Concurrency

Chapter 26

Previously in CS212…

- We've talked about multi-programming
  - You used fork to create multiple processes to split independent work among different processes
  - Can improve overall performance versus running all tasks in one process

- Now we discuss a new method to allow our programs to share their workload among the available resources

Multi-threading

- We can add new points of execution called **threads** to a single process
- Like a process, but instead of getting its own address space is **it shares the same address space of the process**
  - This means they have access to the same data
- Each thread still has its own set of private registers
  - Unique program counters to track what code statements are being executed
- Still requires a context switch
  - State is saved to thread control blocks (TCBs)
  - Since the address space is shared, we **DO NOT NEED TO SWITCH PAGE TABLES**

A Multi-threaded Address Space

- On the left is a single threaded application
- On the right we see that with multiple threads, we have multiple stacks
- Data within each stack is referred to as **thread-local storage**
- Complicates the address space, but generally stacks are small
  - Recursion can be an exception depending on the depth

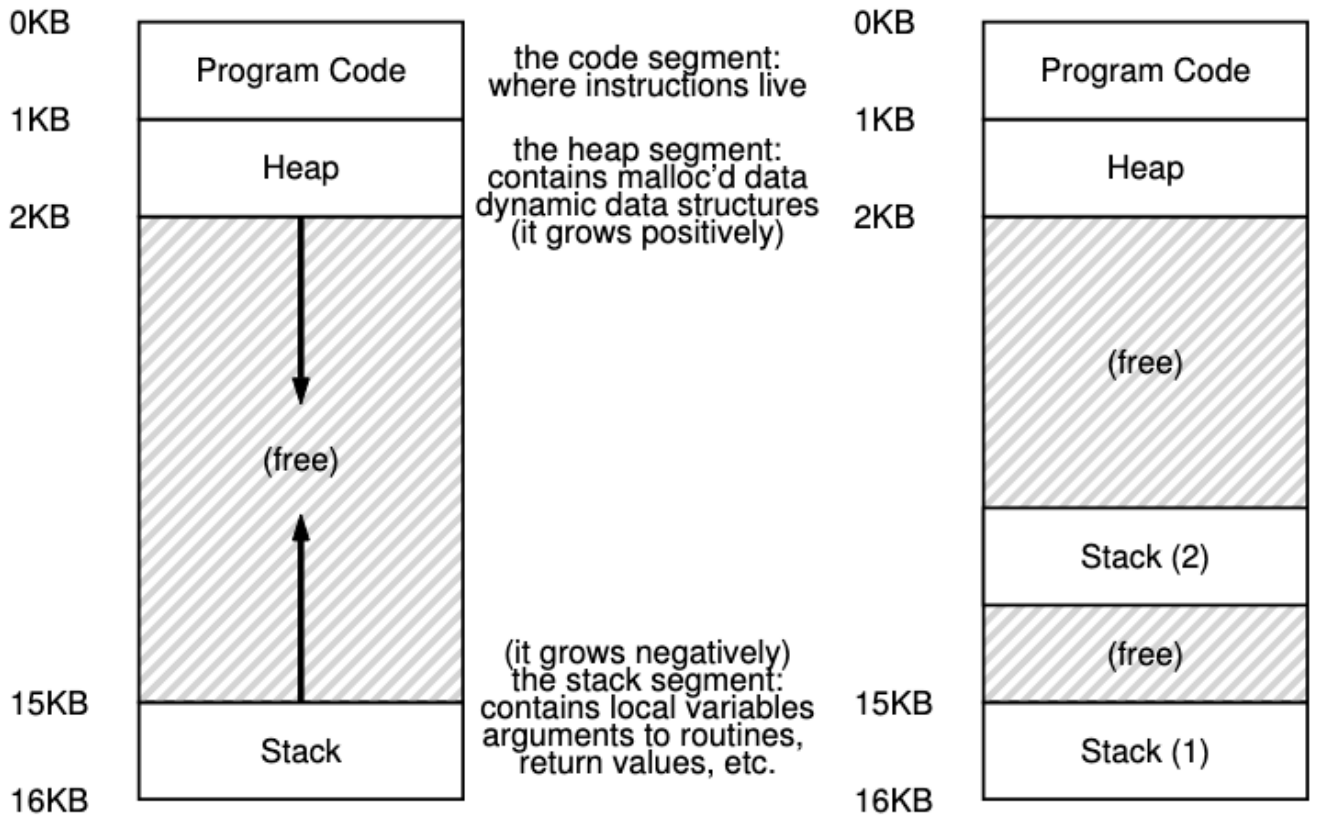


Figure 26.1: **Single-Threaded And Multi-Threaded Address Spaces**

Why Use Threads?

- Parallelism
  - Assume a workload like adding the values in two very large arrays together
  - You could have one thread go through the arrays one position at a time and do the addition
  - Instead, we could use two threads (one per CPU) working simultaneously on different halves of the list
  - Should get the job done faster (usually not precisely x2 the speed, but better than one thread)
- Blocked Process
  - If you have a single thread application that needs to do I/O, this application will be blocked and no longer run on the CPU until I/O is done
  - If your process has more than one thread and other work to do, the process will not be completely blocked as the other thread can continue to run for its time slice

Code Demo!

