**Northeastern University**
**College of Engineering**
**Department of Electrical & Computer Engineering**

EECE7376: Operating Systems: Interface and Implementation

# Homework 6

## Info

Full name: Qucheng Jiang
Student ID: 001569593

## 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 1.a

A:2/6; B:2/4; C:2/4; D:0/2; # Available after Initial Allocation

# P_0 req(1,2,0,0) rls(2,0,1,1)

| Processes | Allocation | Max | Need | Available | Seq |
|---|---|---|---|---|---|

| | 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 | 1 | 2 | 0 | 0 | 2 | 2 | 2 | 0 | |
| $P_1$ | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 2 | 0 | 1 | 0 | 2 | | | | | |
| $P_2$ | 1 | 0 | 1 | 0 | 3 | 2 | 1 | 0 | 2 | 2 | 0 | 0 | | | | | |
| $P_3$ | 0 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 | 0 | 0 | 0 | | | | | |

# P_1 req(0,1,0,2) x cannot do that

| Processes | Allocation | | | | Max | | | | Need | | | | Available | | | | Seq |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 | 1 | 2 | 0 | 0 | 2 | 2 | 2 | 0 | 1 |
| $P_1$ | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 2 | 0 | 1 | 0 | 2 | 4 | 2 | 3 | 1 | |
| $P_2$ | 1 | 0 | 1 | 0 | 3 | 2 | 1 | 0 | 2 | 2 | 0 | 0 | | | | | |
| $P_3$ | 0 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 | 0 | 0 | 0 | | | | | |

# P_2 req(2,2,0,0) rls(1,0,1,0)

| Processes | Allocation | | | | Max | | | | Need | | | | Available | | | | Seq |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 | 1 | 2 | 0 | 0 | 2 | 2 | 2 | 0 | 1 |
| $P_1$ | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 2 | 0 | 1 | 0 | 2 | | | | | |
| $P_2$ | 1 | 0 | 1 | 0 | 3 | 2 | 1 | 0 | 2 | 2 | 0 | 0 | 4 | 2 | 3 | 1 | |
| $P_3$ | 0 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 | 0 | 0 | 0 | | | | | |

# P_3 req(2,0,0,0) rls(0,1,0,1)

| Processes | Allocation | | | | Max | | | | Need | | | | Available | | | | Seq |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 | 1 | 2 | 0 | 0 | 2 | 2 | 2 | 0 | 1 |
| $P_1$ | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 2 | 0 | 1 | 0 | 2 | | | | | |
| $P_2$ | 1 | 0 | 1 | 0 | 3 | 2 | 1 | 0 | 2 | 2 | 0 | 0 | 4 | 2 | 3 | 1 | 2 |
| $P_3$ | 0 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 | 0 | 0 | 0 | 5 | 2 | 4 | 1 | |

# P_1 req(0,1,0,2) rls(1,1,0,0)

| Processes | Allocation | | | | Max | | | | Need | | | | Available | | | | Seq |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 | 1 | 2 | 0 | 0 | 2 | 2 | 2 | 0 | 1 |
| $P_1$ | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 2 | 0 | 1 | 0 | 2 | 5 | 3 | 4 | 2 | |
| $P_2$ | 1 | 0 | 1 | 0 | 3 | 2 | 1 | 0 | 2 | 2 | 0 | 0 | 4 | 2 | 3 | 1 | 2 |
| $P_3$ | 0 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 | 0 | 0 | 0 | 5 | 2 | 4 | 1 | 3 |

# Finish, all processes cleared.

| Processes | Allocation | | | | Max | | | | Need | | | | Available | | | | Seq |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 | 1 | 2 | 0 | 0 | 2 | 2 | 2 | 0 | 1 |
| $P_1$ | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 2 | 0 | 1 | 0 | 2 | 5 | 3 | 4 | 2 | 4 |
| $P_2$ | 1 | 0 | 1 | 0 | 3 | 2 | 1 | 0 | 2 | 2 | 0 | 0 | 4 | 2 | 3 | 1 | 2 |
| $P_3$ | 0 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 | 0 | 0 | 0 | 5 | 2 | 4 | 1 | 3 |

## Problem 1.b

Yes, this is in a safe state. The sequence [P0, P2, P3, P1] satisfies safety criteria by applying the BA safety algorithm.

