CS111 Midterm 1 Review

Overview and some practice problems

Isaac Scheinfeld

Midterm Topics 1/2

Threading and Concurrency

- Threads, processes, and dispatching
  - Threads vs. processes
- Concurrency
  - Atomicity, races
- Locks and condition variables
  - Deadlock
- Scheduling
  - Pros/cons of approaches, optimal/pessimal situations

Midterm Covers 2/2

Memory Management

- Linkers and dynamic linking
  - What happens during compilation, linking, and runtime
- Dynamic storage management
  - Stack vs. heap, allocation and reclamation strategies
- Virtual memory
  - Pros/cons of approaches, x86-64 address translation

Review Problems

- Synchronization
  - Race results (practice exam)
  - Bridge crossing (practice exam)
  - Read-write lock (CS110 section by Jerry Cain)
- Virtual memory
  - Computing an alternate layout (practice exam)
- Linking
  - Linking a shared library (practice exam)

Race Results (Practice Exam)

Suppose two threads execute the following C code concurrently.

Initialization

```
int a = 4;
int b = 0;
int c = 0;
```

Thread 1

```
if (a<0)
  c = b - a;
else
  c = b + a;
```

Thread 2

```
b = 10;
a = -3;
```

What are the possible values for c after both threads complete?

You can assume reads and writes are atomic, and that order of execution is preserved by the C compiler and the hardware.

Bridge (Practice Exam)

Automate traffic flow on a one-lane bridge

- Traffic can flow in only a single direction on the bridge at a time.
- Any number of cars can be on the bridge at the same time, as long as they are all traveling in the same direction.
- “Five car rule”: To avoid starvation, once 5+ consecutive northbound cars have entered the bridge, if there are any southbound cars waiting then no more northbound cars may enter the bridge until some southbound cars have crossed. The rule also applies with flipped directions.

Read-Write Lock (CS110 Lab)

Shared reading, exclusive writing

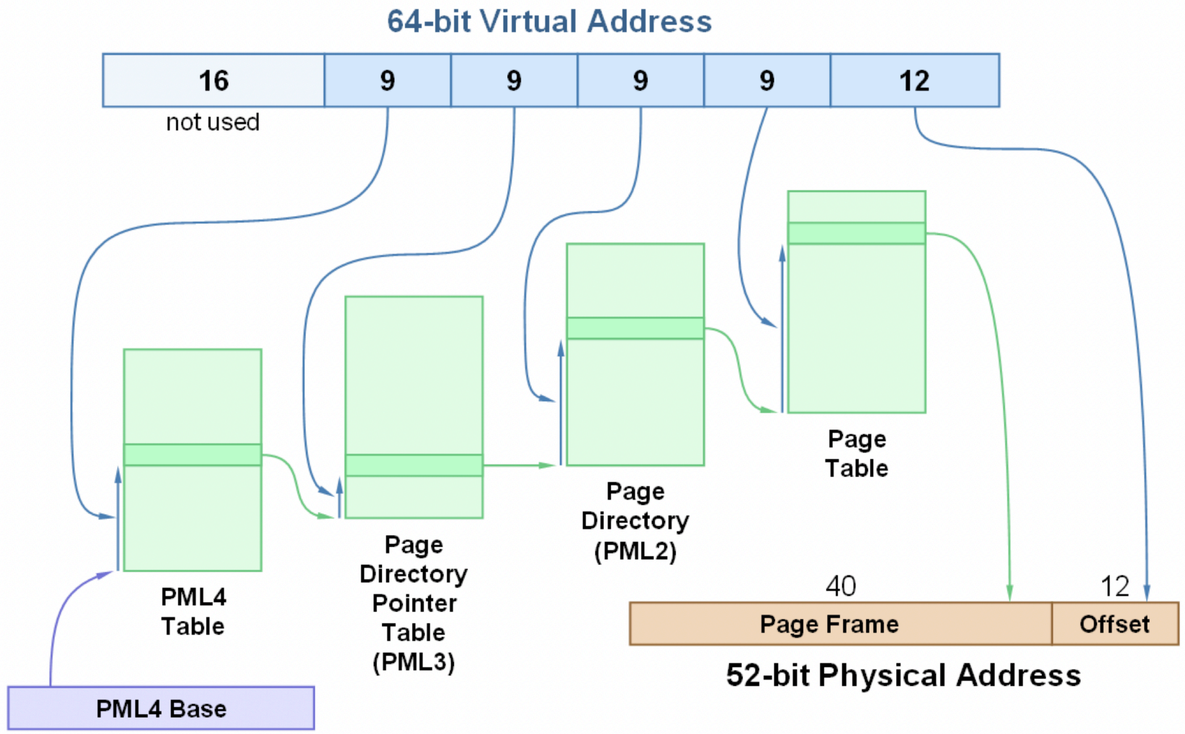
```
class rwlock {
  public:
    rwlock();
    void acquireAsReader();
    void acquireAsWriter();
    void release();
};
```

