Introduction

What Is an Operating System?

* 
* 
*
What Is an OS? (Party Line)

- **Control Program**
  - control execution of user programs
  - prevent errors/misuse of hardware

- **Resource Allocator**
  - Examples of resources?
  - ...
  - ...
  - ...

- **A Support Environment**
  - useful things should (!!) get done in reasonable ways
  - NB (Nota Bene): there is a VERY fine line between what is and what is not considered part of an OS

---

**Computer System Components**

- **Hardware**
- **OS**
- **Applications**
- **Users**
Hardware: Component Review

• Four Major Hardware Elements
  – Processor
    • controls computer operation & data processing, and maintains representation of processor status
  – Main Memory
    • typically volatile, stores data & programs
  – I/O Modules
    • move data between computer & environment/storage
  – System Interconnection (Buses)
    • hardware & mechanisms to support communication amongst other 3 elements

Hardware: Machine Architecture Review

![Machine Architecture Diagram]
Hardware: Fetch-Execute Cycle

- Fetch Instruction pointed to by PC
  - AR <-- PC
  - DR <-- Memory(AR)
  - IR <-- Opcode(DR)

- Increment PC

- Fetch operands & Execute Instruction
  - AR <-- PC
  - DR <-- Memory(AR)
  - IR <-- Opcode(DR)

Hardware/Storage Concerns for OSes

- Memory (Volatile Storage)
  - basic unit (byte, word?, size?)
  - addressing (32-bit? 64-bit?)
  - management?
  - how many registers/how much memory

- Non-Volatile Storage
  - hard drive
  - removable media (ZIP, JAZ, DVD, CD, magnetic tape)
  - how is I/O to these optimized?
  - how is I/O between these and main memory coordinated efficiently?

- caching
- disk-head scheduling
- DMA

registes

cache

main memory

electronic disk

magnetic disk

optical disk

magnetic tapes
Buses and Other Communication Lines

- Memory & Cache
  - Width? 64 bit?
  - Clock Speed? 167 MHz?

- Other Examples
  - IDE (Integrated Drive Electronics)
    - Generic Disks & CD-ROMS
  - SCSI (Small Computer Systems Interface)
    - Fast Disks & Devices like scanners, printers
    - 160 MB/s
  - USB (Universal Serial Bus)
    - USB1 1.5 MB/s, central controller
    - originally for “slow” devices like KBD & mouse
  - FireWire (aka IEEE 1394)
    - 50 MB/s, Camcorders, etc.

Hardware Concerns for OSes

- CPU Instruction Set
  - RISC? CISC?
  - arithmetic, logical, control, others (eg Pentium MMX graphics & PowerPC Altivec Digital Signal Processing instructions)
  - (privileged) instructions for OS support?
    - aka “system calls”

- CPU hardware support
  - any special registers for OS?
  - dual mode bit (user/monitor mode)
  - timers?
  - interrupt handling
Interrupts

- **DEFN**: an event requiring attention; it is a signal to the CPU that something needs attention

- Each possible interrupt has:
  - associated dedicated memory location containing address of interrupt routine (found in interrupt vector).
  - a priority level → interrupt is “immediately” handled only when interrupt priority is higher than processor priority level. Otherwise interrupt is deferred (queued for later handling) by hardware.

- Classes (Examples?)
  - Program (aka “traps”)
  - Timer
  - I/O
  - Hardware failure

Interrupt Handling
Steps in Handling Interrupts

• HARDWARE EVENTS
  – device/system issues an interrupt
  – processor finishes execution of current instruction
  – processor acknowledges interrupt (sends signal)
  – push CPU/process status registers onto control stack
  – load new PC value from INTERRUPT VECTOR for interrupt (based on type, device, etc.)

• SOFTWARE EVENTS
  – raise priority level?
  – save process state info from control stack
  – perform processing for interrupt
  – restore process state info
  – lower priority?
  – restore old status registers and old PC (must be last action!)

Interrupt Handling & Priority Levels
Interrupt Handling & Priority Levels

Hardware: Fetch/Execute Cycle, in Detail

1. Fetch Instruction pointed to by PC
2. Increment PC
3. Check for Interrupt; Handle if necessary
4. BRANCH?
   - YES
     - PC ← Address(DR)
   - NO
     - Fetch operands & Execute Instruction
5. INTERRUPTS ENABLED?
   - YES
   - Check for Interrupt; Handle if necessary
   - NO

OS Development

• Machine architectures and OSes are closely tied together

• Generation 0 (1945 -- 1955)
  – single-user machines, vacuum tubes
  – no programming languages, programmed in binary!
  – no OS

• Generation 1 (1955 -- 1965)
  – transistor technology, punch cards
  – CPUs faster than I/O (‘twere ever so!)
  – “batch processing systems”
    • defn: in a batch system there is no interaction between user and job while job executes.
    • Example: collect “batch” of jobs from cards onto tape, tape is Xferred to faster machine
  – a “resident monitor” program runs batches sequentially (1st primitive OS)

