ECS 150 Homework 1

Hardy Jones
999397426
Professor Levitt
Winter 2015

1. Of the options, the only one that should be able to run outside of kernel mode should be reading the time. Disabling all interrupts at the user level could allow malicious programs to prevent a user from stopping them. Changing the memory map could allow malicious programs to gain access to to memory they should not be allowed. Setting the date changes the date for all users in the system.

   Reading the date command is something that user's do all the time and does not affect other users or allow malicious activity.

2. Since the user with UID 6 has the same GID as the owner of the file, and the file has execute permissions for the group, the file will be executed.

3. There has to be someone administering the system. The ability to create, update and delete users should be relegated to one user rather than allowing all users to have this control. Altering system-wide settings should also be relegated to one user. Installing, updating and deleting system-wide software should also be the job of one user.

4. A process table is needed with timesharing because each process can be halted by the operating system at any time. The operating system needs to keep track of the state of each process when it is halted, so that this state can be restored when the process is ready to be executed again.

5. User 2 will be able to read the file. The *link* command just adds another entry to the i-node table, MINIX only deletes files once no more entries in the i-node table point to the file. Since User 2 still has an entry in the i-node table, the file will still exist on the file system.

6. *chroot* is only available for the superuser because a superuser might want to restrict the available file system that a user can access. If a regular user could execute *chroot*, then they could just override the superuser's initial call to *chroot*. This ability would limit the effectiveness of *chroot*

7. It means that the *set-user-id* and *set-group-id* permissions are removed when the owner is changed.

   If the *set-user-id* or *set-group-id* bits remained set when the user transfered ownership of a file, then a malicious program could be transfered to anyone and then executed with the new owner's *effective user-id*.

10. Since each system call to *read* can read a maximum of the blocksize on each call, the OS will have to perform multiple calls each time to fill the buffer.

    For the first call, the OS reads all of block 1. This puts 512 bytes in the buffer. The buffer can still take another byte, so the OS reads the next block. It uses just the first byte to fill the buffer. So far there have been two *read* system calls.

    The next time the OS reads into the buffer it needs to get data from block 2. It reads block 2 from byte 2 into the buffer. This is 511 bytes, so the OS needs to read 2 more bytes into the buffer. It performs another *read* system call in block 3. This call puts the next 2 bytes into the buffer, filling it. At this point there have been four *read* system calls.

    The next time the OS reads into the buffer it needs to get data from block 3. It reads block 3 from byte 3 into the buffer. This is 510 bytes, so the OS needs to read 3 more bytes into the buffer. It performs another *read* system call in block 4. This call puts the next 3 bytes into the buffer, filling it. At this point there have been six *read* system calls.

    The final time the OS reads into the buffer it needs to get data from block 4. It reads block 4 from byte 4 into the buffer. This is 509 bytes and all of the data that is necessary to be read. At this point there have been seven *read* system calls.

    So all total, there are seven *read* system calls.

    This is not very efficient. If we could read the entire buffer in at once, we would only need to make four *read* system calls. Alternatively, if the OS caches *read* system calls, we could use the cache to subvert additional system calls.

11. Given the following values:

Process	PID
mypid	x
myparentspid	y
actualchildpid	z

    This program should print

    ```
    ID(myPID_1): z, ID(myPID_2): x and ParentPID: y
    ID(myPID_1): x, ID(myPID_2): y and ParentPID: y
    ```

    Though the order is not guaranteed.