Graded Exercises
COMP 310 / ECSE 427 Winter 2025

1 Syscalls [10 points]

1.1 Syscall Wrappers [5 points]

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

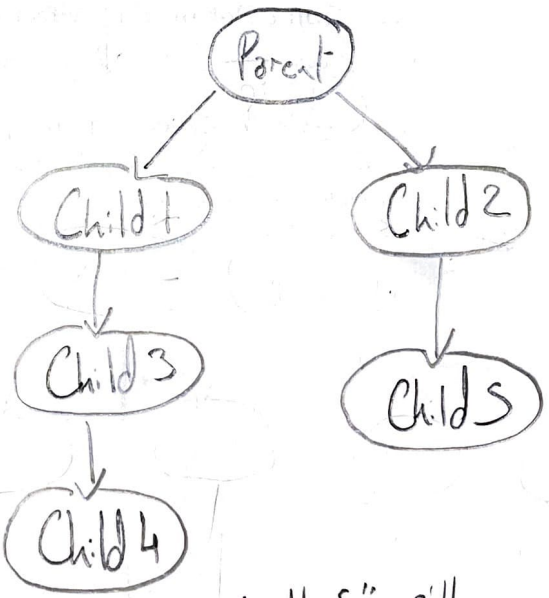
The validity of that pointer should be checked in the kernel. System Calls switch to kernel mode, so libraries in User-space lack access to address space permissions for validation.

1.2 Forking [5 points]

How many times will the below program print `hello5`? Explain why (e.g. by drawing the process tree).

```
1  int main() {
2    if (fork() != 0) {
3      if (fork() != 0) {
4        printf("hello1\n");
5      } else {
6        fork();
7      }
8      printf("hello2\n");
9    } else {
10     if (fork() != 0) {
11       printf("hello3\n");
12     } else {
13       fork();
14     }
15     printf("hello4\n");
16   }
17   printf("hello5\n"); // <-- this one!
18 }
```

↳ "hello 5" will print 6 times



↳ Since "hello 5" follows after all the forks, and inside no conditional code blocks, every process will print "hello 5" once.

2 Synchronization [10 points]

Assume that a finite number of resources of a single type must be managed in a multi-threaded program. Threads acquire a number of these resources and – once finished – return them. The following program segment is used to manage a finite number of instances (`MAX_RESOURCES`) of an available resource. When a thread wishes to obtain a number of resources, it invokes the `decrease_count` function. When a thread wants to return a number of resources it calls the `increase_count` function.

```
1  #define MAX_RESOURCES 5
2  int available_resources = MAX_RESOURCES;
3
4  int decrease_count(int count) {
5    if (available_resources < count) {
6      return -1;
7    } else {
8      available_resources -= count;
9      return 0;
10   }
11 }
12
13 int increase_count(int count) {
14   available_resources += count;
15   return 0;
16 }
```

This is a non-atomic operation can lead to over-subtraction.

This is also non-atomic

2.1 A Race Everyone Loses [1 point]

Unfortunately, the above program fragment causes race conditions. Identify the data involved in the race condition(s).

↳ It's available_resources.

2.2 Don't Bet on the Trifecta [2 points]

Clearly identify the location(s) in the code where the race condition(s) occur.

2.3 Save the Track [7 points]

Fix the race condition. You can either write **pseudo-code** to replace the program above, or clearly explain how to modify the code. (You can refer to C library functions and values seen in class.)

We can use pthread_mutex_t thread and initialize it as a global variable.

↳ Use pthread_mutex_lock and pthread_mutex_unlock to enforce atomicity for both increment & decrement operations.

Eg. in the code above:

```
int decrease_count(int count) {
  pthread_mutex_lock(&mutex);   ← lock acquisition before operation
  int result = -1;
  if (available_resources >= count) {
    available_resources -= count;
    result = 0;
  }
  pthread_mutex_unlock(&mutex);   ← unlock after operation
  return result;
```

} condition check and update performed under same lock.

