ECE 353: Systems Software
Lecture 24

Second Review

1.0.1

Jon Eyolfson
March 9, 2023

Page Tables Translate Virtual to Physical Addresses

The MMU is the hardware that uses page tables, which may:

- Be a single large table (wasteful, even for 32-bit machines)
- Use the kernel allocated pages from a free list
- Be a multi-level to save space for sparse allocations
- Use a TLB to speed up memory accesses

Threads Enable Concurrency

We explored threads, and related them to something we already know (processes)

- Threads are lighter weight, and share memory by default
- Each process can have multiple threads (but just one at the start)

Both Processes and (Kernel) Threads Enable Parallelization

- Each process can have multiple (kernel) threads
- Most implementations use one-to-one user-to-kernel thread mapping
- The operating system has to manage what happens during a fork, or signals
- We now have synchronization issues

We Want Critical Sections to Protect Against Data Races

We should know what data races are, and how to prevent them:

- Mutex or spinlocks are the most straightforward locks
- We need hardware support to implement locks
- We need some kernel support for wake up notifications
- If we know we have a lot of readers, we should use a read-write lock

We Used Semaphores to Ensure Proper Order

Previously we ensured mutual exclusion, now we can ensure order

- Semaphores contain an initial value you choose
- You can increment the value using post
- You can decrement the value using wait (it blocks if the current value is 0)
- You still need to be prevent data races

We Explored More Advanced Locking

We have another tool to ensure order

- Condition variables are clearer for complex condition signaling
- Locking granularity matters
- You must prevent deadlocks

A Forking Question

Consider the following code:

```
int main() {
  pid_t first = fork();
  pid_t second = fork();
  pid_t third = fork();
  printf("first=%d second=%d third=%d\n", first, second, third);
}
```

What is one reasonable set of outputs (assume the initial process is pid 2)?

Are the outputs in any specific order?

What do the relationships between processes look like?

ucontext Question

Global variables:

```
int i = 0;
ucontext_t uA; // Initialized to execute thread_a()
ucontext_t uB; // Initialized to execute thread b()
```

One kernel thread calls `set_context(&uA)`, what happens?

```
void thread_a() {
    int d = 0;
    while (i < 3) {
        i++;
        printf("A: %d\n", i);
        d = 0;
        getcontext(&uA);
        if (d == 0) {
            d = 1;
            setcontext(&uB);
        }
    }
}
```

```
void thread_b() {
    int d = 1;
    while (i < 3) {
        i++;
        printf("B: %d\n", i);
        d = 1;
        getcontext(&uB);
        if (d == 1) {
            d = 0;
            setcontext(&uA);
        }
    }
}
```