Simon Li

ECSE 427

Graded exercises 1

Question 1

1. When a process makes a system call through the kernel API library with an address as argument, does the validity of that argument need to be checked in the library, in the kernel or in both places? Justify your answer.

the validity of that argument needs to be checked in both places - the library and the kernel.

The library should perform basic validation checks on the address argument, such as ensuring it is not a null pointer or checking that it points to a valid memory location in the process's address space. These checks are necessary to prevent passing invalid memory addresses to the kernel, which can result in undefined behavior, crashes, or security vulnerabilities.

However, even if the library performs these checks, the kernel still needs to perform its own validation when it receives the argument from the library. The kernel operates in a privileged mode and has direct access to the system's hardware resources, so it can perform more rigorous validation checks that are not possible in user space. The kernel can check if the address is valid and accessible in the process's memory space, as well as check for other security concerns such as potential buffer overflows.

Overall, both the library and the kernel need to perform their own validation checks on the address argument to ensure the system is secure and operating correctly. Failing to perform these checks can lead to serious consequences such as system crashes, data loss, and security breaches.

Questioin 2

1. The race condition occurs because the variable "available\_resources" is shared among multiple processes, and multiple processes can simultaneously access and modify its value without proper synchronization.
2. The race condition occurs in the "decrease\_count" function where multiple processes can simultaneously access and modify the "available\_resources" variable. Specifically, the race condition occurs when multiple processes call the "decrease\_count" function at the same time and evaluate the condition "available\_resources < count" as true, but before any of them can decrement the "available\_resources" variable, another process decrements it, leading to more resources being allocated than are actually available.

#define MAX\_RESOURCES 5

int available\_resources = MAX\_RESOURCES;

pthread\_mutex\_t mutex = PTHREAD\_MUTEX\_INITIALIZER;

int decrease\_count(int count){

pthread\_mutex\_lock(&mutex);

if (available\_resources < count) {

pthread\_mutex\_unlock(&mutex);

return -1;

} else {

available\_resources -= count;

pthread\_mutex\_unlock(&mutex);

return 0;

}

}

int increase\_count(int count){

pthread\_mutex\_lock(&mutex);

available\_resources += count;

pthread\_mutex\_unlock(&mutex);

return 0;

}

Question 3