An Alternate Layout (Practice Exam)

Modifying the x86-64 page table

Starting with a 48-bit virtual addresses, x86-64 uses 4 levels of page table to translate the address to a 52-bit physical address. Each page table entry is 8 bytes long.

If the page size is increased, the number of levels of page table can be reduced. How large must pages be in order to translate 48-bit virtual addresses with only 2 levels of page table?



An Alternate Layout (Practice Exam)

Modifying the x86-64 page table

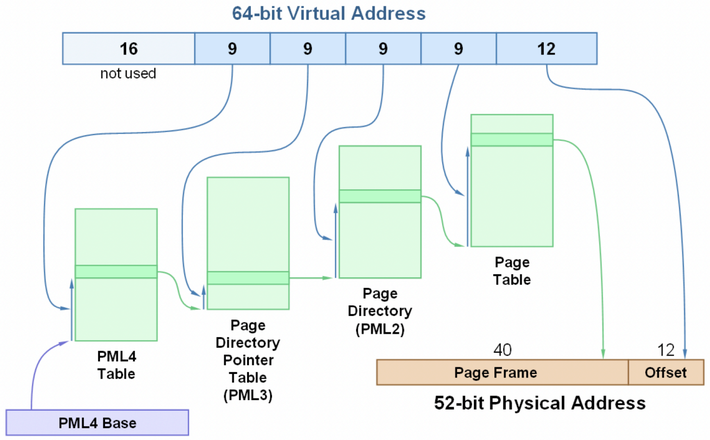
Starting with a 48-bit virtual addresses, x86-64 uses 4 levels of page table to translate the address to a 52-bit physical address. Each page table entry is 8 bytes long.

If the page size is increased, the number of levels of page table can be reduced. How large must pages be in order to translate 48-bit virtual addresses with only 2 levels of page table?

$48 = 2 * |index| + |offset|$

$page\ size = 2\ ^\ |offset|$

$page\ size = 2\ ^\ |index| * entry\ size$



An Alternate Layout (Practice Exam)

Modifying the x86-64 page table

Starting with a 48-bit virtual addresses, x86-64 uses 4 levels of page table to translate the address to a 52-bit physical address. Each page table entry is 8 bytes long.

If the page size is increased, the number of levels of page table can be reduced. How large must pages be in order to translate 48-bit virtual addresses with only 2 levels of page table?

48 = 2 * |index| + |offset|

page size = 2 ^ |offset|

page size = 2 ^ |index| * entry size

2 ^ |offset| = 2 ^ |index| * 2 ^ 3

|offset| = |index| + 3

|offset| = 18 => Page is 2^18 bytes

Linking Shared Libraries

Review Questions

For each of the actions indicate whether the action is carried out by the compiler/assembler, linker, operating system loader (the part of the OS that loads a process into memory and starts its first thread running), or a runtime library that is statically linked into the application. If multiple components share responsibility for the action, explain what each one does.

- Choose the memory address of the jump table.
- Fill in the jump instructions in the jump table.
- Choose the memory address for the shared library code.
- Fill in the instructions that jump into the jump table.