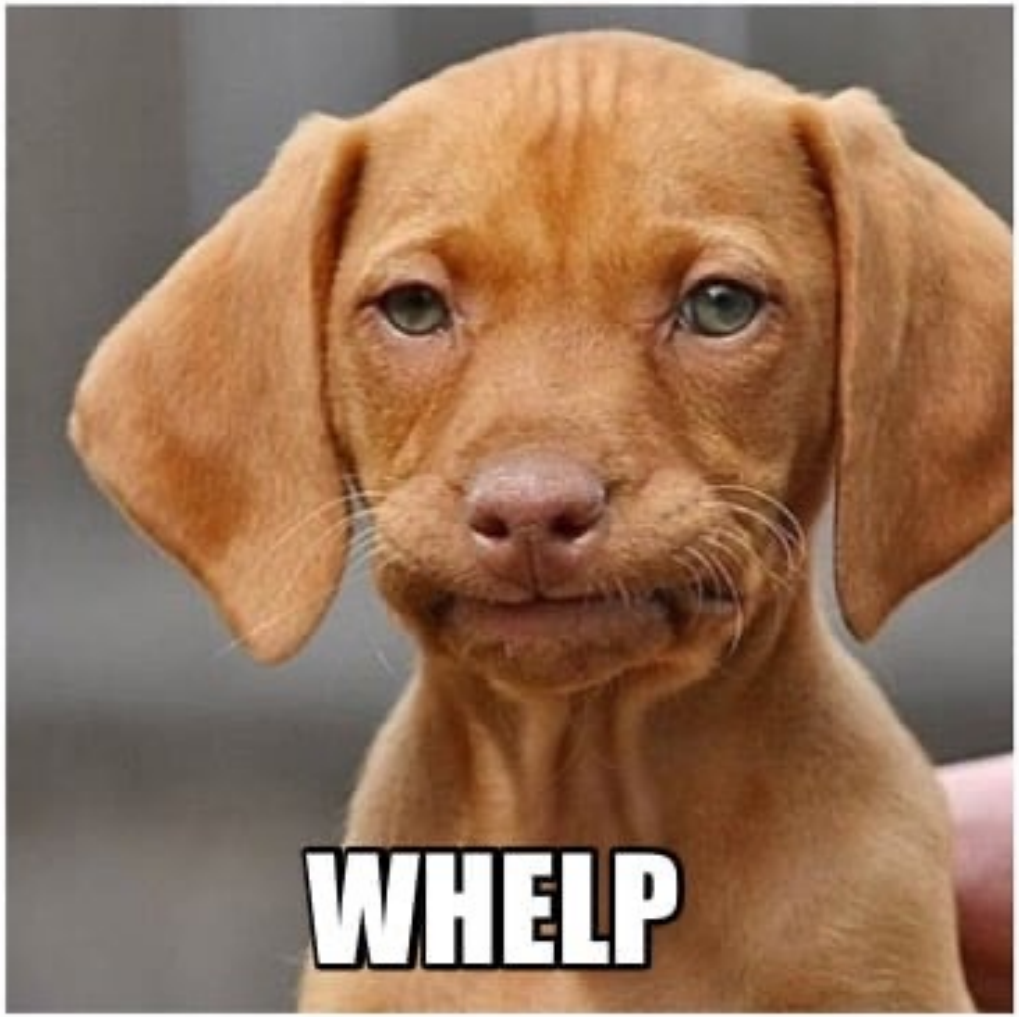
Complications

- Execution order should be considered non-deterministic
  - Just because you've created one process before another **do not assume** they run in that order
- Shared Data
  - Now that thread share an address space, they can potentially manipulate the same data
  - This can lead to strange and non-deterministic results depending on a variety of factors
- Why does this happen?

Uncontrolled Scheduling

- The OS and its scheduler make decisions in a somewhat opaque fashion, and we have little control outside of our program
- If our program has two threads and for some reason one gets interrupted by the OS, it's possible that it is stopped in the middle of an operation
  - Remember that most instructions require multiple low-level steps (assembly)
- This can put our program in a compromised state when it returns to executing if another thread jumped in and changed things

Cool, sounds like more work?



Kind of yea..

- When we are multi-threading, we need to consider when two threads may interact with the same data

- Without any coordination, your program suffers from a **race condition**
  - Which thread gets to the data first or in what state a thread may be stopped before it completes an operation can change the outcome

- Code in which threads share data is called a **critical section**
  - Ideally, when one thread working on that shared data the other can't (called **mutual exclusion**)

Thread Synchronization

- We have a need for **atomic operations** that cannot be interrupted
  - They either finish completely, or they simply aren't done

- Hardware can provide a few such atomic operations called **synchronization primitives** that we can use to control access (more on this later)

- More complex interactions like one thread needing to finish work before another can start, requires **condition variables** put a thread to sleep and wake it up when it's time to work

Next Time

- We take a closer look at some code to make threaded applications possible