Main Memory: Part 1

COMP 3659 – Operating Systems
Monday, November 8, 2021

Text: Chapter 9 (9.1-9.2)

Announcements and Course Administration

- Programming project 2:
  - Concept memos have been received
  - Except group feedback early this week – watch your email

Overview

- Last lecture:
  - Deadlock: liveness; system model; deadlock in multithreaded applications; characterization; overview of methods for handling (text sections 6.8, 8.1-8.4)
- Starting this week:
  - A two-week unit on memory management: textbook chapters 9-10
- Today's focuses:
  - Background 9.1
  - Contiguous memory allocation 9.2

Lab Debrief

- Discuss: What key things did you learn in the lab?
    - Summary of key learnings
        - About phenomena, interpretations, concepts, emerging themes, …
    - Insights? Questions? New goals?
    - Challenges and successes?
    - Other?

Last Lecture Recap:

- Deadlock
  - A main category of liveness problem (note also livelock)
  - >1 thread waiting for an event that can only be signaled by one of the waiting threads
- Often involves the incorrect introduction of thread synchronization
  - E.g., using semaphores
  - Attempting to solve a race condition can introduce the potential for deadlock!
- Can be visualized using resource allocation graph
  - No cycle → no deadlock; cycle → potential for deadlock (not always)
- E.g., the dining philosophers problem and one potential solution
- Deadlock prevention, avoidance, detection, and recovery
  - kernel can also ignore the possibility of deadlock

Background

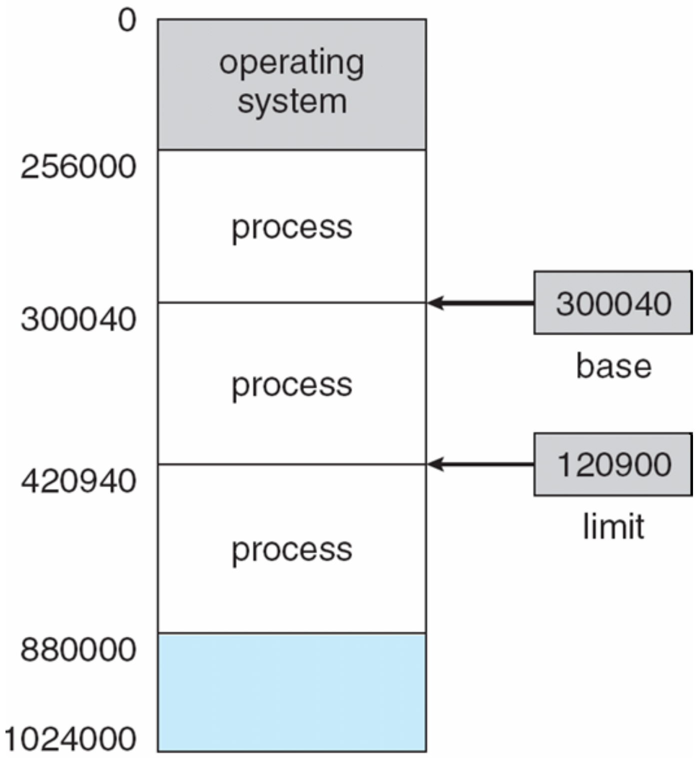
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (or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

