pid; int i=0; if((pid = fork())==0) { sleep(10); exit(0); } while(waitpid(pid,NULL,WNOHANG)==0){ printf("%d\n", i++); sleep(1); } }</pre>

- i. What output would you expect from each of the two programs above? Explain clearly.
- ii. Show how to modify `q2c_A.c` program so that when the child exits, the parent prints out its process ID and its exit status.
- iii. Extend `q2c_A.c` so that two identical children are forked. When one child exits, the parent prints out its process ID and then exits. Explain what happens to the second child process.
- iv. Show how the `atexit(...)` routine can be used to perform clean-up before a process exits.

(10 marks)

Continued on the next page

3. (a) Consider the following session:

```

[examiner@host q3]$ ls -l
total 12
-rwx----- 1 examiner grp          80 Jan  1 10:00 q3a_1
-rw----- 1 examiner grp        234 Jan  1 10:00 q3a_2.c
-rwx----- 1 examiner grp          27 Jan  1 10:00 store
[examiner@host q3]$ cat q3a_1
#!/bin/sh

trap 'store' 16

while [ 1 ]
do
    sleep 1
done
[examiner@host q3]$ cat store
#!/bin/bash

date >> log

[examiner@host q3]$ cat q3a_2.c
#include <stdio.h>
#include <signal.h>
int main(int argc, char** argv) {
    char input[10];
    int pid = atoi(argv[1]);
    do {
        printf("Enter info > ");
        fgets(input, 10, stdin);
        kill(pid, 16);
    } while (input[0] != 'q');
}
[examiner@host q3]$ cc -o q3a_2 q3a_2.c
[examiner@host q3]$ q3a_1 &
[1] 12624
[examiner@host q3]$ q3a_2 12624
Enter info > line1
Enter info > line2
Enter info > line3
Enter info > quit
[examiner@host q3]$

```

Describe in detail what happens during the session shown above.

(7 marks)

(b) In the context of signals and signal processing explain what is meant by the following terms. Use sample code where appropriate to help illustrate your answer:

- i. Signal handling
- ii. Signal blocking
- iii. Normal termination
- iv. Abnormal termination
- v. Core dump

(8 marks)

- (c) Consider the following system:
- i. A server process monitors the status of a file by checking every second whether its size has changed.
 - ii. Client processes (NOT children of the server) are interested in finding out when the size of the file has changed, but don't care what the actual size of the file is.
 - iii. When client processes start up they must inform the server process that they exist by getting their process ID to the server.
 - iv. The whole system must be implemented using signals as the only inter-process communication mechanism.

Some code for the server is shown below. Show how to fully implement the server and provide the code for the client.

```
#include <stdio.h>
#include <sys/stat.h>
#include <signal.h>

int client_pids[100];
int num_clients = 0;

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

    void inform_children();
    int file_size;
    char *filename;

    filename = argv[1];
    struct stat statbuf;
    stat(filename, &statbuf);
    file_size = statbuf.st_size;

    while(1) {
        if(statbuf.st_size != file_size) {
            inform_children();
            file_size = statbuf.st_size;
        }
        sleep(1);
    }
}

void inform_children() {
```

(10 marks)

Continued on the next page

4. (a) Consider the following code:

```
#include <unistd.h>
#include <stdio.h>
#include <sys/types.h>

int main() {

    int p[2], i;
    char message[21];
    char recieved[21];
    char m[1];
    pid_t pid;
    pipe(p);
    for(i = 0; i < 10; i++) if((pid = fork()) == 0) break;

    if(pid == 0) {
        close(p[0]);
        sprintf(message, "Message from child %d", i);
        for(i = 0; i < 21; i++) {
            m[0] = message[i];
            write(p[1], m, 1);
        }
    }
    else {
        for(i = 0; i < 10; i++) {
            read(p[0], recieved, 21);
            printf("%s\n", recieved);
        }
    }
}
```

Describe the behaviour of the above program. Show the output that you would expect and explain why the output may vary for different executions of this program.

(7 marks)

- (b) Provide the C code to redirect the standard output of one process into the standard input of another process. Describe this code in detail.

(8 marks)

Continued on the next page

- (c) Consider the following code, in file q4c.c:
-

```

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

int main() {

    int fd;
    char buffer[1];

    fd = open("f1", O_RDWR);

    fd_set set, master;
    FD_ZERO(&master);
    FD_SET(0, &master);
    FD_SET(fd, &master);

    while(set=master, select(fd + 1, &set, NULL, NULL, NULL) > 0) {
        if (FD_ISSET (0, &set)) {
            read(0, buffer, 1);
            printf("B --> %c\n", buffer[0]);
        }
        if (FD_ISSET(fd, &set)) {
            read(fd, buffer, 1);
            printf("B --> %c\n", buffer[0]);
        }
    }
}

```

Now consider the following session:

```

[examiner@host q4]$ cc -o q4c q4c.c
[examiner@host q4]$ mknod f1 p
[examiner@host q4]$ echo "Hello world" > f1 &
[1] 27750
[examiner@host q4]$ q4c

```

Describe precisely what would happen if the above commands were executed, using the supplied program code.

Describe how you would implement your own version of the function `select(...)` using standard Unix system calls and C libraries. You do not need to provide all the code for `select(...)` but you must identify the key system calls and functions required to implement it, and show how they would be used.

(10 marks)

Continued on the next page

5. (a) Using an example scenario, show why record locking is important for multi-process environments. Demonstrate, using sample code, how record locking is implemented in C on Unix. **(7 marks)**
- (b) Illustrate, using diagrams and sample code, what is meant by *deadlock* in terms of record locking. To what degree is it prevented by Unix? Clearly describe a sample deadlock scenario that would not be prevented by Unix. **(8 marks)**
- (c) In many situations it is desirable that only one instance (i.e. process) of a program would run on a host at any point in time – for example only one instance of a server program.

Using your knowledge of the various C libraries and Unix system calls, describe clearly *three* different mechanisms you could employ when writing a program to ensure that only one instance of that program will be running at any point in time on any one host.

You should use diagrams and sample code in your answer.

(10 marks)

6. (a) Explain clearly what is meant by a *facility key* in terms of inter-process communication facilities. Show how it is possible can ensure that separate processes use the same key by relating the key to an ordinary file. **(5 marks)**
- (b) Give a detailed account of any *one* of the following inter-process communication mechanisms:
- i. Message queues
 - ii. Semaphores
 - iii. Shared memory

You should use diagrams where appropriate to explain the theory, and sample code to demonstrate how inter-process communication is implemented on Unix using your chosen mechanism.

(10 marks)

- (c) Show how it is possible to implement *connection-oriented* and *connectionless* communication between processes on separate hosts.

Describe the circumstances under which connection-oriented or connectionless communication would be appropriate.

(10 marks)