# CS 417: Operating Systems

## Spring 2025: Practice Midterm Exam

This exam is closed book and closed notes, except for one double-sided page of notes. All cell phones and laptops must be turned off. No calculators may be used.

You will have one hour and 25 minutes to complete the real exam; this exam is written to be a bit longer, and will likely require over two hours to complete.

## Part 1 True/False

For each statement below, circle whether it is true or false. You may add a brief explanation to explain your reasoning (which may be used to award partial points).

**Q1** A program's "stack" is always at the "top" of the address space (in the highest virtual addresses) and its heap at the "bottom" of the address space, each always growing towards the other.

True False

**Q2** Each process's process control block includes a copy of its TLB.

True False

**Q3** In limited direct execution, a process runs directly on the CPU until but is limited in the instructions it can run and CPU registers it can access.

True False

**Q4** One problem with paged virtual memory is fragmentation of physical memory.

True False

**Q5** In a basic segmented architecture with only a single “base” and single “bounds” register, virtual memory translation is performed by first checking if virtual address is in bounds, and then adding it to the “base” register value to get the target physical address.

True False

**Q6** System calls are just like normal library calls—there’s no difference between what happens on a system call, and what happens on a standard library call.

True False

## Part 2 Code Analysis

Read and analyze the following code snippet and answer the questions below. For multiple-choice answers, you may show your work and provide a justification in the space provided. Justifications will be used to award partial credit.

```c
#include <stdio.h>
#include <unistd.h>

int main() {
  int a = 0;
  for (int i = 0; i < 4; i++) {
    if (fork() == 0) {
      printf("Hello\n");
      a++;
      return 0;
    } else {
      printf("Hello\n");
      a++;
    }
  }
  printf("a is %d\n", a);
}
```

**Q7** How many times will the message "Hello\n" be displayed?

a. 2
b. 4
c. 6
d. 8
e. None of the above
f. All of the above

**Q8** What will be the final value of “a” as displayed in the final `printf` in the program?

a. Due to race conditions, “a” may have different values on different runs of the program.
b. 0
c. 4
d. 8
e. None of the above

# Part 3 Design Questions

## Tickless kernel

Both xv6 and the OSTEP book describe a hardware timer that is setup on boot to “tick” periodically, and used to interrupt processes that run for a long time and switch them out for other processes. In practice, most modern operating systems are “tickless”, meaning the timer is used on-demand, rather than continuously. This is a big advantage in systems where most processes are waiting for I/O most of the time where a tickless design means CPUs need not be woken up periodically when there is no work to do. Consider, for example, a smart-phone which spends most of its time waiting for a user to do something.

**Q9** How would you design a scheduler to use a timer in this manner?

**Q10** How would a tickless design impact the precision of the time quanta allowed for each process?

## Segments and Pages

Some CPU architectures have segment registers, while others rely on a TLB and page table to handle memory translation. Others, such as the VAX, have a hybrid and feature both segment registers and a TLB-based page table. In this question, we will explore the implementation of a virtual memory translation in a series of hypothetical architectures using TLBs and/or segment registers.

### A Simple TLB

Assume that you have a 16-bit address-space computer with single TLB with 32 rows, each of which has the following entries:

```
[virtual address ; physical address ; valid bit; protection]
```

Memory translation is performed by first checking the TLB to see if a translation is present; if it is absent, the hardware will trap into the OS, at which point the OS should populate the TLB with the correct translation. The OS is not subject to the TLB; all OS code will reference physical addresses directly, and cannot use virtual addresses.

**Q11** Describe the entries in a single-level page table for this "simple TLB" architecture. Include an explanation of what all bits in this page table do, and how they correspond to the entries in the TLB.

**Q12** Assuming 4KB pages and a 16-bit address space, how many physical pages would be mapped in this address space? You may write this as an expression, e.g. $2^4$ - 3, as we did not permit calculators on this test.

**Q13** To the nearest order of magnitude (e.g. 10, 100, 1000), how many megabytes of memory would it require to represent this page table physically?

**A Hybrid Architecture**

Assume you also have 9 new registers, defining three segments, each with a virtual-base, physical-base, and bound register. Memory translation now follows a new algorithm:

1. if the virtual address's highest-order bit is 0, use the TLB as before. Otherwise, use the segment registers
2. if virtual-address ≥ virtual-base-1 and virtual-address < bound-1, add virtual-address to physical-base-1 to derive physical address
3. if virtual-address ≥ virtual-base-2 and virtual-address < bound-2, add virtual-address to physical-base-2 to derive physical address
4. if virtual-address ≥ virtual-base-3 and virtual-address < bound-3, add virtual-address to physical-base-3 to derive physical address

Additionally, the operating system is now subject to virtual memory translation; there is no difference between OS and process memory translation. The system retains a notion of "privileged" mode, in which the segment registers can be written, and "unprivileged" mode, in which the segment registers can only be read.

**Q14** Consider your single-level page table from the previous architecture. How would you leverage the segment registers to reduce physical storage required for this single-level page table?

**Q15** In this architecture, the OS is subject to virtual address translation. How would you lay out the OS's process space, ensuring that all valid virtual addresses are accessible?

**A Simple TLB with OS Virtual Address Translation**

In this architecture, we remove the 9 segment registers, but retain the requirement that both OS and process memory is subject to virtual address translation.

**Q16** Consider a process which has allocated 8 pages: 5 contiguous pages in the “low” virtual address range, and 3 contiguous pages in the “high” virtual address range. In our new architecture, how much physical memory would a single-level page table require for this process?

a. less than 10 pages
b. 10-100 pages
c. 100-1000 pages
d. more than 1000 pages

**Q17** Describe the process of handling a TLB miss in this new architecture, remembering to list the virtual translations required in both the OS and the process code.

**Q18** In this architecture, are all valid memory dereferences (those which have been mapped, and have entries in the page table) guaranteed to eventually succeed? Why or why not? If not, how can the OS ensure that this guarantee holds?