Simon Li
260984998
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.

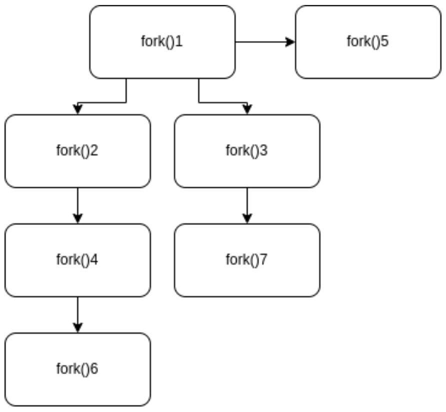
   the validity of that argument needs to be checked in both places - the library and the kernel.

   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.

2) 3times



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.

3)

```
#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

Virtual memory is 5 bits.

Physical memory is log2(16)=4bits.

3 segments so need log2(3)=2bits for segment in virtual address.

Then 5-2=3bits for offset

Virtual address	seg	offset	base	physical
(00000)	(00)	(000)	12	12(1100)
(00111)	(00)	(111)	12	Fault, offset too big
(11111)	(11)	(111)	Fault, not a segment	NA
(10100)	(10)	(100)	1	Fault, offset too big

(00011)	(00)	(011)	12	15(1111)

Question 4

1) 4.3 seconds
2) 4GB =2^32B; 4GB/2^2B*4ns = 2^30 ns = 4.295s

Question 5

1) Given that the page size is 4 Kbytes, the page offset will be 12 bits (2^12 = 4096). Therefore, the remaining 20 bits will be used to index the page tables. Since the top-level page table has 1024 entries, each entry will point to a second-level page table. Therefore, the second-level page table size will be equal to the page size (4 Kbytes) divided by the size of a page table entry, which is 4 bytes (32 bits). number of entries in a single second-level page table will be: (4 Kbytes)/(4 bytes) = 1024 entries
2) total size of the address space is 18 Mbytes + 27 Mbytes (90 Mbytes - 63 Mbytes) = 45 Mbytes. Entries 0 to 4 for the range from 0 to 18 Mbytes. Entries 22 to 29 for the range from 90 to 117 Mbytes.

   from 0 to 18 Mbytes, we have 4500 pages. Therefore, we need ceil(4500 / 1024) = 5 second-level page tables to represent this range.

   from 90 to 117 Mbytes, we have 6750 pages. Therefore, we need ceil(6750 / 1024) = 7 second-level page tables to represent this range.

   Total, we need a minimum of 5 + 7 = 12 second-level page tables to represent the program with a sparse address space.