CSE 153
Design of Operating Systems
Winter 2019
Lecture 4: Processes
If you have not already, please make sure to enroll in Piazza
  ~20 students have not yet

Lab 1: please check the hints part of the lab (bottom section) -- I will may post additional items
Last class

- OS structure, operation, and interaction with user progs
  - **Privileged mode**: To enforce isolation and manage resources, OS must have exclusive powers not available to users
    - How does the switch happen securely?

- OS is not running unless there is an event:
  - OS schedules a user process to run then goes to sleep
  - It wakes up (who wakes it?) to handle events
  - Many types of events

- **Program view and system calls**: program asks the OS when it needs a privileged operation
Today, we start discussing the first abstraction that enables us to virtualize (i.e., share) the CPU – processes!
What is virtualization?

- What is a virtual something?
  - Somehow not real? But still functional?

- Provide illusion for each program of own copy of resources
  - Lets say the CPU or memory; every program thinks it has its own
  - In reality, limited physical resources (e.g., 1 CPU)
    » It must be shared! (in time, or space)

- Frees up programs from worrying about sharing
  - The OS implements sharing, creating illusion of exclusive resources
    ➔ Virtualization!

- Virtual resource provided as an object with defined operations on it. ➔ Abstraction
Virtualizing the CPU -- Processes

- This lecture starts a class segment that covers processes, threads, and synchronization
  - Basis for Midterm and Labs 1 and 2

- Today’s topics are processes and process management
  - How do we virtualize the CPU?
    - Virtualization: give each program the illusion of its own CPU
    - What is the magic? We only have one real CPU
  - How are applications represented in the OS?
  - How is work scheduled in the CPU?
The Process

- The process is the OS abstraction for execution
  - It is the unit of execution
  - It is the unit of scheduling

- A process is a program in execution
  - Programs are static entities with the potential for execution
  - Process is the animated/active program
    » Starts from the program, but also includes dynamic state
    » As the representative of the program, it is the “owner” of other resources (memory, files, sockets, …)

- How does the OS implement this abstraction?
  - How does it share the CPU?
How to support this abstraction?

- First, we’ll look at what state a process encapsulates
  - State of the virtual processor we are giving to each program

- Next we will talk about process behavior/CPU time sharing
  - How to implement the process illusion

- Next, we discuss how the OS implements this abstraction
  - What data structures it keeps, and the role of the scheduler

- Finally, we see the process interface offered to programs
  - How to use this abstraction
  - Next class
A process contains all the state for a program in execution
- An address space containing
  - Static memory:
    - The code and input data for the executing program
  - Dynamic memory:
    - The memory allocated by the executing program
    - An execution stack encapsulating the state of procedure calls
- Control registers such as the program counter (PC)
- A set of general-purpose registers with current values
- A set of operating system resources
  - Open files, network connections, etc.

A process is named using its process ID (PID)
Address Space (memory abstraction)

- Stack
  - Stack Pointer (SP)
- Heap (Dynamic Memory Alloc)
- Static Data (Data Segment)
- Code (Text Segment)

Address Space:
- 0x00000000
- 0xFFFFFFFF

PC

Dynamic

Static
How to support this abstraction?

- First, we’ll look at what state a process encapsulates
  - State of the virtual processor we are giving to each program

- Next we will talk about process behavior/CPU time sharing
  - How to implement the process illusion

- Next, we discuss how the OS implements this abstraction
  - What data structures it keeps, and the role of the scheduler

- Finally, we see the process interface offered to programs
  - How to use this abstraction
  - Next class
### Process Execution State

- A process is born, executes for a while, and then dies

- The process **execution state** that indicates what it is currently doing
  - **Running**: Executing instructions on the CPU
    - It is the process that has control of the CPU
    - How many processes can be in the running state simultaneously?
  - **Ready**: Waiting to be assigned to the CPU
    - Ready to execute, but another process is executing on the CPU
  - **Waiting**: Waiting for an event, e.g., I/O completion
    - It cannot make progress until event is signaled (disk completes)
Execution state (cont’d)

- As a process executes, it moves from state to state
  - Unix “ps -x”: STAT column indicates execution state
  - What state do you think a process is in most of the time?
  - How many processes can a system support?

<table>
<thead>
<tr>
<th>PROCESS STATE CODES</th>
</tr>
</thead>
</table>
| Here are the different values that the s, stat and state output specifiers (header "ST") display:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>uninterruptible sleep (usually IO)</td>
</tr>
<tr>
<td>R</td>
<td>running or runnable (on run queue)</td>
</tr>
<tr>
<td>S</td>
<td>interruptible sleep (waiting for an event to complete)</td>
</tr>
<tr>
<td>T</td>
<td>stopped, either by a job control signal or because it is being traced.</td>
</tr>
<tr>
<td>W</td>
<td>paging (not valid since the 2.6.xx kernel)</td>
</tr>
<tr>
<td>X</td>
<td>dead (should never be seen)</td>
</tr>
<tr>
<td>Z</td>
<td>defunct (“zombie”) process, terminated but not reaped by its parent.</td>
</tr>
</tbody>
</table>

For BSD formats and when the stat keyword is used, additional characters may be displayed:

- < high-priority (not nice to other users)
- N low-priority (nice to other users)
- L has pages locked into memory (for real-time and custom IO)
- s is a session leader
- t is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)
- + is in the foreground process group.
Execution State Graph

New → Ready
Create Process

Ready → Waiting
I/O Done

Waiting → Running
Schedule Process

Running → Waiting
Unschedule Process

Waiting → Terminated
I/O, Page Fault, etc.

Terminated → New
Process Exit