Goals for Today

- **Learning Objectives**
  - Understand basic concept of process (control info, creation, termination, states, etc.)
  - Review some important UNIX syscalls and concepts for sys. programming

- **Announcements**
  - Grading scheme updated in D2L
    - *still subject to change at any time…*
  - 8 people missing submission for HW1…
    - *6? 2 unnamed papers turned in…*
  - Coming Soon…
    - Rough schedule through the 1st exam to be published soon.
    - 1st programming assignment
    - Details about project
Today in Context

• Focus on **Processes** and how…
  • they work
  • they are represented
  • they are managed
  • etc.
The Process

• The notion of a **Process**…

  • many definitions…e.g., *an instance of a program running on a computer*;

  • consists of two (three) critical things:
    • (1) an executable program **(code)**,
    • (2) associated **data**,
    • (3) **execution context** (info the OS needs to manage the process)

• is realized as nothing more than a data structure!

Why Processes?

• **Q:** Why is the notion of a process useful?
Why Processes?

- **Q:** Why is the notion of a process useful?
  - A standard approach for defining executable code with associated data
  - An OS abstraction for executing a program with limited privileges
  - ...
The Process Control Block (PCB) & Getting Process Info

• A “snapshot” that contains all necessary data to restart a process where it left off

• While the program is executing, this process can be uniquely characterized by a number of elements

• In practice, how to get info about a process?
  • getpid() //get the proc. ID
  • getppid() //get the parent’s proc. ID
  • getpriority() //get the proc. priority
  • etc...

---

<table>
<thead>
<tr>
<th>Identifier</th>
<th>State</th>
<th>Priority</th>
<th>Program counter</th>
<th>Memory pointers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Process States (Simple: 2-States)

- Simple model:
  - proc. is either **running** or **not running**
  - running = on the processor
  - not running = not on the processor
- **Q:** In a single processor machine, how many processors can be running? 1!
- **Q:** What do we do with the other processes? Hold in a queue; later, interrupt running proc and dispatch next proc.
Process States

- Not all processes are ready to execute…
  **Q:** What problems arise with using a single queue?
Process States (5 States / Multiple ‘Blocked’ Queues)

- Not all processes are ready to execute…
  **Q:** What problems arise with using a single queue?

- Additional States:
  - **New**—the process is being created
  - **Ready**—the process is ready to run; waiting to be assigned to a processor
  - **Running**—the instructions of the process are being executed
  - **Blocked**—the process is waiting for an event to occur (e.g., I/O)
  - **Exit**—the process has finished execution

---

---

---

---
Creating a Process

• **Q:** How do UNIX processes get created?
• Some discussion in the text—processes created when…
  • …system boots,
  • …user opens an app,
  • …an existing process spawns a child process
  • etc.
Creating a Process

**DEMO**

take a look at a process tree (already-created processes)

```
ps axjf
```

Some things to note:
- **names** (e.g., init),
- **relationships** (parents, children, grandchildren),
- **IDs** (PID, PPID)
Photos of whiteboard after class
PCB
- ID
- Priority
- I/O info
- Memory
- Printer
- # Que

*Code*
- Code 1
- Code 2

*Data*
- Data

*Context*
- Context

**The Process**
- CPU
- MM

**I/O**
- I/O mouse
- I/O printer
- I/O network

**Dispatcher**
- t/e
- Running
- Run for
- I/O

**Running**
- Waiting for
- Event

**Not Running**
- Enter

**Code**
- P1
- P2
- P3

**IO Q**