Outline

- Announcements
  - Project groups
  - Questions?

- Computer-system structures
  - I/O structures (cont'd)
  - Storage structures and hierarchy
    - memory, secondary storage, tape
  - Hardware support for protection

- Operating-system structures
  - different views: functional, components, services, structure

- Processes
  - the process concept
  - process scheduling

(Review) I/O Operation

- Two approaches: Synchronous and Asynchronous

  - Problem with the above schemes: CPU handles all I/O
    - it can spend all its time doing interrupt processing
      - disk I/O, network I/O, video I/O

Solution: Direct Memory Access (DMA)

- The main idea: add a special device to “intervene” between the device controller and the system's memory

  - Operation
    - the CPU tells this DMA controller
      - the “chunk” size to be transferred
        - e.g., 128 - 4096 bytes (sectors) for disks
      - the starting address in memory where this chunk ought to be stored
    - the DMA controller
      - accesses the secondary device via its controller
      - transfers the chunk from the device to system memory (and vice-versa)

  - Benefit: Interrupts are now less frequent
    - at the level of chunks of data: only to indicate completion
    - hence, CPU can do a lot of work between interrupts
Memory-Mapped I/O

- Traditionally, the CPU could directly access only main memory
  - all other devices are handled via controllers
  - accessed using special I/O instructions

- In most recent systems
  - controller’s registers are mapped into RAM space
    - results in uniform treatment of the I/O devices
    - can be handled via memory management procedures
    - all addressing is to RAM space
    - DMA access, interrupt handling, polling, …
  - controller’s buffers are mapped into RAM space
    - makes sense if the I/O is to a device that is particularly fast
    - e.g., a CRT screen where each pixel is an addressable location in RAM

Computer-System Structures (2): Storage

- Primary storage: Main memory (volatile)
  - accessed directly using load/store instructions
    - 1 cycle (registers), 2-5 cycles (cache), 20-50 cycles (RAM)
    - before: only one outstanding memory operation, CPU waits for completion
    - now: several outstanding operations

- Secondary storage: Disks (non-volatile)
  - accessed using a disk controller
  - supports random access but with non-uniform cost

- Tertiary storage: Tapes (non-volatile)
  - typically used only for backup
  - very inefficient support for random access

- Organized as a hierarchy
  - small amount of faster, more expensive storage closer to the CPU
  - larger amounts of slower, less expensive storage further away

Storage Hierarchy

- Rationale
  - keep CPU busy: lots of fast memory
  - keep system cost down

- How does it work
  - caching: upon access, move datum or instruction and its neighbors into higher levels of the hierarchy
  - replacement when a level fills up
  - copies need to be kept coherent

- Why does it work
  - Real programs demonstrate locality
    - e.g., rows and columns of a matrix
    - e.g., sequential instructions
  - once a datum or instruction is used, things “near” them are likely to be used “soon”

Computer-System Structures (3): Protection

- Goal: Prevent user processes from accidentally/maliciously damaging
  - the OS structures
  - parts of other process’s memory space
  - other user’s I/O devices

- Mechanisms address different ways in which protection breaks down
  1. dual-mode operation
    - Prevent user process taking over part of the OS and using this to overwrite other processes or even modify the OS itself (as in MS-DOS)
  2. privileged instructions
    - Prevent user process intervening in I/O of another process via control of the I/O handlers and indirectly causing damage
  3. memory protection
    - Prevent user process directly accessing another user process’ storage
  4. CPU protection via timers
    - Prevent hanging the OS -- e.g., via an infinite loop
Protection Mechanisms (1):
Dual-mode Operation and Privileged Instructions

- Dual-mode operation
  - supervisor and user modes
  - system starts off in supervisor mode and reenters it for interrupt processing
  - operating system gains control in supervisor mode

- Privileged instructions
  - restrict use of certain instructions to supervisor mode
    - I/O, including interrupt control
    - may be done by memory mapping
    - affect memory mapping
    - affect CPU mode (user/supervisor)
  - hardware support crucial for performance and for atomicity

Protection Mechanisms (2):
Memory Protection

- Basic method: Memory is divided into segments
  - Furthermore
    - logical addresses are mapped to physical addresses
    - provides sharing, etc.
    - hardware support for address mapping
    - a memory protection violation is detected
      - user process traps to (interrupts) the OS

Protection Mechanisms (3): Timers

- OS code can enforce policies only if it gets a chance to run

- Timers maintain a count of elapsed (system) clock ticks
  - when timer expires, the CPU is interrupted → run the OS code

- Used for
  - interrupting hung processes
  - context switching in time-shared systems

- Access to timers is (usually) privileged
  - WHY?

Outline

- Announcements
  - Project groups
  - Questions?

- Computer-system structures
  - I/O structures (cont’d)
  - Storage structures and hierarchy
    - memory, secondary storage, tape
  - Hardware support for protection

- Operating-system structures
  - different views: functional, components, services, structure

- Processes
  - the process concept
  - process scheduling

[Silberschatz/Galvin/Gagne: Chapters 2-4]
Hardware and OS Structures

User Applications

Support Applications
Compilers, Linkers, Windowing Systems, …

User-mode
Protection and Security
Process Management
I/O Device Management

Kernel-mode
Storage Management
Networking

Software

Hardware
Storage
memory
disk, tape
interrupts
DMA

Controllers

CPU
priv. instructions
memory protection
timers

Functional View
what does it do?

