Northeastern University
College of Engineering
Department of Electrical & Computer Engineering

EECE7376: Operating Systems: Interface and Implementation

Homework 6

Instructions

- For programming Problems:
    1. Your code must be well commented by explaining what the lines of your program do. Have at least one comment for every 4 lines of code.
    2. At the beginning of your source code files write your full name, students ID, and any special compiling/running instruction (if any).
    3. Before submitting the source code file(s), your code must compile and tested with a standard GCC compiler (available in the CoE Linux server).
    4. Do not submit any compiled object or executable files.
- Submit the following to the homework assignment page on Canvas:
    1. Your homework report developed by a word processor (e.g., Microsoft Word) and submitted as one PDF file. The report cover page must include your full name, student ID, course/section/semester information. For answers that require drawing and if it is difficult on you to use a drawing application, which is preferred, you can neatly hand draw "only" these diagrams, scan them, and insert the scanned images in your report document. The report includes the following (depending on the assignment contents):
        a. Answers to the non-programming Problems that show all the details of the steps you follow to reach these answers.
        b. A summary of your approach to solve the programming Problems.
        c. The screen shots of the sample run(s) of your program(s)
    2. The source files (i.e., the .c, .h, or .S files) that contain your well-commented code.

Do NOT submit any files (e.g., the PDF report file and the source code files) as a compressed (zipped) package. Rather, upload each file individually.

Note: You can submit multiple attempts for this homework, however, only what you submit in the last attempt will be graded. This means all required files must be included in this last submission attempt.

Problem 1 (20 Points)

The following table represents the current state of a system in which four resources *A*, *B*, *C* and *D* are needed by the shown four processes. The system contains the following total instances of each resource: 6 of *A*, 4 of *B*, 4 of *C*, 2 of *D*.

Processes	Allocation				Max				Need				Available			
	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D
P_0	2	0	1	1	3	2	1	1								
P_1	1	1	0	0	1	2	0	2								
P_2	1	0	1	0	3	2	1	0								
P_3	0	1	0	1	2	1	0	1								

Answer the following questions using the Banker's algorithm:

a) Fill the missing entries in columns *Need* and *Available* in the above table.
b) Is the system in a safe state? Explain why.
c) Can a request from P_2 of (2,2,0,0) be granted? Explain why.

Problem 2 (30 Points)

Write a C program that opens a file, with the `open()` system call, and then calls `fork()` to create a child process.

a) Test whether both the child and parent can access the same physical file using the descriptor returned by `open()`. Explain why (given that the parent and child each has their own stack and data segments).
Hint: you can use the following code to write different strings from the child and the parent to the file:

```
write(fd, "Child\n", 6);
…
write(fd, "Parent\n", 7);
```

Where `fd` is the file descriptor returned by `open()`.

b) What happens when both the parent and the child are writing to the file concurrently?
Hint: Test that by writing several times from within the parent and the child with `sleep(1)` in between each writing.

Attach your full program in a file named `hw6pr2.c`

Problem 3 (50 Points)

The goal of this problem is implementing a small file manipulation command-line tool called `files` to perform different actions on existing files. Its first command-line argument specifies the action to be performed by the tool, and the rest of the command-line arguments are interpreted based on that particular action. This is the list of supported actions, and their arguments:

- `./files info <file>`. Provide information about an existing file that is passed in argument `<file>`. The information should include the **inode** number of the file, the size of the file in bytes, and its access permissions. Permissions should be provided in the same format as command `ls -l`, excluding the left-most character that identifies the inode type (for example: `rw-r--r--`). You can check the documentation for system call `stat` (`man 2 stat`) for details on how to extract this information.
- `./files link <src> <dest>`. Create a new hard link `<dest>` for file `<src>`.
- `./files symlink <src> <dest>`. Create a new soft link `<dest>` for file `<src>`.
- `./files rm <file>`. Remove `<file>`. Here you need to make sure that your code does not remove a directory if the given argument is for a directory instead of a file. You need to do that without checking for the argument "type".

In all cases, the tool should provide proper error messages if an action failed to execute, or if the tool is invoked with the wrong syntax. Function `perror()`, defined in the standard C library, can be useful here.

Attach your full program in a file named `hw6pr3.c`

Here are some execution examples:

```
$ ls
files files.c
$ ./files info hello.txt
Error: No such file or directory
$ ./files info files
Inode: 10753369
Size: 43
Permissions: rwxr-xr-x
$ ./files symlink files files2
$ ls -l
[ ... ] files
[ ... ] files2 -> files
```

Test your program with examples that cover all the problem requirements.