Processes
CS 111
Operating System Principles
Peter Reiher

Outline

- Processes and threads
- Going from conceptual to real systems
- How does the OS handle processes and threads?
- Creating and destroying processes

Processes and Threads

- Threads are a simple concept
- They are used in real operating systems
- But they aren't the actual key interpreter abstraction of real operating systems
- Systems like Linux and Windows use another abstraction
  - The *process*

What Is a Process?

- Essentially, a virtual machine for running a program
- So it contains state
- And resources required to do its work
  - Like threads, virtual memory, communications primitives
- Most machines run multiple processes
  - Serially and simultaneously

Processes and Programs

- A program is a static representation of work to be done
- A process is the dynamic, running instantiation of a program
- Most programs are run many different times
  - On the same or different machines
- Each individual run is represented by a unique process
  - Which has a discrete start and (usually) end

How Does a Process Differ From a Thread?

- Processes are a higher level abstraction
- They can contain multiple threads
  - Implying that there can be simultaneous actions within one program
  - Which is not possible in a thread
- They typically encapsulate an entire running program
- They are heavier weight

The OS and Processes

- The OS must multiplex virtual processes onto physical processors
  - Start and end processes
  - Set them up to run properly
  - Isolate them from other processes
  - Ensure that all processes get a chance to do their work
  - Share the physical resources properly
- One important aspect of this task is properly handling process state

Process State

- Similar to thread state
- Need information on:
  - What instruction to run next
  - Where the process' memory is located
  - What are the contents of important registers
  - What other resources (physical or virtual) are available to the process
  - Perhaps security-related information (like owner)
- Major components are register state (e.g., the PC) and memory state

Process State and Memory

- Processes have several different types of memory segments
  - The memory holding their code
  - The memory holding their stack
  - The memory holding their data
- Each is somewhat different in its purpose and use

Process Code Memory

- The instructions to be executed to run the process
- Typically static
  - Loaded when the process starts
  - Then they never change
- Of known, fixed size
- Often, a lot of the program code will never be executed by a given process running it

Implications for the OS

- Obviously, memory object holding the code must allow execution
  - Need not be writeable
    - Self-modifying code is a bad idea, usually
  - Should it be readable?
- Can use a fixed size domain
  - Which can be determined before the process executes
- Possibility of loading the code on demand

Process Stack Memory

- Memory holding the run-time state of the process
- Modern languages and operating systems are stack oriented
  - Routines call other routines
  - Expecting to regain control when the called routine exits
  - Arbitrarily deep layers of calling
- The stack encodes that

Stack Frames

- Each routine that is called keeps its relevant data in a stack frame
  - Its own piece of state
- Stack frames contain:
  - Storage for procedure local (as opposed to global) variables
  - Storage for invocation parameters
  - Space to save and restore registers
    - Popped off stack when call returns

Characteristics of Stack Memory

- Of unknown and changing size
  - Grows when functions are called
  - Shrinks when they return
- Contents created dynamically
  - Not the same from run to run
  - Often data-dependent
- Not inherently executable
  - Contains pointers to code, not code itself
- A compact encoding of the dynamic state of the process

Implications for the OS

- The memory domain for the stack must be readable and writeable
  - But need not be executable
- OS must worry about stack overrunning the memory area it's in
  - What to do if it does?
    - Extend the domain?
    - Kill the process?

Process Data Memory

- All the data the process is operating on
- Of highly varying size
  - During a process run
  - From run to run of a process
- Read/write access required
  - Usually not execute access
  - Few modern systems allow processes to create new code

Implications for the OS

- Must be prepared to give processes new domains for dynamic data
  - Since you can't generally predict ahead of time how much memory a process will need
  - Need strategy if process asks for more memory than you can give it
- Should give read/write permission to these domains
  - Usually not execute

Layout of Process in Memory

- In Unix systems, data segment grows up
- Stack segment grows down
- They aren't allowed to meet

Loading Programs Into Processes

- The program represents a piece of code that could be executed
- The process is the actual dynamic executing version of the program
- To get from the code to the running version, you need to perform the *loading* step
  - Initializing the various memory domains we just mentioned

Loading Programs

- The load module
  - All external references have been resolved
  - All modules combined into a few segments
  - Includes multiple segments (code, data, symbol table)
- A computer cannot "execute" a load module
  - Computers execute instructions in memory
  - Memory must be allocated for each segment
  - Code must be copied from load module to memory