CS 451/551 Quiz 5 Annotated Solution

Date: Apr 13, 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 one of the following statements about a *dynamically-linked executable* is necessarily **false**?

   a) The executable does not contain all the code which will be needed at runtime.

   b) The executable will be smaller in size than the corresponding statically-linked executable.

   c) The executable will contain references to the code needed from dynamic libraries.

   d) At runtime, the executable need not be linked with exactly the same dynamic libraries it was compiled against.

   e) The executable will contain all the necessary code from the dynamic libraries.

   Answer: (e)

   A dynamically linked executable does not contain the library code; that is what makes it different from a statically linked executable which does contain the library code. Hence (e) is clearly false.

   (a) is true since the dynamic library code is not included in the executable.

   Since a statically-linked executable contains all the library code, (b) is true.

   The dynamically-linked executable will contain references to the code from the dynamic library making (c) true.

   (e) is true since at runtime the executable need be linked only with a library which is *compatible* with the library it was compiled against.

2. Which one of the following statements about a *daemon process* is necessarily **true**?

   a) It is a zombie.

   b) It cannot have a controlling terminal.

   c) It does not have a parent process.

   d) It must be monitoring some well-known IPC facility in order to respond to client requests.

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

Answer: (b)

The essential characteristic of a daemon is that it cannot have a controlling terminal.

A zombie is a process which has terminated but its parent has not done a wait for it; a daemon is still running; hence (a) is false.

All user processes have a parent process; a daemon will have the `init` process as its parent; hence (c) is false.

A daemon need not necessarily be a server process and not need to communicate with clients; hence (d) is false.

The daemon need not even be a server or it may be a *iterative server*; in both cases, it need not have any child worker processes; hence (e) is false.

3. Specifying `O_NONBLOCK` for a file descriptor will have no effect if the file descriptor refers to a

   a) Terminal.

   b) FIFO.

   c) Message queue.

   d) Regular file.

   e) Anonymous pipe.

   Answer: (d)

   Specifying `O_NONBLOCK` for a terminal, message queue, anonymous or named pipe can have an effect since those devices can block. Since regular files cannot block, the `O_NONBLOCK` flag will have no effect if the file descriptor refers to a regular file.

4. Which of the following is **not** useful for monitoring multiple input file descriptors to allow responding to the first descriptor which is ready?

   a) Multiple Unix processes.

   b) Multiple Unix threads.

   c) File locking.

   d) The use of the `O_NONBLOCK` flag.

   e) The `select()` call.

   Answer: (c)

Concurrent monitoring of multiple file descriptors can be achieved by have multiple threads of control using multiple processes (a) or multiple threads (b), or by using the `select()` call (c) to block on the descriptors being monitored or by using a busy wait loop successively polling the multiple descriptors using the `O_NONBLOCK` flag. File locking does not allow any easy direct aid for this.

5. For **mandatory** file locking:

   a) The `F_MAND` flag must be specified in the `fcntl()` call.

   b) The `flock()` call must be used.

   c) The `setgid` bit on the file must be set.

   d) The `setuid` bit on the file must be set.

   e) The sticky bit on the file must be set.

   Answer: (c)

   Mandatory locking can be enabled on a file which is not group-executable by setting the `setgid` bit as long as the filesystem on which the file resides is set up to allow mandatory locking.

6. A private anonymous memory mapping can be used for

   a) IPC between related processes.

   b) IPC between unrelated processes.

   c) Implementing file read/write operations using memory operations.

   d) Allocating large blocks of zero-filled memory.

   e) For interacting with I/O devices using memory-mapped I/O.

   Answer: (d)

   Since the mapping is private, (a), (b) and (e) (sort-of for the last) are ruled out. Since the mapping is anonymous (c) is ruled out.

7. Given the following pseudo-code sequence:

```
1: int fd1 = open(...);
2: //lock entire file via descriptor fd1
   fcntl(fd1, F_SETLK, { .l_type = F_WRLK, .l_whence = SEEK_SET,
                         .l_start = 0, .l_len = 0 });
3: int fd2 = dup(...);
4: close(fd1);
5: //lock entire file via descriptor fd2
   fcntl(fd2, F_SETLK, { .l_type = F_WRLK, .l_whence = SEEK_SET,
                         .l_start = 0, .l_len = 0 });
```

a) The code will deadlock on line (5) since the file is already locked from the lock on line 2.

b) The `close()` on line (4) will fail since the file is locked.

c) The lock attempt on line (5) will fail since the descriptor was closed.

d) The lock attempt on line (5) will succeed.

e) The `dup()` on line (3) will fail since the file is locked.

Answer: (d)

Before the quiz it was announced that the `dup(...)` on line (3) should read `dup(fd1)` and that you should not assume any failures other than those explicitly listed in the alternatives.

The `close()` on line (4) will release all locks on the file. Hence the lock attempt on line (5) should succeed on re-locking the entire file.