Name: Wan Muhammad Rayhan Arwindra

NPM: 1806241210

This week, we learned about make, stdio, and I/O buffering. The make command actually executes a Makefile file. The Makefile file contains a set of instructions which is used to automate tasks, such as to compile multiple programs, deploying and running server from a backend web framework project, and so on. This file can be run by typing in "make", and could optionally be added with a target to specify which rule to run.

Makefile contains instructions grouped into "rules", which contains a target file, dependencies, and the commands. To execute a certain rule, we can write in the shell "make [TARGET_NAME]", where TARGET_NAME is the name of the target within the rule. If we simply write "make" without a target name, then we will execute the default target, which is the first target written in the make file. We can also write variables within the file with the command [VARIABLE_NAME] := [VALUE], and we can call them by using ${VARIABLE_NAME}.

In Linux, we can get information about our system's limitation by using the getconf command. This command outputs to the standard output information queried by the variable specified. For example, to get the maximum length of a login username, we can use getconf LOGIN_NAME_MAX. The resulting output would be the maximum length of a login username. We can also call getconf with the flag -a, which will return all configuration variables.

When we use I/O commands such as read() and write(), we actually utilize a buffer, which acts as an interface between library calls and the I/O system calls. For example, if we call the function printf(), it utilizes the write() system call. So first, the data to be written is put into a buffer, and afterwards the data within the buffer is forwarded into the write() system call. We can force the buffer to flush its data by using the fflush() command. We can also flush to a physical storage (such as a hard disk) by using the fsync() command.

Buffering is useful to reduce the number of I/O system call, as the data is accumulated into the buffer until its full (unless fflush() is called). This is useful because I/O system calls can be quite expensive, so repeatedly calling them would be costly. Thus, by using file buffering, we can improve the performance of the program, especially if it's writing or reading a large file, or a large number of small reads or writes.