

# Lab One

---

Eric Stenton

Eric.Stenton1@Marist.edu

September 9, 2019

## 1 PROBLEM ONE

**Question:** What are the advantages and disadvantages of using the same system call interface for manipulating both files and devices?

**Answer:** Using the same system call interface for manipulating both files and devices has both the advantages of a single interface to deal with the similar file and device system calls, but also the disadvantages of needing to be generalized to the extent that certain functionality may be lost. According to the textbook, the difference between files and I/O devices in terms of their commands/system calls is minimal to the extent that many operating systems such as UNIX merge them into a combined file-device structure[1]. Due to this similarity, the more specific details of dealing with files and devices can be abstracted to use the same set of system calls and be interacted with in a similar way reducing development time. Such a design makes the file-device structure simple, albeit monolithic. The monolithic nature of the structure offers the disadvantage of not handling complexity and specific differences between files and devices. Since new devices can be added similar to files, future ones may require different system calls than the single system call interface can handle leading to a drop in certain functionality and performance. In these cases, a separate system call interface with the supported system calls/commands would perform better than a combined file and device one.

## 2 PROBLEM TWO

**Question:** Would it be possible for the user to develop a new command interpreter using the system call interface provided by the operating system? How?

**Answer:** Since the system call interface acts as the mediator between commands specified in a command interpreter and the related system calls that carry out the desired action in kernel mode, a user could indeed develop a new command interpreter that interacts with the existing system call interface of the OS. According to the textbook, there are two common command interpreter designs. The first design describes a command interpreter that houses the code for each command and supports communication with the system call interface in its own code. The second describes a slimmer command interpreter that references a location where system programs are stored as files and are called to carry out the user defined system calls[1]. The latter design is common in Linux machines where commands such as 'ps' and 'ls' are files with independent code. The user could develop a command interpreter of either design to interact with the system call interface since the user has access to this functionality in user mode. The code, whether within the command interpreter or in a dependent file, will exert a system call interrupt which will be placed on a queue and eventually be managed by the interrupt service routine (ISR); the user program's desired actions, in this case the command interpreter's request, will be carried out in kernel mode. The Window's PowerShell is an example of a command interpreter that was later added to Microsoft's operating system that interacts with the system call interface similar to the Linux shell.

## REFERENCES

- [1] A. Silberschatz, P. B. Galvin, and G. Gagne. *Operating System Concepts (9. ed.)*. Wiley, 2013.