OS Development

• Generation 2 (1965 -- 1980ish)
  – IC-board technology, “families” of machines
  – OSes had to be generic!
  – SPOOLING develops
    • Defn?
    • Examples?
  – “Multiprogramming:” several jobs are in memory
  – “Virtual Memory”
  – “Timesharing:” (MULTICS) several users/online terminals using single CPU
  – UNIX!! “one-user”

• Generation 3 (1980ish -- 1990)
  – Personal Computer Age!
  – Assumption: users will be using software they didn’t write
    • OSes had to support “user-friendly” software
  – MS-DOS, MacOS, Unix, VAX/VMS
OS Development

• Generation 4 (1988ish -- 2000ish)
  – Age of Networking
  – RISC architectures vs. CISC architectures
  – machines with >1 CPU becoming more prevalent (multiprocessors)
    • symmetric multiprocessing vs. asymmetric
  – networks of machines becoming common (distributed systems)
  – OSes need to support efficient parallelism and effective security!
  – “Real Time” OSes: jobs *must* be completed by some deadline, or else their output is useless (soft vs. hard deadlines)

• Generation 5 (2000 -- ??)
  – Portable Computing (Back to the Future - sometimes no Virtual Memory)
  – Ubiquitous Computing (embedded programming)
  – 64-bit buses
  – Security?
  – Future???

OS Organization

• “System Call” Idea
• Monolithic Organization (Simple Structure)
• Layered Approach
• Microkernel (Client/Server)
• Virtual Machines
OS Organization: System Calls

• Users often need to execute “privileged” instructions or access “privileged” data (“monitor mode” only).

• For security purposes, OS insures all user programs run in “user mode” at all times.

• Solution: use system calls to alert OS to handle things indirectly for user.
  – NB: Java does not give direct access to these!
  – Text claims there are 5 general categories
    • System calls can also provide API support
  – Examples
    • file attribute set/get
    • get/set sys time
    • open/close file
    • draw window?

BRIEF Intro to Unix/Linux Shell

• “Shell” refers to the process that “sits between” the user and the Unix/Linux OS
  – often makes system calls on behalf of user

• Typical commands:
  – passwd: sets new password for account
  – man: show manual pages for a command
  – pwd: show current directory
  – date: show current date/time
  – ls: show files in current directory
  – cd: change current directory to something else
  – cat: display file contents on standard out
  – lp: print file to a printer
  – wc: show number of characters/words/lines in a file
  – sort: sort file contents line-by-line according to some criteria fields
  – chmod: set protections on files
  – grep: look for files containing a string
  – rm,cp,ln:
  – ps:
Unix/Linux Intro, continued...

- **Command Invocation**
  - flags: `ls -l`, `man -k “window”`
  - arguments: `grep -n “klassner” file.txt`

- **More powerful “commands”**
  - sed: a simple stream editor
  - awk: more powerful stream editor
  - NB: Perl is replacing these (see WWW)

- **Metacharacters**
  - have special meaning beyond their appearance
  - `*,?,$, &`

- **Supports redirection and piping of program I/O:**
  - e.g. `cat *.c | grep “Author: Dr. Klassner” > temp.txt`
  - “|” - pipe
  - `>`,`<<>`

---

**OS Organization: Monolithic System**

[Diagram of OS organization]

Organizationally, every OS procedure is visible to every other one.

However, each procedure does have a well-defined interface in terms of parameters & results.
OS Organization: Layered System

Procedures/Processes in outer rings make “system calls” to those in inner rings to accomplish their tasks.

Benefit: Hardware enforces checking of parameters in calls.

OS Organization: Microkernel (Client/Server)

OS functions split off into own processes
OS kernel handles process communication only
OS Organization: Virtual Machines

- **KEY:** VM provides interface identical to computer hardware
  - **Benefits**
    - software-based emulation is cheaper than hardware
    - OS R&D (protection between kernels on emulated hardware)
  - **Drawbacks?**

### Example 1: Java as a language for a virtual machine

- Java program
  - bytecode
  - Java Virtual Machine
  - OR
  - Browser
  - JVM
  - Operating System (Monitor)
  - Hardware
OS Organization: Virtual Machines

• Example 2: PowerPC (Macintosh platform)
  – New processor runs “old” binaries through virtual machine for 68000 chip
  – Pro: Solve compatibility problems
  – Con:

Major OS Components

• Command Interpreter
  – includes GUI Interface concepts

• Process Management
  – process scheduling!

• Memory Management

• File Management

• I/O Management

• Protection System

• Networking (Distributed System Management)
BIG TOPIC: Asynchronous Concurrent Processes

• “Process”
  – basic unit of work in modern OS
  – a program in execution (NOT == a program!)

