Processes

CSE 410, Spring 2007
Computer Systems

http://www.cs.washington.edu/410/

---

**Example OS in operation**

- **Application Interface (API)**
  - Firefox
  - Photoshop
  - Acrobat
  - iTunes

- **Hardware Abstraction Layer**
  - File Systems
  - Memory Manager
  - Process Manager
  - Network Support

- **Device Drivers**
- **Interrupt Handlers**
- **Boot & Init**

---

**Reading and References**

- **Reading**
  - Chapter 4 through 4.5.4, *Operating System Concepts*, Silberschatz, Galvin, and Gagne

- **Other References**

---

**Programs and Processes**

- A **program** is passive
  - a file on disk with code that can be run
- A **process** is active
  - an instance of a program in execution
  - also called job, task, sequential process
- There are always many processes running
- Some may be running the same program
  - but they are still separate and independent processes
What are the parts of a process?

- code for the running program
- data for the running program
  - heap, stack
- location of the next instruction (PC)
- current state of the general-purpose registers
- list of open resources
  - files, network connections
- lots of OS management data

Process State

- Each process has an execution state that indicates what it is currently doing:
  - ready: waiting to be assigned to the CPU
  - running: executing instructions on the CPU
  - waiting: waiting for an event, e.g., I/O completion, so that it can be made ready
- As a program executes, the OS moves the process from state to state

Process State Changing

Processes move from state to state as a result of actions they perform (e.g., system calls), OS actions (rescheduling) and external actions (interrupts)

Process Data Structures

- At any time, there are many processes active in a system
- The OS has data structures representing each process
  - primary structure is the Process Control Block (PCB)
- PCB contains info about a process
  - including pointers to other related data blocks
PCBs and Hardware State

- When a process runs, its PC, SP, and registers, are loaded on the CPU
- When the OS switches to a new process, it
  » saves the current process’s register values to its PCB
  » loads the next process’s register values from its PCB
- This is called a context switch. It occurs 100-1000 times per second
  » why so often?
  » why not more often?

Context switch is pure overhead

- Switching processes can be expensive
  » register reload
  » OS data structures
- Lightweight context reduces cost of switch
  » threads
- Special hardware reduces cost of switch
  » larger register files with register windows

Simple Process Control Block

- process state
- process number
- program counter
- stack pointer
- 32 general-purpose registers
- memory management info
- username of owner
- queue pointers for state queues
- scheduling info (priority, etc.)
- accounting info

Simplified W2K Process Data

- Process environment block
- System address space
- Thread environment block
- Process block
- Win32 process block
- Handle table
- Thread block

Copied from Inside Windows 2000
Process State Queues

Ready Queue Header
- PCB
- PCB
- PCB

Head ptr
Tail ptr

Wait Queue Header
- PCB
- PCB

Head ptr
Tail ptr

Many wait queues—
one for disk, one for
user input, etc.

PCBs and State Queues

- PCBs are data structures in OS memory
- A PCB is created for a process when it starts
  and put on the ready queue
- While the process is active, PCB is on one of
  the state queues
- When the process is terminated, its PCB is
deallocated (after a little while)

Getting control back

- How does the OS get control back from a
  running process?
  » The process could explicitly return control to
    the OS (in many real-time systems)
  » Generally, we can’t trust the process to do this
- OS sets a timer on the CPU (privileged
  instruction) and starts a user process
- When the timer expires control passes to OS
  » impact on “hard real-time” system?

Scheduling a process

- Batch processes tend to be scheduled over a
  long period by a job scheduler
  » explicit dollar value on priority
  » longer time in CPU once loaded and started
- Interactive or soft real time processes are
  started as needed and compete for CPU
  » dynamic priorities
  » rapid context switching of many processes
Creating a process

- The OS creates processes upon request
- The first few processes are all part of the operating system itself
  - services, sessions, spoolers, network tools, ...
- Further processes created as response to login, user command, scheduled events
  - winlogin, sshd, navigator, photoshop, ...

**create-process**

- OS provides create-process system call
  - parent process creates one or more children
  - each child can create more children
  - the result is a process tree
- Parent can wait or continue immediately
  - create a process and block (synchronous)
  - create a process and continue (asynchronous)

```plaintext
C:\home\finson>pslist -t
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Pid</th>
<th>Pri</th>
</tr>
</thead>
<tbody>
<tr>
<td>explorer</td>
<td>1376</td>
<td>8</td>
</tr>
<tr>
<td>firefox</td>
<td>528</td>
<td>8</td>
</tr>
<tr>
<td>WINZIP32</td>
<td>1964</td>
<td>8</td>
</tr>
<tr>
<td>CMD</td>
<td>1272</td>
<td>8</td>
</tr>
<tr>
<td>pslist</td>
<td>1956</td>
<td>13</td>
</tr>
<tr>
<td>ADUserMon</td>
<td>1340</td>
<td>8</td>
</tr>
<tr>
<td>powerpnt</td>
<td>1684</td>
<td>8</td>
</tr>
<tr>
<td>jusched</td>
<td>1648</td>
<td>8</td>
</tr>
<tr>
<td>rundll32</td>
<td>1608</td>
<td>8</td>
</tr>
<tr>
<td>mmtray</td>
<td>1628</td>
<td>8</td>
</tr>
<tr>
<td>powerpnt</td>
<td>1684</td>
<td>8</td>
</tr>
<tr>
<td>pageant</td>
<td>1576</td>
<td>8</td>
</tr>
<tr>
<td>updaterui</td>
<td>1592</td>
<td>8</td>
</tr>
<tr>
<td>rundll32</td>
<td>1608</td>
<td>8</td>
</tr>
<tr>
<td>vstskmgr</td>
<td>584</td>
<td>8</td>
</tr>
<tr>
<td>svchost</td>
<td>456</td>
<td>8</td>
</tr>
<tr>
<td>spoolsv</td>
<td>456</td>
<td>8</td>
</tr>
</tbody>
</table>

W2K CreateProcess function

- Open the program file to be executed
- Create the W2K executive process object
- Create the initial thread (stack, context, ...)
- Notify Win32 subsystem about new process
- Start execution of the initial thread
- Complete initialization (eg, load dlls)
- Continue execution in both processes