CS370 Operating Systems

Colorado State University
Yashwant K Malaiya

Fall 2016 Lecture 27

Deadlocks
Main Memory

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

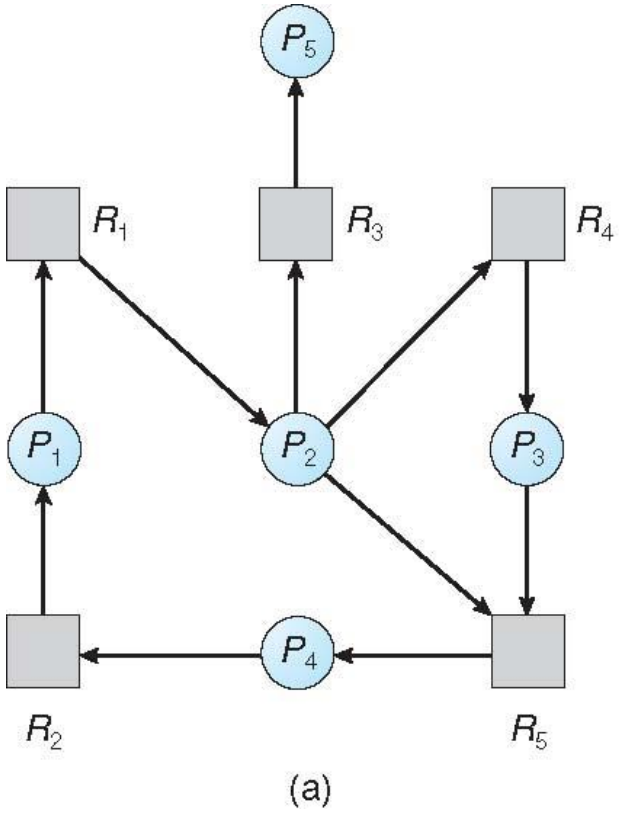
FAQ

- Representing resource allocation graph, wait for graph in 2D? Need nodes and connectivity.
- Terms: Our terms are from the text book.
  - Bankers algorithm for safety: Allocation, Need (Max – Allocation)
  - Bankers algorithm for detection: Allocation, Request
  - Available (initially), Available (at a given instant)
- Can there be multiple safe sequences?
- Usefulness of deadlock detection? Deadlocks may occur in databases, manufacturing systems, communications etc.
- Can we find out exactly an OS is implemented? Linux 4.8 kernel 23 M LOC, Windows 50 M LOC?

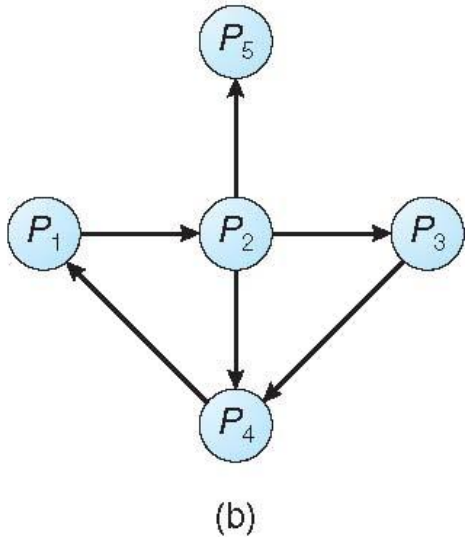
Notes

- PA4 due Thursday, PA5 (Producer/Consumer) will be available
- iClickers: Please see note on Piazza.
- Quiz questions
- Project: Detailed abstract Nov 9
- Quiz 9 Fri?

Resource-Allocation Graph and Wait-for Graph



Resource-Allocation Graph



Corresponding wait-for graph

2 cycles. Deadlock.

Colorado State University

Example of Detection Algorithm

- Five processes P_0 through P_4; three resource types A (7 instances), *B* (2 instances), and *C* (6 instances)

- Snapshot at time T_0:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Sequence <P_0, P_2, P_3, P_1, P_4> will result in ***Finish[i] = true*** for all i. ***No deadlock***

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur
  - How many processes will need to be rolled back
    - one for each disjoint cycle
- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.

Recovery from Deadlock: Process Termination

Choices

- Abort all deadlocked processes

- Abort one process at a time until the deadlock cycle is eliminated

  – In which order should we choose to abort?