## Problem 1.c

A:2/6; B:2/4; C:2/4; D:0/2; # Available after Initial Allocation

# P_2 req(2,2,0,0)
# P_2 req(0,0,0,0) rls(3,2,1,0)

| Processes | Allocation | | | | Max | | | | Need | | | | Available | | | | Seq |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 | 1 | 2 | 0 | 0 | | | | | |
| $P_1$ | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 2 | 0 | 1 | 0 | 2 | | | | | |
| $P_2$ | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | |
| $P_3$ | 0 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 | 0 | 0 | 0 | | | | | |

# P_3 req(2,0,0,0) rls(0,1,0,1)

| Processes | Allocation | Max | Need | Available | Seq |
|---|---|---|---|---|---|

| | 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 | 1 | 2 | 0 | 0 | | | | | |
| $P_1$ | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 2 | 0 | 1 | 0 | 2 | | | | | |
| $P_2$ | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 1 |
| $P_3$ | 0 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 | 0 | 0 | 0 | 3 | 2 | 3 | 0 | |

# P_0 req(1,2,0,0) rls(2,0,1,1)

| Processes | Allocation | | | | Max | | | | Need | | | | Available | | | | Seq |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 | 1 | 2 | 0 | 0 | 3 | 3 | 3 | 1 | |
| $P_1$ | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 2 | 0 | 1 | 0 | 2 | | | | | |
| $P_2$ | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 1 |
| $P_3$ | 0 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 | 0 | 0 | 0 | 3 | 2 | 3 | 0 | 2 |

# P_1 req(0,1,0,2) rls(2,0,1,1)

| Processes | Allocation | | | | Max | | | | Need | | | | Available | | | | Seq |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 | 1 | 2 | 0 | 0 | 3 | 3 | 3 | 1 | 3 |
| $P_1$ | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 2 | 0 | 1 | 0 | 2 | 5 | 3 | 4 | 2 | |
| $P_2$ | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 1 |
| $P_3$ | 0 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 | 0 | 0 | 0 | 3 | 2 | 3 | 0 | 2 |

# Finish, all processes cleared.

| Processes | Allocation | | | | Max | | | | Need | | | | Available | | | | Seq |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 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 | 1 | 2 | 0 | 0 | 3 | 3 | 3 | 1 | 3 |
| $P_1$ | 1 | 1 | 0 | 0 | 1 | 2 | 0 | 2 | 0 | 1 | 0 | 2 | 5 | 3 | 4 | 2 | 4 |
| $P_2$ | 3 | 2 | 1 | 0 | 3 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 1 |
| $P_3$ | 0 | 1 | 0 | 1 | 2 | 1 | 0 | 1 | 2 | 0 | 0 | 0 | 3 | 2 | 3 | 0 | 2 |

Yes, request from $P_2$ of (2,2,0,0) can be granted. The sequence [P2, P3, P0, P1] satisfies safety criteria by applying the BA safety algorithm. System still in a safe state.

## **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);
```

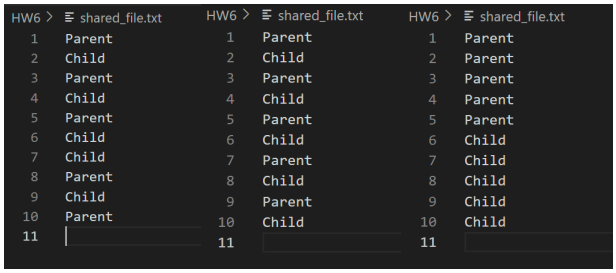
Where `fd` is the file descriptor returned by `open()`.

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`

a. Child and parent can access the same physical file.
When fork() is called, the child process receives its own copy of the file descriptors, but these copies of fd still point to the same file table entries in OS kernel as their parent's copies. Both parent and child are interacting with the same file through their respective file descriptors.

b. Result:
with sleep(1), with sleep(1), without sleep(1).




The writing syscalls will write to the file in sequence that based on time and os scheduler.

## **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.