The Simplest Model of Memory Protection

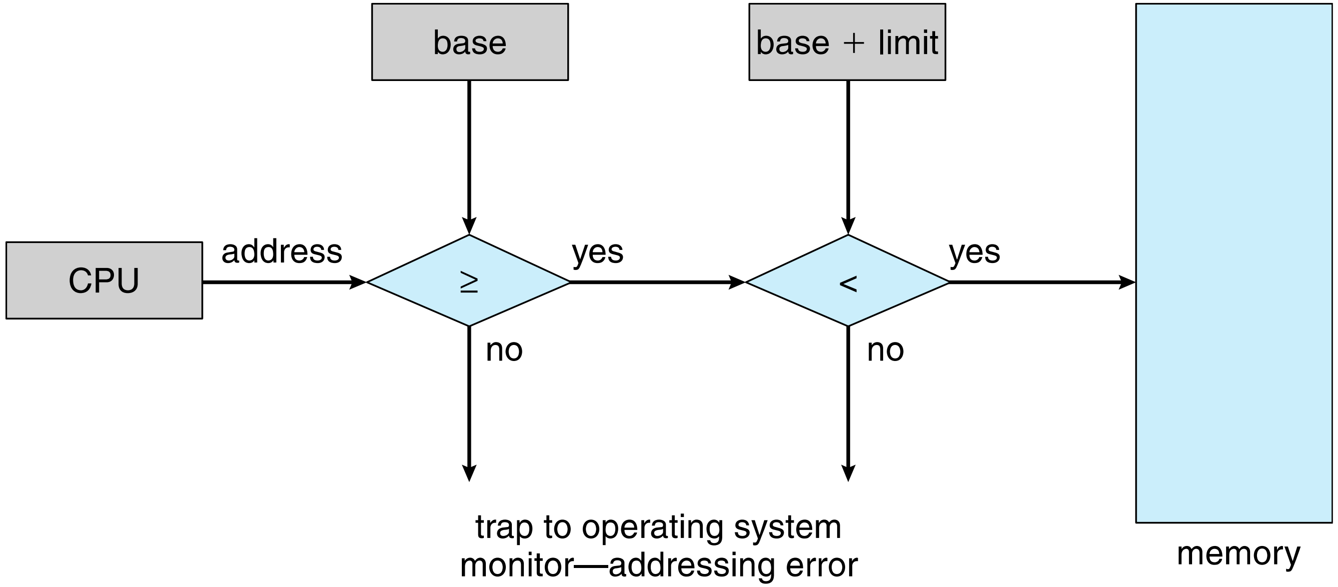
- Multiple process (and the kernel) all co-exist in physical memory
- The operating system allocates non-overlapping regions of physical memory to each
  - Can consider each region to be that process's "logical address space"
- Implemented via:
  - Base register
  - Limit register
- Basically, each process executes in an "address space jail"

Base and Limit Registers

- A pair of **base** and **limit registers** define the logical address space
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



Hardware Address Protection



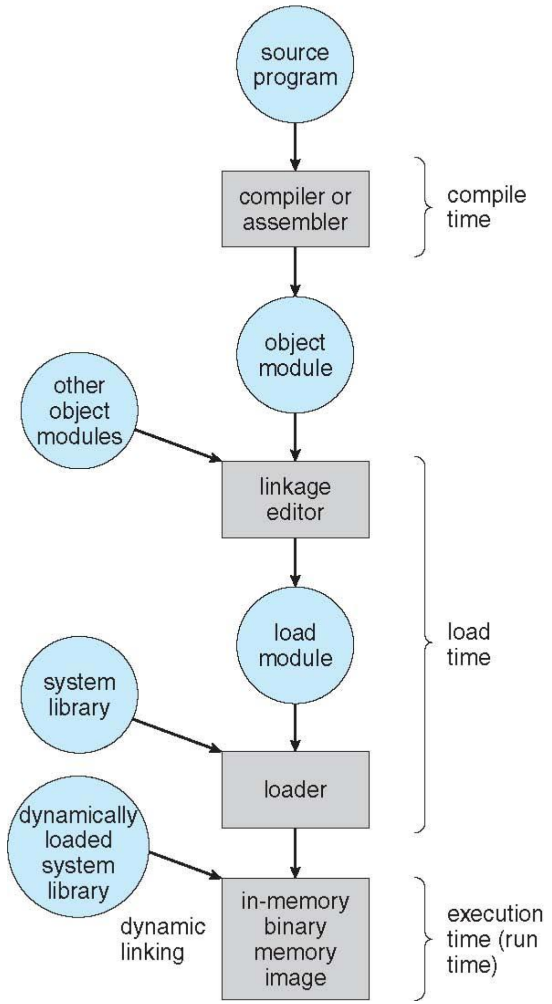
Address Binding

- 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?
- Further, addresses represented in different ways at different stages of a program's life
  - Source code addresses usually 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

