Goals of Today’s Lecture

- How multiple programs can run at once
  - Processes
  - Context switching
  - Process control block
  - Virtual memory
- Boundary between parts of the system
  - User programs
  - Operating system
  - Underlying hardware
- Mechanics of handling a page fault
  - Page tables
  - Process ID registers
  - Page faults

Operating System

- Supports virtual machines
  - Promises each process the illusion of having whole machine to itself
- Provides services:
  - Protection
  - Scheduling
  - Memory management
  - File systems
  - Synchronization
  - etc.

What is a Process?

- A process is a running program with its own ...
  - Processor state
    - EIP, EFLAGS, registers
  - Address space (memory)
    - Text, bss, data, heap, stack
- Supporting the abstraction
  - Processor
    - Saving state per process
    - Context switching
  - Main memory
    - Sharing physical memory
    - Supporting virtual memory
  - Efficiency, fairness, protection
Divide Hardware into Little Pieces?

• Idea: registers, memory, ALU, etc. per process
  ○ Pro: totally independent operation of each process
  ○ Con: lots of extra hardware;
    some parts idle at any given time;
    hard limit on the number of processes

Indirection, and Sharing in Time?

• Idea: swap processes in and out of the CPU; map references into physical addresses
  ○ Pro: make effective use of the resources by sharing
  ○ Con: overhead of swapping processes;
    overhead of mapping memory references

When to Change Which Process is Running?

• When a process is stalled waiting for I/O
  ○ Better utilize the CPU, e.g., while waiting for disk access

1: CPU I/O CPU I/O CPU I/O
2: CPU I/O CPU I/O CPU I/O

• When a process has been running for a while
  ○ Sharing on a fine time scale to give each process the illusion of running on its own machine
  ○ Trade-off efficiency for a finer granularity of fairness

Life Cycle of a Process

• Running: instructions are being executed
• Waiting: waiting for some event (e.g., I/O finish)
• Ready: ready to be assigned to a processor
Switching Between Processes

Process Control Block

Sharing Memory

Context Switch: What to Save & Load?

• Process state
  ◦ New, ready, waiting, halted

• CPU registers
  ◦ EIP, EFLAGS, EAX, EBX, ...

• I/O status information
  ◦ Open files, I/O requests, ...

• Memory management information
  ◦ Page tables

• Accounting information
  ◦ Time limits, group ID, ...

• CPU scheduling information
  ◦ Priority, queues

Process Control Block

• For each process, the OS keeps track of ...
  ◦ Process state
  ◦ CPU registers
  ◦ CPU scheduling information
  ◦ Memory management information
  ◦ Accounting information
  ◦ I/O status information

Sharing Memory

• In the old days…
  ◦ MS-DOS (1990)

• Problem: protection
  ◦ What prevents process 1 from reading/writing process 3’s memory?
  ◦ What prevents process 2 from reading/writing OS’s memory?

• In modern days, Virtual Memory protection
  ◦ IBM VM-370 (1970)
  ◦ UNIX (1975)
  ◦ MS Windows (2000)
Virtual Memory

- Give each process illusion of large address space
  - E.g., 32-bit addresses that reference 4 Gig of memory
- Divide the physical memory into fixed-sized pages
  - E.g., 4 Kilobyte pages
- Swap pages between disk and main memory
  - Bring in a page when a process accesses the space
  - May require swapping out a page already in memory
- Keep track of where pages are stored in memory
  - Maintain a page table for each process to do mapping
- Treat address as page number and offset in page
  - High-order bits refer to the page
  - Low-order bits refer to the offset in the page

Virtual Memory for a Process

Address Translation

0 Virtual Address Space 0 Physical Address Space

address

offset in page

virtual page number

physical page number

Page Tables

Process 1 Virtual Address Space

Process 2 Virtual Address Space

Process 1 Virtual Address Space

OS V.A.S.

OS V.A.S.
Page Tables Reside in Memory...

Protection Between Processes

Paging

- User-mode (unprivileged) process cannot modify Process ID register
- If page tables are set up correctly, process #1 can access only its own pages in physical memory
- The operating system sets up the page tables
Page Fault!

Process ID

```
movl 0002104, %eax
```

Physical Address Space

Write Some Other Page to Disk

Process ID

```
movl 0002104, %eax
```

Physical Address Space

Fetch Current Page, Adjust Page Tables

Process ID

```
movl 0002104, %eax
```

Physical Address Space

Measuring the Memory Usage

Unix

```
% ps -l
F  UID   PID  PPID PRI   VSZ   RSS STAT  TIME COMMAND
0  115  7264  7262  17  4716  1400  SN   0:00 -csh
0  115  7290  7264  17 15380 10940  SN   5:52 emacs
0  115  3283  7264  23  2864   812  RN   0:00 ps -l
```

Virtual memory usage
Physical memory usage ("resident set size")
CPU time used by this process so far

Windows

CPU Usage: 79.7%
Virtual Memory Usage: 85.5%
Physical Memory Usage: 23.6%
Context Switch, in More Detail

Fault-handler hardware:
1. Enters privileged mode
2. Sets EIP to specific location in operating system
3. Sets ESP to operating-system stack in OS memory
4. Pushes old (process 1) EIP and ESP on OS stack

OS software:
5. Pops saved EIP,ESP into PCB1
6. Copies rest of registers into PCB1
7. Sends instructions to disk drive to fetch page
**Summary**

- **Abstraction of a “process”**
  - CPU: a share of CPU resources on a small time scale
  - Memory: a complete address space of your own

- **OS support for the process abstraction**
  - CPU: context switch between processes
  - Memory: virtual memory (VM) and page replacement
  - Files: open/read/write, rather than “move disk head”
  - Protection: ensure process access only its own resources

- **Hardware support for the process abstraction**
  - Context switches, and push/pop registers on the stack
  - Switch between privileged and unprivileged modes
  - Map VM address and process ID to physical memory

---

**Resuming Some Other Process**

**OS software**
8. Sets process-ID register to 2
9. Pushes saved EIP,ESP from PCB2 onto OS stack
10. Copies rest of registers from PCB2
11. Executes “return from interrupt” instruction

**Hardware**
12. Pops EIP,ESP into registers
13. Switches back to unprivileged mode
14. Resumes where process 2 left off last time

---

**System call, just another kind of fault**

**Process 1**

- **Running**
  - `mov $4,%eax`
  - `int $0x80`
  - `addl %eax, %ecx`

- **Waiting**

---

**System call (privileged instruction)**