CSE 451: Operating Systems Winter 2009

Module 4 Processes

Mark Zbikowski
Gary Kimura

Process management

- This module begins a series of topics on processes, threads, and synchronization
  - this is the most important part of the class
- Today: processes and process management
  - what are the OS units of ownership / execution?
  - how are they represented inside the OS?
  - how is the CPU scheduled across processes?
  - what are the possible execution states of a process?
    - and how does the system move between them?

The process

- The process is the OS’s abstraction for execution
  - the unit of ownership
  - the unit of execution (sorta)
  - the unit of scheduling (kinda)
  - the dynamic (active) execution context
    - compared with program: static, just a bunch of bytes
- Process is often called a job, task, or sequential process
  - a sequential process is a program in execution
    - defines the instruction-at-a-time execution of a program

What’s in a process?

- A process consists of (at least):
  - an address space
  - the code for the running program
  - the data for the running program
  - at least one thread
    - Registers, IP
    - Floating point state
    - Stack and stack pointer
  - a set of OS resources
    - open files, network connections, sound channels, …
- In other words, it’s all the stuff you need to run the program
  - or to re-start it, if it’s interrupted at some point

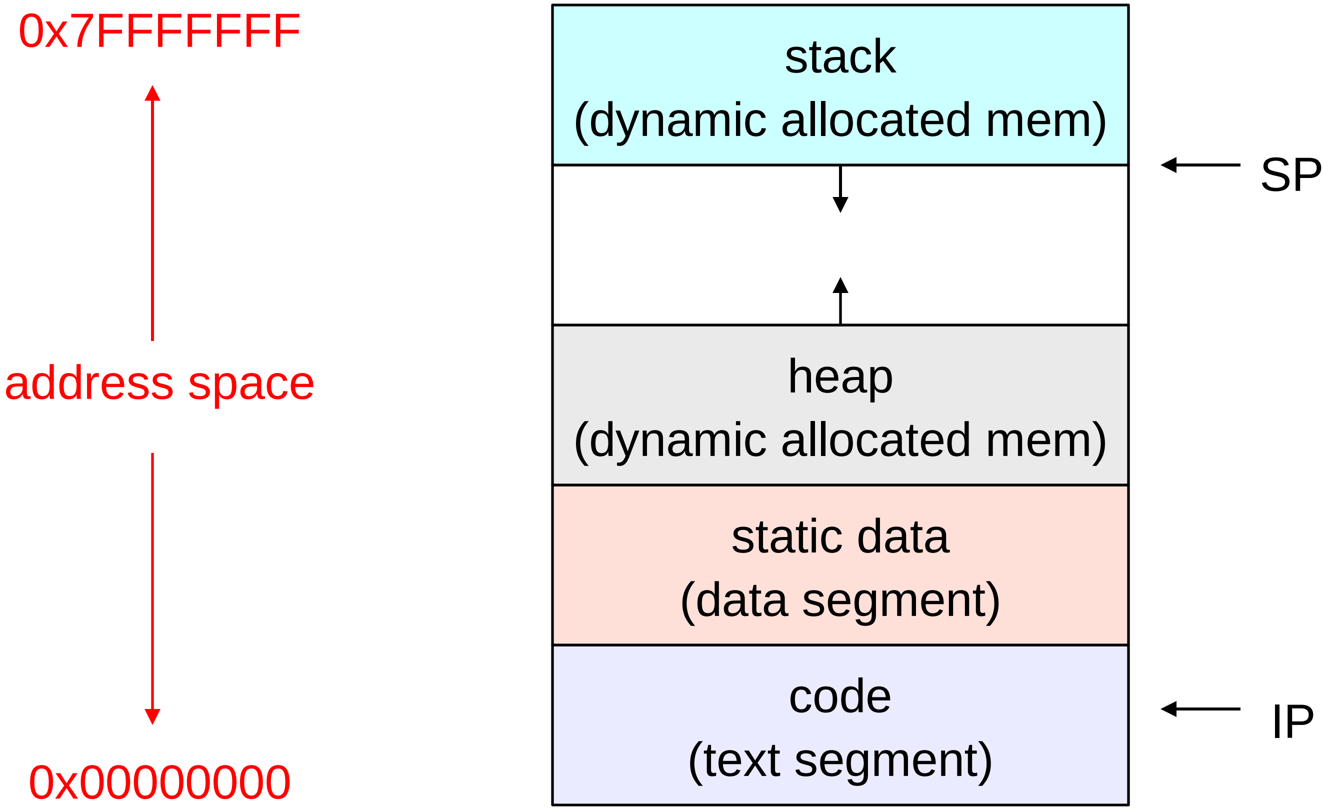
The Process Object

- There's a data structure called the process object (_KPROCESS in base\ntos\inc\ke.h) that holds all this stuff
  - Processes are identified from user space by a process ID, returned by NtCreateProcess.
- OS keeps all of a process's hardware execution state in the _KTHREAD (same file) when the process isn't running
  - IP, SP, registers, etc.
  - when a process is unscheduled, the state is transferred out of the hardware into the _KTHREAD
- Note: It's natural to think that there must be some esoteric techniques being used
  - fancy data structures that'd you'd never think of yourself

Wrong! It's pretty much just what you'd think of!

Except for some clever assembly code…

A process's address space (very simplified)



Process creation

- New processes are created by existing processes
  - creator is called the parent
  - created process is called the child
  - what creates the first process, and when?
- In some systems, parent defines or donates resources and privileges for its children
  - LINUX/UNIX: child inherits parent's security context, environment, open file list, etc.
  - NT: all the above are *optional (remember, mechanism vs policy)*, the Windows subsystem provides policy.
- When child is created, parent may either wait for it to finish, or may continue in parallel, or both!

Process Creation 2

- In LINUX, fork/exec pairs.
  - fork() *clones* the current process, duplicates all memory, “inherit” open files
  - exec() throws away all memory and loads new program into memory. Keeps all open files!
  - Very useful, but… wasteful. >99% of all fork() calls followed by exec(). Copy-on-write memory helps but still a big overhead.
- Windows has parent process doing the work
  - Create process
  - Fill in memory
  - Pass handles
  - Create thread with stack and IP
  - Many system calls (compared with LINUX) but all policy is in user code. More flexible.

Process Destruction

- Privileged operation!
  - Process can always kill itself
  - Killing another process requires permission
- Terminates all threads (next lecture)
- Releases owned resources to known state
  - Files
  - Events
  - Memory
- Notification sent to interested parties
- KPROCESS is freed