Name: Wan Muhammad Rayhan Arwindra

NPM: 1806241210

For this week, we learnt about system calls, what they are, how they work, and several examples of them. System calls are the gateway to the kernel's services, it provides a way for application/user applications to gain access to the kernel's services and make use of their functionalities. System calls are also one way of switching between user mode and kernel mode.

An application program doesn't directly interact with the system call service routine to get their services. Instead, it uses a wrap function in the user mode, which then prepares the requested function and register to be used by the system call. Afterwards, the wrapper function forwards the information to the trap handler, which switches the system from user mode to kernel mode, and arranges what function is required from the system call. It then receives the function from the system call service routine, and executes it in the register designated by the wrapper function. Finally, the trap handler then switches the system back to user mode.

There's a significant difference between a library function and a system call. A library function is essentially part of our program's process, just like calling a function or method which we created ourselves. Library functions doesn't necessarily require us to switch from user mode to kernel mode in order to carry out its functionality, but in certain cases it might do. For example when we use C language and call the function malloc, if we still have enough space inside the process memory, then it doesn't require a system call. If not however, then it must call the brk system call to get the kernel's service to reserve more memory from the global virtual memory.

In some cases, invoking a system call might result in an error. The most common indication of a system call error is that it returns the value "-1". However, there exists a couple of system calls that will return -1 even though it is successful. Therefore, we need another way of finding an error, which is by using errno or error number. <errno.h> is a header file which consists of all error numbers and its corresponding error. Thus, before making a system call, we can set our errno to zero, then after the system call has been made and it returns something which might signify an error (-1 for example), then we can check if our errno is now non zero or not. If it is, then an error has occurred, if not then our system call is successful.