  1. Priority of the process
  2. How long process has computed, and how much longer to completion
  3. Resources the process has used
  4. Resources process needs to complete
  5. How many processes will need to be terminated
  6. Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

Take resources from some processes and give to others

- **Selecting a victim** – minimize cost
- **Rollback** victim– return to some safe state, restart process for that state, or abort and restart
- **Starvation** – same process may always be picked as victim,
  - include number of rollback in cost factor

Deadlock recovery through rollbacks

- **Checkpoint** process periodically
  - Contains memory image and resource state
- Deadlock detection tells us *which* resources are needed
- Process owning a needed resource
  - **Rolled back** to before it acquired needed resource
    - Work done since rolled back checkpoint discarded
  - **Assign** resource to deadlocked process

Livelocks

In a livelock two processes need each other's resource

- Both run and make no progress, but neither process blocks
- Use CPU quantum over and over without making progress

Ex: If fork fails because process table is full

- Wait for some time and try again
- But there could be a collection of processes each trying to do the same thing
- Avoided by ensuring that only one process (chosen randomly or by priority) takes action

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2016 Lecture 27

Main Memory

Slides based on

- **Text by Silberschatz, Galvin, Gagne**
- **Various sources**

Chapter 8: Main Memory

Objectives:

- Ways of organizing memory hardware
- Memory-management techniques, including paging and segmentation
- Examples: the Intel and ARM architectures

Chap 8: Memory Management

- Background
- Swapping
- Contiguous Memory Allocation
- Segmentation
- Paging
- Structure of the Page Table
- Example: The Intel 32 and 64-bit Architectures
- Example: ARM Architecture