Components View
what does it contain?

Services View
what does it provide?

Structure View
how is it built?

OS Views (1): Functional View

• What are the functions performed by an OS?

• Explicit operations
  – program execution and handling
  – I/O operations
  – file-system management
  – inter-process communication
  – exception detection and handling
    • e.g., notifying user that printer is out of paper

• Implicit operations
  – resource allocation
  – accounting
  – protection
    • e.g., maintaining data integrity, logging invalid login attempts

OS Views (2): Components View

• Processes: run-time representations of user programs
  – create, terminate, suspend, resume
  – access to shared resources (e.g., printers)

• Storage
  – allocation of memory among resident processes
  – disk management (e.g., scheduling of disk accesses)

• I/O
  – device drivers, handling of device interrupts
  – files and directories

• Protection
  – user access to system resources

Course organization follows this view

OS Views (3): Services View

• Two issues
  – What services does an OS provide? (same as functional view)
  – How do users and user programs access these services?

• Interface between the user and the OS: Command Interpreter
  – typical commands
    • process creation and (implicitly) destruction
    • I/O handling and file system manipulation
    • communication: interact with remote devices
    • protection management: changing file/directory access control, etc.
  – different varieties
    • the interpreter contains the code for the requested command (e.g., delete)
    • the interpreter calls a system routine to handle the request
    • the interpreter spawns new process(es) to handle the request
      – process lookup through some general procedure

▶ you will implement a simple shell in Nachos Lab 5
OS Views (3): Services View (contd.)

- Interface between a user program and the OS: System Calls
  - arguments passed in registers, a memory block, or on the stack
  - entry into the kernel using the trap mechanism

- Standard system calls
  - process control
  - file manipulation
  - device manipulation
  - information maintenance
    - get/set system data (time, memory/cpu usage), process and device attributes
  - communications

OS Views (4): Structure View

- How to structure OS functionality
  - Layering
  - Microkernels
  - Virtual machines
- Designing and implementing an OS

- Read Sections 3.5-3.9, Silberschatz, Galvin, and Gagne
- Look at Nachos source code
  - Thomas Narten’s roadmap

Outline

- Announcements
  - Project groups
  - Questions?
- Computer-system structures
  - I/O structures (cont’d)
  - Storage structures and hierarchy
    - memory, secondary storage, tape
  - Hardware support for protection
- Operating-system structures
  - different views: functional, components, services, structure
- Processes
  - the process concept
  - process scheduling

What is a Process?

- A process is a program in execution.
- The components of a process are:
  - the program to be executed
  - the data on which the program will execute
  - the resources required by the program—such as memory and file(s)
  - the status of the execution
- A process is the unit of
  - resource ownership
  - protection
  - dispatching

[ Silberschatz/Galvin/Gagne: Chapters 2-4]
The State of a Process

- Can be one of: **New, Ready, Running, Waiting, Stopped, Terminated**

### Process Control Information

#### Process Control Block (PCB)

- **Process #**
- **Proc. status**
- **Program ctr.**
- **Register save area**
- **Memory-management information**
- **Accounting information**
- **I/O status information**
- **Scheduling information**

#### Scheduling Processes

- “Decide which process to run next”

#### Concern 1: Process Context Switch

- **ProcessA** executes
- **ProcessB** executes

Look at the Nachos code: Thread::Yield, SWITCH

Nachos Lab 1: Consequences of asynchronous context switches
Concern 2: Process Queues

<table>
<thead>
<tr>
<th>State</th>
<th>Transition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>new</td>
<td>create</td>
<td>new processes are created and added to the ready queue</td>
</tr>
<tr>
<td>ready</td>
<td>dispatch</td>
<td>processes are dispatched to the CPU</td>
</tr>
<tr>
<td>running</td>
<td>exit</td>
<td>processes are removed from the ready queue</td>
</tr>
<tr>
<td>terminated</td>
<td>kill</td>
<td>processes are terminated</td>
</tr>
<tr>
<td>waiting</td>
<td>event or resource available</td>
<td>processes are suspended</td>
</tr>
</tbody>
</table>

Concern 3: Schedulers

- **The long-term scheduler**
  - **operation**: creates processes and adds them to the ready queue
  - **frequency**: infrequent, ~minutes
  - **objective**: maintain good throughput by ensuring mix of I/O and CPU jobs

- **The short-term scheduler**
  - **operation**: allocates CPU and other resources to ready jobs
  - **frequency**: frequent, ~100 ms (a context switch takes ~10s of µsecs)
  - **objective**: ensure good response times in time-sharing systems

- **The medium-term scheduler**
  - **operation**: swaps some processes out of the short-term scheduler’s loop
  - **frequency**: somewhere between the short- and long-term schedulers
  - **objective**: to prevent over-multiprogramming (thrashing)
    - required when the long-term scheduler underestimates process requirements