Davon Grant

HW #2

**Ex. 1.5 - What is the role of the magic number for binary executable files? Where is it stored?**

The magic number for binary executable files is used as a file signature. It is used to identify or verify the content of files, and it is stored in the beginning of a file.

**Ex. 1.18 — The term context-switching is used to describe switching from one program to another in execute. In here what is the “context” being switched?**

The context that is being switched is the registers and the program counter in the CPU. The registers get loaded with the new program and the PC switches to a new instruction.

**Ex. 2.3 — On Page 56, we discussed three different ways a process can be created. The third case, in which a currently running process creates a new process, in general covers the first two cases. Explain why.**

The third case covers the first two cases because when a human user double-clicks a program icon, they are essentially telling the OS to do a fork() command and create a new process. Also, when a currently running process creates a sub-process prompted by an external event, it calls the fork() instruction to create that new process. Fork() is always used to create a new process, which is why all of these are essentially the same.

**Ex. 2.4 — In the example of fork() system call in Figure 2.1, which line is executed as the very first line by the child process and why?**

The first line that is executed is “I am the parent; the child’s ID is 4 and val is -1.” This is because when the child process is first created with fork, it is identical to the parent. It isn’t until the image gets replaced that it becomes its own unique process.

**Ex. 2.5 — Study how calling and returning from a subroutine in a program written in C (or most procedural languages) and compare how this technique is related to calling a system call and returning from it. Hint: you can search for terms like “compilers,” “activation record,” “calling and returning from a subroutine.” This is a hard question but one with a substantial benefit.**

A subroutine generally involves many system calls, while a system call is a single interrupt into the kernel code. System calls interact directly with the OS code. System calls happen more often than subroutine calls, but programmers do not have to use system calls because the libraries within C often handle them for us.

**Ex. 2.16 — Discuss pros and cons of implementing a thread system at the kernel level or at the user level. Is implementing a thread system at the user level useful anyway, since the kernel can only see I/Os are done at the process level?**

Implementing a thread system at the kernel level should only be done for light-weight threads, as it is directly managed by the OS. User level threads exist in C instead of the OS, and they are much easier to use for programmers. However, user level threads can still call into the kernel level, but kernel level threads can only run in kernel level.

**Ex. 2.17 — Write a pseudo-code for exec(), fork(), waidpid(), and exit() system calls. Explain how these system calls are inter-related.**

fork(){

Check if there is available resources

Store PCB information of current process

Create a new process

Insert PCB information from current process into this new process

Put this new process into the ready queue

}

exec(){

Find core image of current process

Replace information with new core image

Put process into the ready queue

}

waitpid(){

Set program that is waiting to blocked

Read current process and see if it is still executing

While it is still executing

Keep waiting program in blocked

When it is done executing, set program that was waiting to running

}

exit(){

Terminate program

Release all resources held

}