• “Concurrent”
  – CPU is time-shared by sequential processes
  – this concurrency is “logical”
    • CPU-sharing details invisible to processes
  – advantages?
    •
    •
    •

• “Asynchronous”
  –
Process Conceptualization

cooking activity == process1

INTERRUPT! (crying child)

first-aid activity == process2
Processes from OS’s View

• A process is a user of resources
  – Examples?
  –

• OS has three responsibilities
  – control execution of processes on CPU
  – share resources fairly among processes that are ready to use them
  – “queue up” processes that have to wait for resources

• As processes await, gain, lose, and release resources, they advance through various life cycle states

Idealized Process Life Cycle

- Creation
- Ready to Run
- Running
- Blocked “waiting for something”
- Termination
OS Representation of Processes (General)

- OS maintains a set of Process Control Blocks (PCBs)
- one PCB per process
- PCB contains (or has pointers to) all info about a process:
  - 
  - 
  - 
  - 

- “state” of a process may be explicitly in PCB, or may be defined implicitly by placement on some system queue.
**System Queues**

- These are lists of processes in various stages of the process life cycle

- **Examples:**
  - Job Queue
  - Ready Queue
  - Device Queue

- **The SCHEDULER(s) of an OS are responsible for moving PCBs from queue to queue**
  - Short-Term scheduler:
  - Long-Term scheduler: (not often implemented)
  - Medium-Term scheduler: (seen mostly in time-shared systems)

**OS and Process Concept**

- **NEAT CONCEPTS:**
  - OS is just another program
  - OS relinquishes control of CPU periodically
    - must depend on timer interrupts or system calls to “bring it back”

- **Can OS be a process?**
Process Switching

• What happens when the current process X must be switched with process Y?

• Answer:
  – -1. Current process X is running
  – 0. Interrupt is signalled (lookup interrupt vector).
  – 1. Hardware puts PC, Regs., etc. on stack
  – 2. Hardware loads new PC from interrupt vector
  – 3. Interrupt handler (assembly code) saves stacked info to PCB
  – 4. Interrupt handler sets up new stack
  – 5. OS code runs, marks Y as READY.
  – 6. OS Scheduler code runs, selects next process to run
  – 7. control passed back to interrupt handler
  – 8. handler loads in “next” process’ PCB info into registers, etc.
  – 9. “next” process is now running.

Specifying Processes through Programming

• NOTE: When we say “specify a process in code,” we’re really saying “how do I show in my program where to start a new process”

• Fork (UNIX)

• JAVA
Fork (UNIX/C)

- “fork” call creates new child process
  - 1 process executes fork(), but 2 return values
- A “copy” of parent process’ address space & context is made, EXCEPT FOR...
  - NO shared data structures between parent & child processes
    - communication?

```c
S0;
if ( ( proc_id = fork() ) == 0 )
{
    . . . /* executed by child process */
    exit(); /* child dies */
}
S1;
x = wait(); /* wait for some child to die */
S2;
```

Fork() Implementation

Before “fork ( )” call

PCB 1 (Process Control Block 1)

![Diagram of PCB 1 with status, id, proc code seg, data seg, stack seg, next, PSW, and R0 registers labeled]
Fork( ) Implementation

After “fork( )” call

Parent Process

PCB 1

status = 1
id = Y Parent-ID=W
proc code seg
data seg
stack seg.

PCB 2

status = 1
id = X Parent-ID=Y
proc code seg
data seg
stack seg.

child process

PCB 2

next = nil
next

Proc code

Fork Events in Unix

1. FORK( ) causes system call interrupt
2. Raise priority level to mask out some interrupts
3. Find some free PCB & free process ID, X
4. Initialize Child:
   a. PCB.Parent := PID of current process
   b. copy Parent’s proc ptr to child PCB
   c. allocate space for child stack & data segments
   d. COPY Parent’s stack & data area to child’s
   e. fix up child’s stack
   f. set child status to ready
5. Add child PCB to Ready Queue
6. Lower Priority level
7. Return from interrupt, continuing with current process
Yet Another View of Fork() & Multiprocessing

```c
x = 1;
if ((pid = fork()) == 0)
    { x = x + 1; /* child starts here */
      printf("%d\n", x);
      exit(); /* child terminates here */
    }

x = x + 2; /* parent continues here */
printf("%d\n", x);
```

### MACHINE CODE

- `LOD r1,1`
- `MOV r1, @x`
- `JMP (fork)`
- `BNZ r0, 12`
- `MOV @x, r1`
- `INC r1,1`
- `MOV r1, @x`
- `CALL (printf)`
- `JMP (exit)`
- `MOV @x, r1`
- `INC r1,2`
- `MOV r1, @x`
- `CALL (printf)`

---

Process Creation in Java

- **Note:** Java really doesn’t implement processes per se, but provides a clean interface to spawn processes to run system programs outside of the JVM

- **“Process” class in Java**
  - `Runtime.getRuntime.exec(""")` method will return a process object (interface) that executes what is in the string argument
  - all process objects support the following methods:
    - `waitFor()`
    - `exitValue()`
    - `destroy()`

```java
try {
    Process child1 = Runtime.getRuntime().exec("/bin/ls");
    InputStream in = child1.getInputStream();
    // echo output of 'ls'
    child1.waitFor();
    System.out.println("Child exited with "+ child1.exitValue());
} catch(IOException e) {
    System.err.println(e);
}
```