OPERATING SYSTEMS

by
Marwa Yusuf

Lecture 12
Sunday 13-12-2020

Chapter 3 (3.1 to 3.2)

Memory Management

Memory Management

- Memory sizes grow, but program sizes grow faster.
- Parkinson’s Law:
  - “Programs expand to fill the memory available to hold them.”
- That is why OS has to manage memory very carefully.

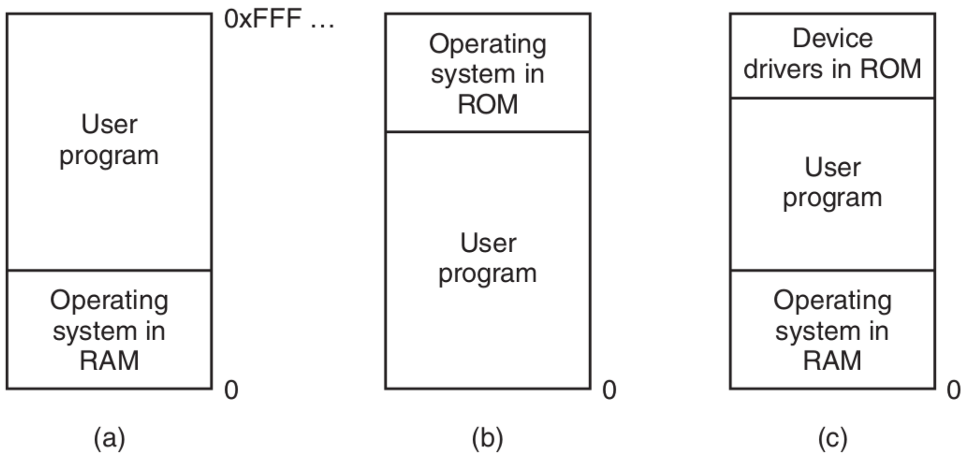
Memory Management

- What we want? Large, Fast, non-Volatile and Cheap memory. But it is just a dream (for now!), so we have to compromise.
- Memory Hierarchy is the solution (registers, cache, RAM, Disks and others).
- OS has to manage all of those.
- **Memory Manager**: Allocate, deallocate and keep track.
- Cache is managed by hardware, Disk will be discussed later. We focus on **RAM**, how to abstract it to the programmer?

No Memory Abstraction

- Early computers presented no abstraction, just the physical memory (a set of addresses starting from 0) as it is.
  - Each address is a cell of bits (commonly 8).
  - Ex: MOV REGISTER1,1000
- Hence, only one program in memory.

No Memory Abstraction (cont.)



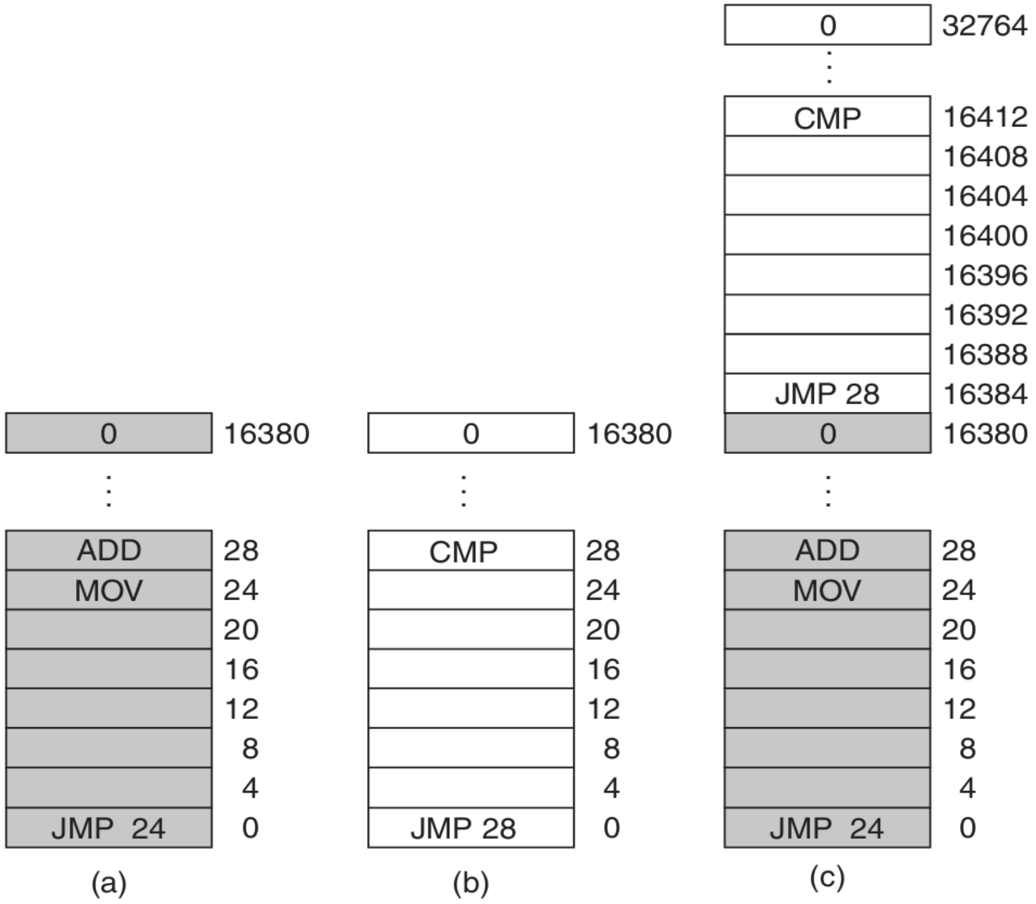
- 1st is rarely used now, 2nd is used with some handheld and embedded, 3rd with early PC (BIOS).
- 1st and 3rd model may lead to disasters, how?
- Only one process at a time, loaded by OS according to a user command.
- To get parallelism, you may use threads, but:
  - We need processes not threads.
  - Can such a primitive system support threads?