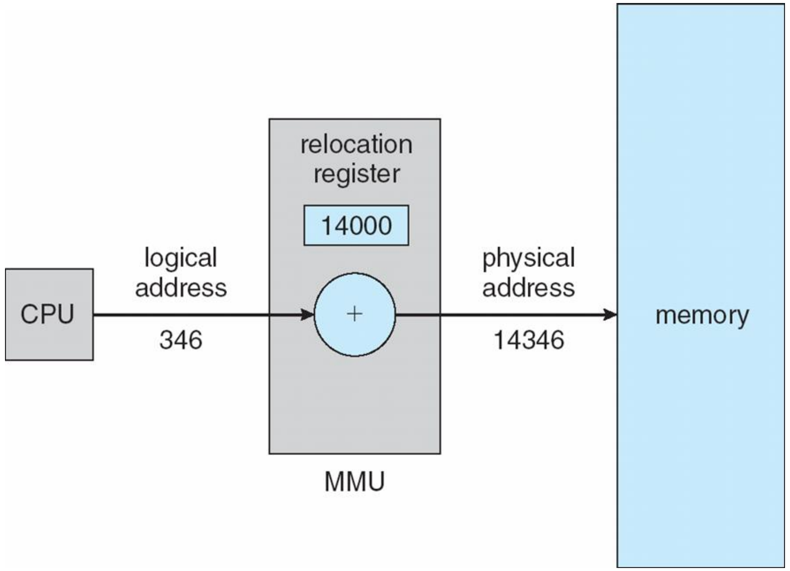
- The concept of a logical address space that is bound to a separate **physical address space** is central to proper memory management
  - **Logical address** – generated by the CPU; also referred to as **virtual address**
  - **Physical address** – address seen by the memory unit
- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **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, covered in the rest of this chapter
- To start, 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

- 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



Dynamic Linking

- **Static linking** – system libraries and program code combined by the loader into the binary program image
- 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 libraries
- System also known as **shared libraries**
- Consider applicability to patching system libraries
  - Versioning may be needed

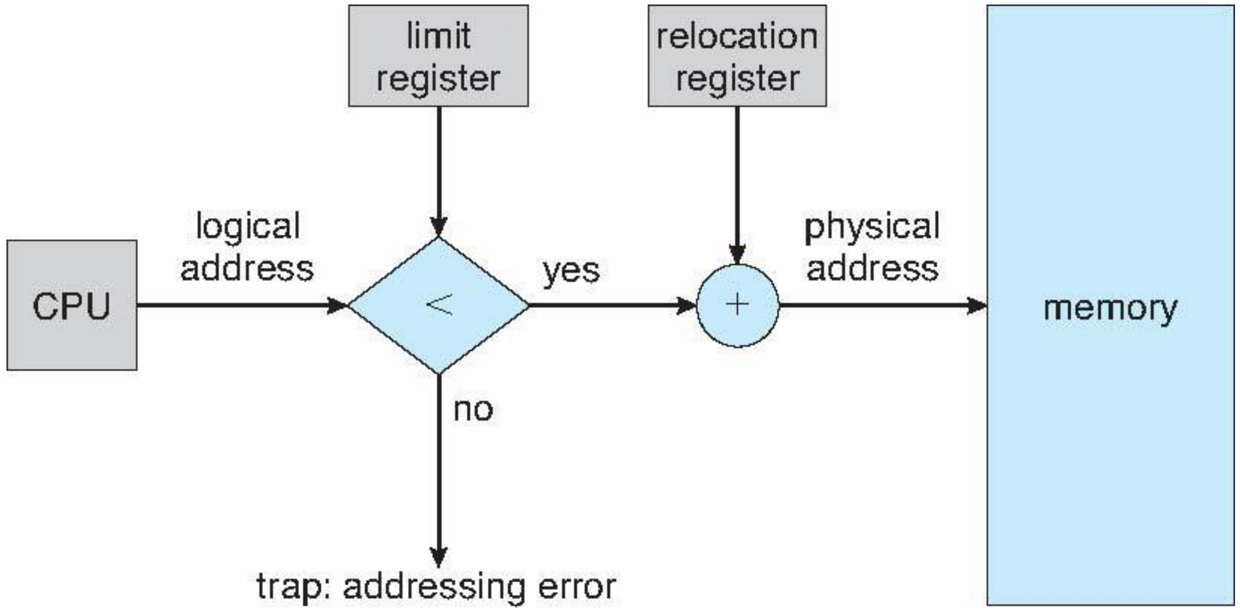
Contiguous Allocation

- Main memory must support both OS and user processes
- Limited resource, must allocate efficiently
- Contiguous allocation is one early method
- Main memory usually into two **partitions**:
  - Resident operating system, usually held in low memory with interrupt vector
  - User processes then held in high memory
  - Each process contained in single contiguous section of memory

Contiguous Allocation (Cont.)