3 Compaction [5 points]

This question asks you to do a bit of math. Specifically, we want to see that you understand *what compaction is doing* and how that affects the scenario.[1] We do not need to see that you can accurately do arithmetic. Use a calculator if you wish, and give an approximate answer (say, two digits of precision).

3.1 Turbocharge the matter compressor [5 points]

(Yes, that is a Futurama joke.)
A memory management system eliminates holes by compaction. Assume that a random distribution of holes and many segments means that very nearly the entire memory must be copied. Assume we can read an 8-byte value from physical storage in 4 nanoseconds (ns) and that we can also write 8-byte values in 4ns.
How long does it take to compact 4GB of memory? Explain your reasoning.
Hint: does a copy require only a read, only a write, or both? Copying requires both read & writing.

Per each 8-byte block that is 4ns(read) + 4ns(write) = 8ns.

↳ For 4GB compaction, this is $\frac{4\,GB}{8\,bytes} \times 8ns = 4,294,967,296\,ns$

= 4.29 seconds

4 Virtual Memory: Segmentation [10 points]

For this question, all numbers prefixed by `0b` and in that font are in binary. Therefore, `0b100` is the decimal number 4.

A CPU boasts 5-bit virtual addresses and 16 bytes of byte-addressable physical memory.[2] There are three segments currently mapped, with the following segment table:

Seg#	Base	Bound	R W
0	0b1100	0b100	0 1
1	0b0000	0b000	0 0
2	0b0001	0b011	1 1
3	0b1000	0b010	1 0

4.1 Cartography [1 point each]

Give either the corresponding physical address, or the phrase "segmentation fault," for each of the following accesses. If the access faults, briefly explain why. (You may draw a diagram of the physical memory if that makes it easier to explain.)

1. `0b00000` read
2. `0b11001` read
3. `0b01000` read
4. `0b10010` read
5. `0b11010` read
6. `0b00111` write
7. `0b10001` write
8. `0b00000` write
9. `0b11000` write
10. `0b10100` write

[1] Note also that these numbers are not realistic.
[2] Quite the boast, huh? Don't tell them.

5 Virtual Memory: Paging [15 points]

We will consider two machines in this question. (There is a third part on the next page.) The first machine has 32-bit virtual addresses. The machine supports paging with a 4KB page size and a 2-level paging scheme. The page directory[3] contains 1024 entries.

5.1 This Sounds Oddly Realistic [2 points]

How many entries are there in a single inner-level page table? Justify your answer.

A 4KB page size = 12 bit offset. 1024 entries = 10 bits : 32-12-10 = 10 bits = 1024 entries

5.2 Spatial Locality [5 points]

A program running on this first machine is only sparsely using its address space. The addresses between 0 (inclusive) and 18MB (exclusive) are mapped. Additionally, the addresses between 90MB and 117 MB are also mapped.

What is the minimum number of inner-level page tables needed? Justify your answer and take special care to consider whether or not 90 and 117 are on page directory boundaries. (Again, we want to see that you know how to approach the problem, not that your arithmetic is accurate.)

- 0 - 18MB region:
  - $\left\lceil \frac{18MB}{4KB \times 1024 \text{ directory entries}} \right\rceil = 4 \leftarrow (0-4) \rightarrow 5$ inner page tables required.
- 90MB - 117MB region:
  - If 90 and 117 on page boundaries:
    $117 - 90 = \left\lceil \frac{27MB}{4KB \times 1024} \right\rceil = 6 \leftarrow (0-6) \rightarrow 7$ inner page tables required.
  - If 90 and 117 not on page boundary:
    $116 - 91 = \left\lceil \frac{25MB}{4KB \times 1024} \right\rceil = 6$

⇒ 12 inner page tables needed.

[3] Recall "page directory" is the name for the outer-level page table.