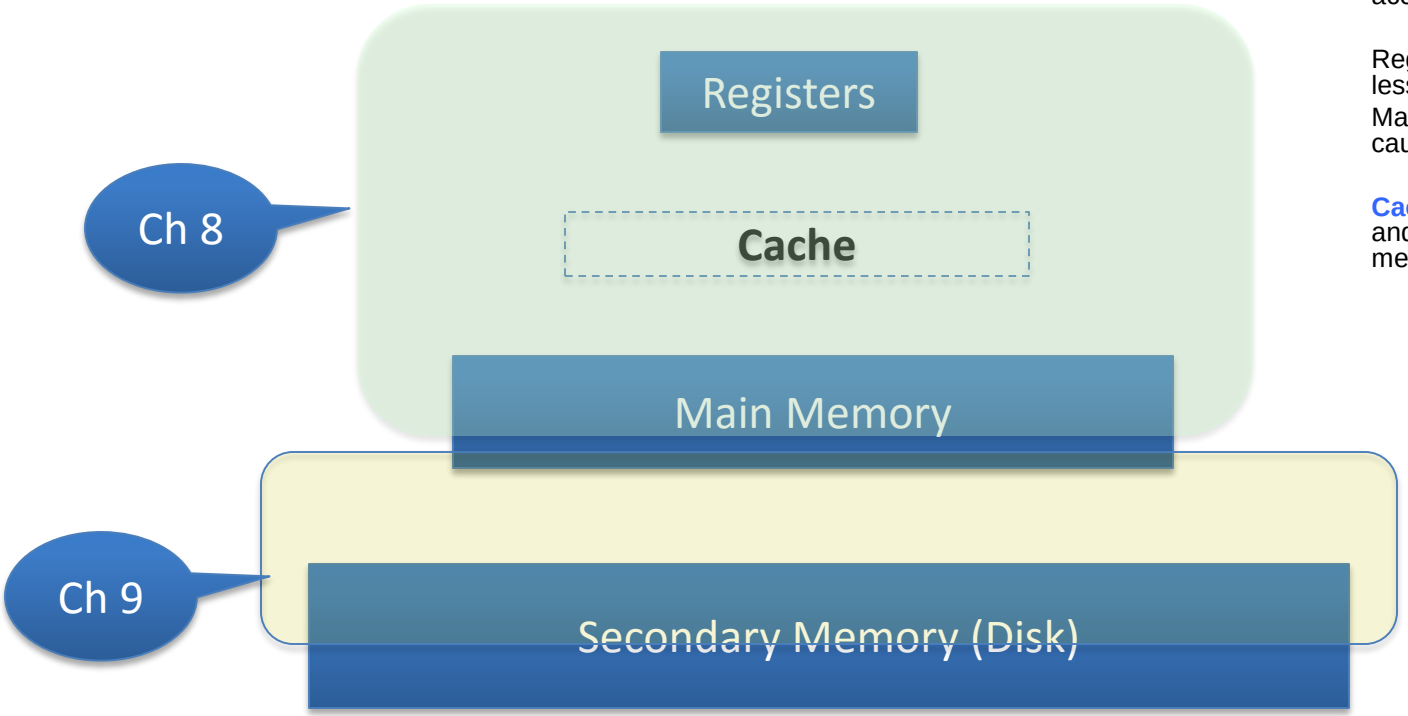
What we want

- Memory capacities have been increasing
  - But programs are getting bigger faster
  - Parkinson's Law: Programs expand to fill the memory available to hold
- What we would like
  - Memory that is
    - infinitely large, infinitely fast
    - Non-volatile
    - Inexpensive too
- Unfortunately, no such memory exists as of now

Background

- Program must be brought (from disk) into memory and run as a process
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of
  - addresses + read requests, or
  - address + data and write requests
- Access times:
  - Register access in one CPU clock (or less)
  - Main memory can take many cycles, causing a **stall**
  - **Cache** sits between main memory and CPU registers making main memory appear much faster
- Protection of memory required to ensure correct operation

Hierarchy



Main memory and registers are only storage CPU can access directly access

Register access in one CPU clock (or less).
Main memory can take many cycles, causing a **stall.**

Cache sits between main memory and CPU registers making main memory appear much faster

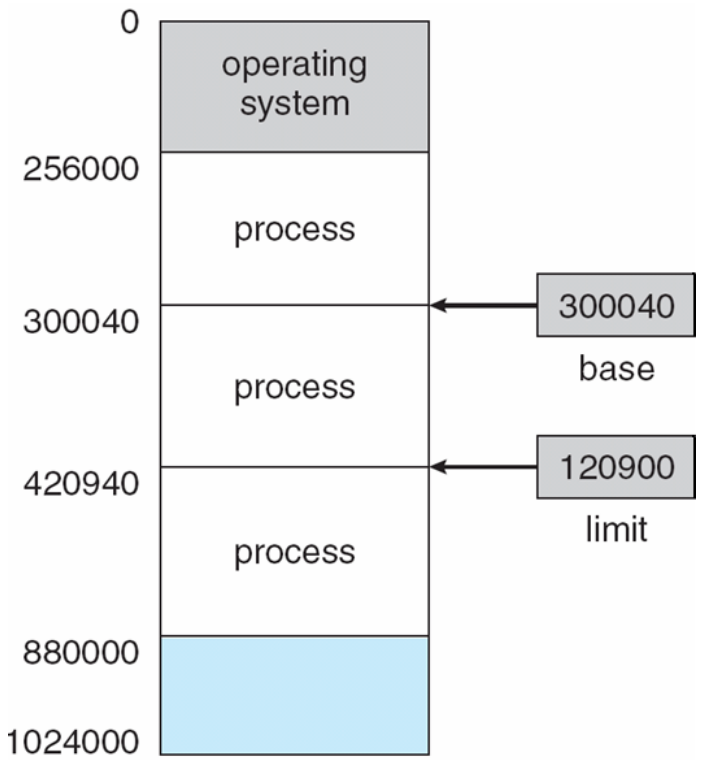
Ch 10,11,12: Disk, file system Cache: CS470