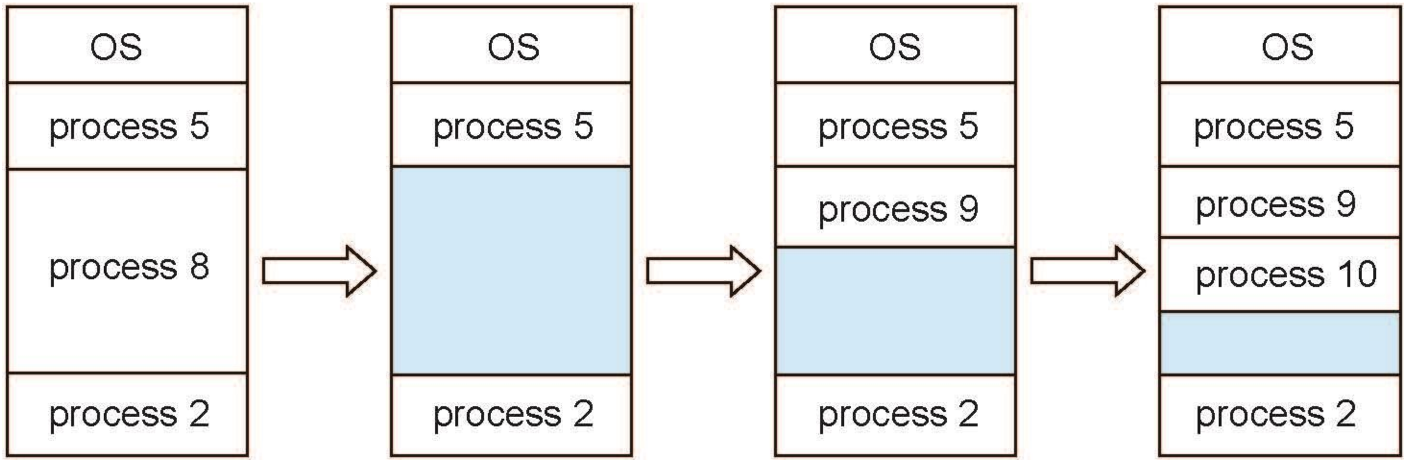
- Relocation registers used to protect user processes from each other, and from changing operating-system code and data
  - Base register contains value of smallest physical address
  - Limit register contains range of logical addresses – each logical address must be less than the limit register
  - MMU maps logical address *dynamically*
  - Can then allow actions such as kernel code being **transient** and kernel changing size

Hardware Support for Relocation and Limit Registers



Multiple-partition allocation

- Multiple-partition allocation
  - Degree of multiprogramming limited by number of partitions
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
  - **Hole** – block of available memory; holes of various size are scattered throughout memory
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Process exiting frees its partition, adjacent free partitions combined
  - Operating system maintains information about:
    a) allocated partitions   b) free partitions (hole)



Dynamic Storage-Allocation Problem

How to satisfy a request of size ***n*** from a list of free holes?

- **First-fit**: Allocate the ***first*** hole that is big enough
- **Best-fit**: Allocate the ***smallest*** hole that is big enough; must search entire list, unless ordered by size
    - Produces the smallest leftover hole
- **Worst-fit**: Allocate the ***largest*** hole; must also search entire list
    - Produces the largest leftover hole

First-fit and best-fit better than worst-fit in terms of speed and storage utilization

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- First fit analysis reveals that given *N* blocks allocated, 0.5 *N* blocks lost to fragmentation
  - 1/3 may be unusable -> **50-percent rule**

Fragmentation (Cont.)

- Reduce external fragmentation by **compaction**
  - Shuffle memory contents to place all free memory together in one large block
  - Compaction is possible *only* if relocation is dynamic, and is done at execution time
  - I/O problem
    - Latch job in memory while it is involved in I/O
    - Do I/O only into OS buffers
- Now consider that backing store has same fragmentation problems

Conclusion

- A modern kernel must allocate space to ≥ 1 processes (and itself)
  - While providing memory protection
  - Raises allocation strategy and fragmentation issues
- One “simple” solution: contiguous allocation with base & limit registers
- A program’s instructions & data must be bound to the (logical) address range provided by the kernel
  - Can only be done at compile time if absolute (logical) addresses known a priori
  - Must be done at load and/or execute time otherwise; code must be relocatable
- A hardware MMU supports relocation
  - And better-distinguishes the processes *logical address space* from physical addresses
- An elegant upcoming solution: paging
- Next:
  - Paging (and related topics) 9.3-9.4, 10.1-10.2

Recommended Post-class Studying

- Review these slides and your lecture notes
  - Ideally within 24-48 hours
  - Then review text sections 9.1-9.2 as necessary
- As always, please let me know if you have any questions