CS 451/551 Quiz 4 Annotated Solution

Date: Mar 21, 2016 **Max Points**: 15

Important Reminder As per the course Academic Honesty Statement, cheating of any kind will minimally result in receiving an F letter grade for the entire course.

1. Which combination of `open()` flags can be used to implement a locking ADT?

   a) `O_RDONLY|O_CREAT`

   b) `O_WRONLY|O_CREAT`

   c) `O_RDONLY|O_CREAT|O_APPEND`

   d) `O_WRONLY|O_CREAT|O_APPEND`

   e) `O_WRONLY|O_CREAT|O_EXCL`

   Answer: (e).

   A prerequisite for locking is a check for a state and an action to undo the state where the check and the action must be **atomic**. The only of the above combinations which meets this prerequisite is (e), where the state is the non-existence of a file and the action is creating the file.

2. Which of the following statements is necessarily true about a orphan process which is still running but whose parent has terminated?

   a) It is a zombie process.

   b) It cannot have a controlling terminal.

   c) It will have been adopted by the `init` process.

   d) It is a daemon process.

   e) It must have one or more active child processes.

   Answer: (c).

   A zombie is a process which has terminated but has not yet had it's status reaped; this rules out (a). A orphan process may still have a controlling terminal and will still receive signals associated with its parent's controlling terminal; hence (b) is false. An essential characteristic of a daemon is that it does not have a controlling terminal; hence (d) is false. Finally, any process including orphans may have 0 active child processes; this rules out (3). Hence (c) is the only possibiility; in fact, it is the case that orphan processes are indeed adopted by `init`.

3. Which of the following octal umask values ensures that permissions for newly created files ensures that other will not have any access?

   a) 0777

   b) 022

   c) 011

   d) 044

   e) 0000

   Answer: (a).

   If the umask has a bit turned on, then the corresponding permission is turned off. For other permissions, we need to look at the rightmost octet and to ensure that all permissions are turned off it must have the value 7. The only of the above alternatives with 7 as the rightmost octet is (a); in fact, it turns off all permissions for user, group and other (a fairly unusual umask).

4. Which of the following statements is **clearly false**?

   a) Without O_NONBLOCK, a open() of a FIFO for read-only blocks until some process opens the pipe for writing.

   b) With O_NONBLOCK, a open() of a FIFO for read-only returns immediately.

   c) Without O_NONBLOCK, a open() of a FIFO for write-only blocks until some process opens the pipe for reading.

   d) If a process has a pipe open for reading with the O_NONBLOCK flag off, then any attempt to read the pipe when it is empty will block.

   e) If a process has a pipe open for writing with the O_NONBLOCK flag off and the pipe has no reader processes, then any attempt to write the pipe when it is full will block.

   Answer: (e).

   Behaviors (a), (c) and (d) are clearly desirable if blocking pipes are to be self-synchronized. (b) is pretty much the definition of what O_NONBLOCK should do. That leaves (e).

   Consider:

   ```
   grep SOME_COMMON_REGEX very-large-file | head
   ```

   If a complete grep on very-large-file produces a large number of hits, and head only outputs a small prefix of those hits before terminating, then it would be useful if grep was automatically terminated without needing to be aware of the fact that its standard output stream is dangling. This is achieved by having a write to the pipe result in a SIGPIPE signal after head has closed it's read end; the default action on the signal is to terminate the grep process without requiring any awareness of the basic `grep` code that it is writing to a pipe.

Hence (e) is false.

Update: Alternative (d) should have been phrased:

> If a process has a pipe open for reading with the `O_NONBLOCK` flag off, then any attempt to read the pipe when it is empty will block **if no process has ever closed the write end**.

Since a read of a empty pipe whose write end has been closed will succeed returning 0, (d) as originally phrased can be false. Hence a choice of (d) as the answer will also be regarded as correct.

5. A directory `d` and file `d/f` within it have the same user owner. The user owner cannot create any files within `d` but can write the contents of `d/f`. Which of the following `ls -l` permissions for `d` and `f` are consistent with this behavior?

   a) `d` has permissions `d-wxrwxrwx` and `f` has permissions `---xr-xr-x`.

   b) `d` has permissions `drwxrwxrwx` and `f` has permissions `-rwxr-xr-x`.

   c) `d` has permissions `dr-xr-xr-x` and `f` has permissions `-r-xr-xr-x`.

   d) `d` has permissions `d--x--x--x` and `f` has permissions `--wxr-xr-x`.

   e) `d` has permissions `d-wxrwxrwx` and `f` has permissions `-r-xr-xr-x`.

   Answer: (d).

   Owner behaviors are governed by the leftmost 3 read-write-execute permission bits in the `ls -l` listing. The stated behaviors require `*-x` for `d` and `*w*` on `f` where `*` means that the corresponding permission is not specified. (c) and (d) are the only alternatives which have `*-x` on `d`, whereas (b) and (d) are the only alternatives which have `*w*` on `f`. The intersection is alternative (d).

6. The `setgid` bit on a file affects the operation of:

   a) The `fork()` system call.

   b) The `exec()` family of system calls.

   c) The `wait()` family of system calls.

   d) The `stat()` family of system calls.

   e) The `open()` system call.

   Answer: (b).

   The `setgid` bit on a file controls the effective gid of a process which `exec()'`s that file.

7. Assume that a program performs the following steps:

```
FILE *f = fopen("file", "w");
int fd = fileno(f);
write(fd, "abc", 3);
lseek(fd, 0, SEEK_SET);
write(fd, "", 1);
FILE *f1 = fopen("file", "r");
int c = fgetc(f1);
```

Assuming that none of the calls fail, the value of `c` will be:

a) `’\0’`

b) `’a’`

c) `’b’`

d) `’c’`.

e) `EOF`

Answer: (a).

The `write()` will write 3-bytes `’a’`, `’b’` and `’c’` into the file, but the subsequent `lseek()` and `write()` will replace the first byte in the file with the `’\0’` `NUL`-terminator for the empty string `""`. Hence the subsequent `fgetc()` which reads the first byte in the file will return `’\0’`.