Protection: Making sure each process has separate memory spaces

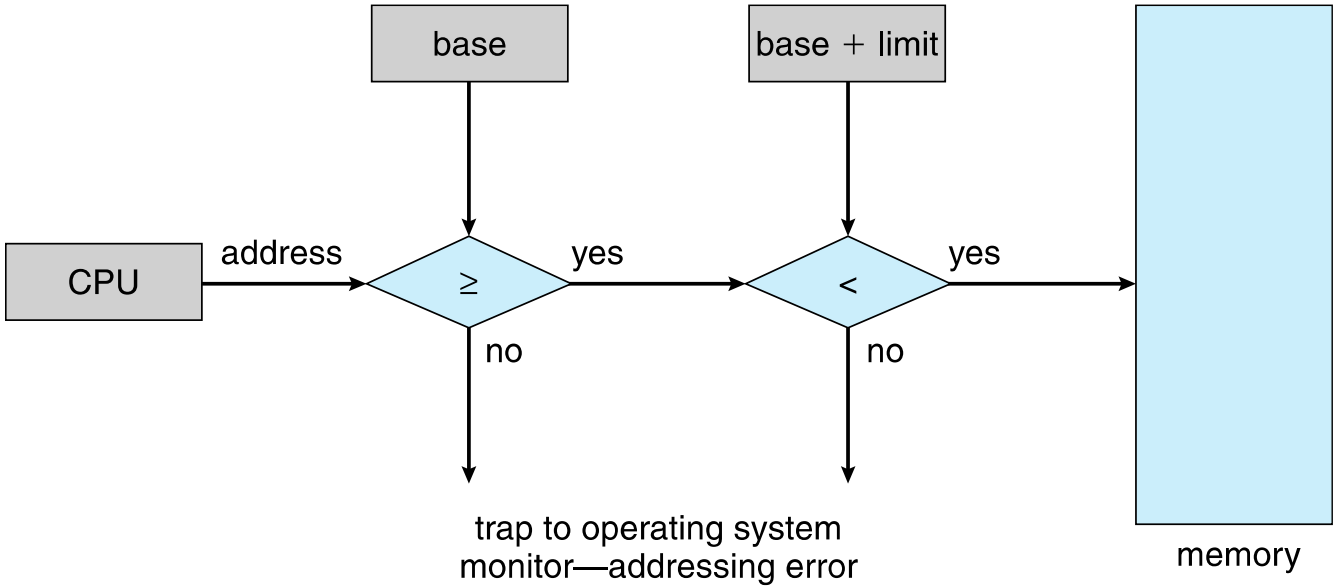
- OS must be protected from accesses by user processes
- User processes must be protected from one another
  - Determine range of legal addresses for each process
  - Ensure that process can access only those
- Approach: Partitioning address space
  - Or separate address spaces (later)
- Base and Limit for a process
  - **Base**: Smallest legal physical address
  - **Limit**: Size of the range of physical address

Base and Limit Registers

- A pair of **base** and **limit registers** define the logical address space for a process
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user
- Base: **Smallest** legal physical address
- Limit: Size of the **range** of physical address
- Eg: Base = 300040 and limit = 120900
- Legal: 300040 to (300040 + 120900 -1) = 420939



Hardware Address Protection



Legal addresses: **Base address** to **Base address + limit -1**

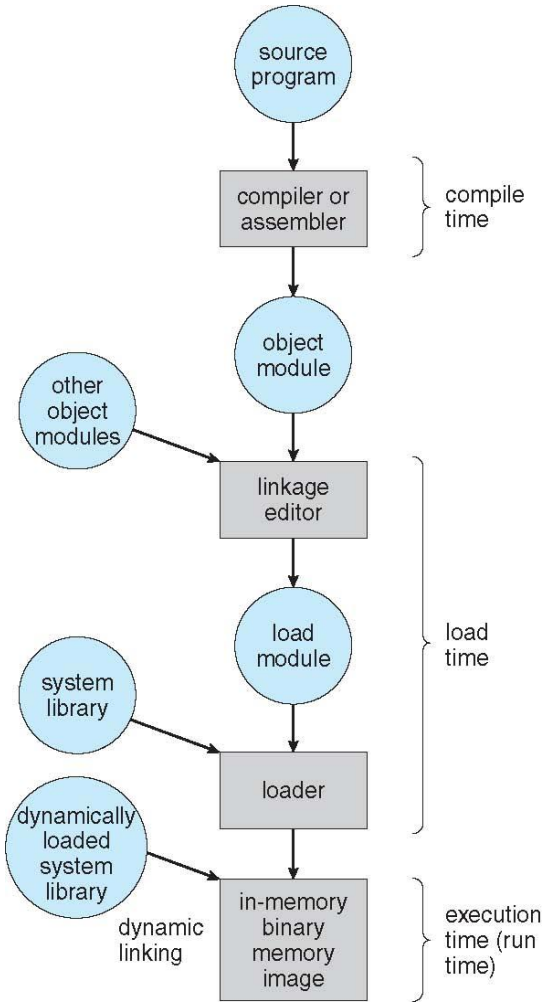
Address Binding Questions

- Programs on disk, ready to be brought into memory to execute form an **input queue**
  - Without support, must be loaded into address 0000
- Inconvenient to have first user process physical address always at 0000
  - How can it not be?
- Addresses represented in different ways at different stages of a program's life
  - **Source code** addresses are symbolic
  - **Compiled code** addresses **bind** to relocatable addresses
    - i.e. "14 bytes from beginning of this module"
  - **Linker or loader** will bind relocatable addresses to absolute addresses
    - i.e. 74014
  - Each binding maps one address space to another

Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
  - **Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes
  - **Load time**: Must generate **relocatable code** if memory location is not known at compile time
  - **Execution time**: Binding delayed until run time if the process can be moved during its execution from one memory segment to another
    - Need hardware support for address maps (e.g., base and limit registers)

Multistep Processing of a User Program



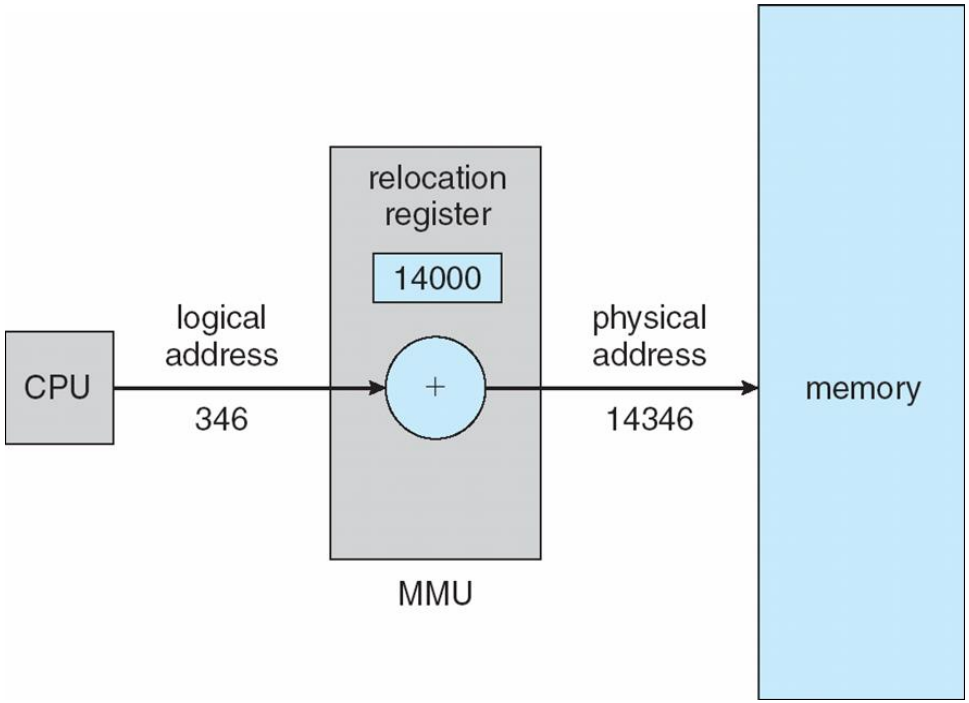
Logical vs. Physical Address Space

- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

Memory-Management Unit (MMU)

- Hardware device that at run time maps virtual to physical address
- Many methods possible, we will see them soon
- Consider simple scheme where the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
  - Base register now called **relocation register**
  - MS-DOS on Intel 80x86 used 4 relocation registers
- The user program deals with *logical* addresses; it never sees the *real* physical addresses
  - Execution-time binding occurs when reference is made to location in memory
  - Logical address bound to physical addresses

Dynamic relocation using a relocation register



Loading vs Linking

- **Loading**
  - **Load executable into memory prior to execution**
- **Linking**
  - **Takes some smaller executables and joins them together as a single larger executable.**

Linking: Static vs Dynamic

- **Static linking** – system libraries and program code combined by the loader into the binary image
  - Every program includes library: wastes memory
- **Dynamic linking** –linking postponed until execution time
  - Operating system checks if routine is in processes' memory address

Dynamic Linking

- **Dynamic linking** –linking postponed until execution time
- Small piece of code, **stub**, used to locate the appropriate memory-resident library routine
- Stub replaces itself with the address of the routine, and executes the routine
- Operating system checks if routine is in processes' memory address
  - If not in address space, add to address space
- Dynamic linking is particularly useful for
  - shared libraries
  - Separate patching

Dynamic loading of routines

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded
- All routines kept on disk in relocatable load format
- Useful when large amounts of code are needed to handle infrequently occurring cases
- No special support from the operating system is required
  - Implemented through program design
  - OS can help by providing libraries to implement dynamic loading