Multiple Programs with no Abstraction

- Swapping (between memory and disk).
- With special hardware, multiprogramming is possible without swapping.
- In early IBM 360:
  - Divide memory into 2 KB blocks, each assigned 4 bit protection key held in special registers.
    - Ex: 1 MB memory → 512 key → 256 bytes
  - PSW has protection key. No process can access memory of another.
  - OS only can change keys.

Problem

- The 2nd program will crash immediately.
- They access absolute physical memory.
- Solution by IBM-360: **static relocation**.
  - Slow loading.
  - Info about executable (what is an address?).



Multiple Programs with no Abstraction (cont.)

- No abstraction is still used in embedded and smart card systems:
  - Programs known in advance, not general purpose.

Memory Abstraction: Address Spaces

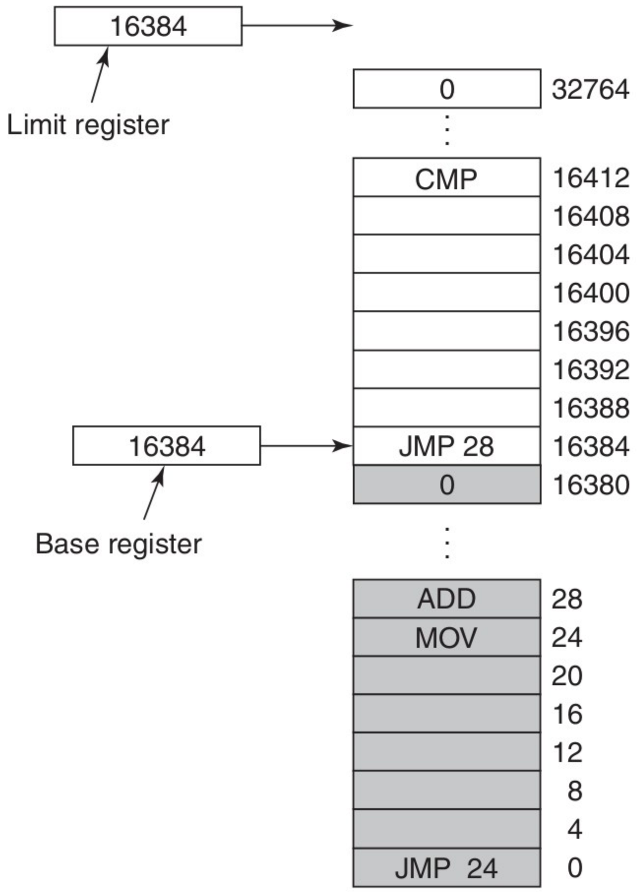
- Problems with physical memory access:
  - If the program can access any address → crash other programs (maybe OS).
  - Difficult for mutliprogramming.
- Two issues to solve:
  - Protection
  - Relocation

The Notion of an Address Space

- Abstraction of memory for programs to live in.
- An **address space** is the set of addresses that a process can use to address memory.
- An address space for each process protected from others (except in case of communication)
- Ex: telephone numbers, x86 ports, web addresses (.com), IPv4.
- Question is: how 28 means different physical locations in different programs?

Base and Limit Registers

- Simple version of **dynamic relocation**.
- Add **base** to each address and compare with **limit** for protection.
- Ex: JMP 28 → JMP 16412 <= 32764
- Only OS can modify **base** and **limit**.
- Intel 8088: no **limit**, several **base**s, allowing for allocation parts separately.
- Disadvantage: ?

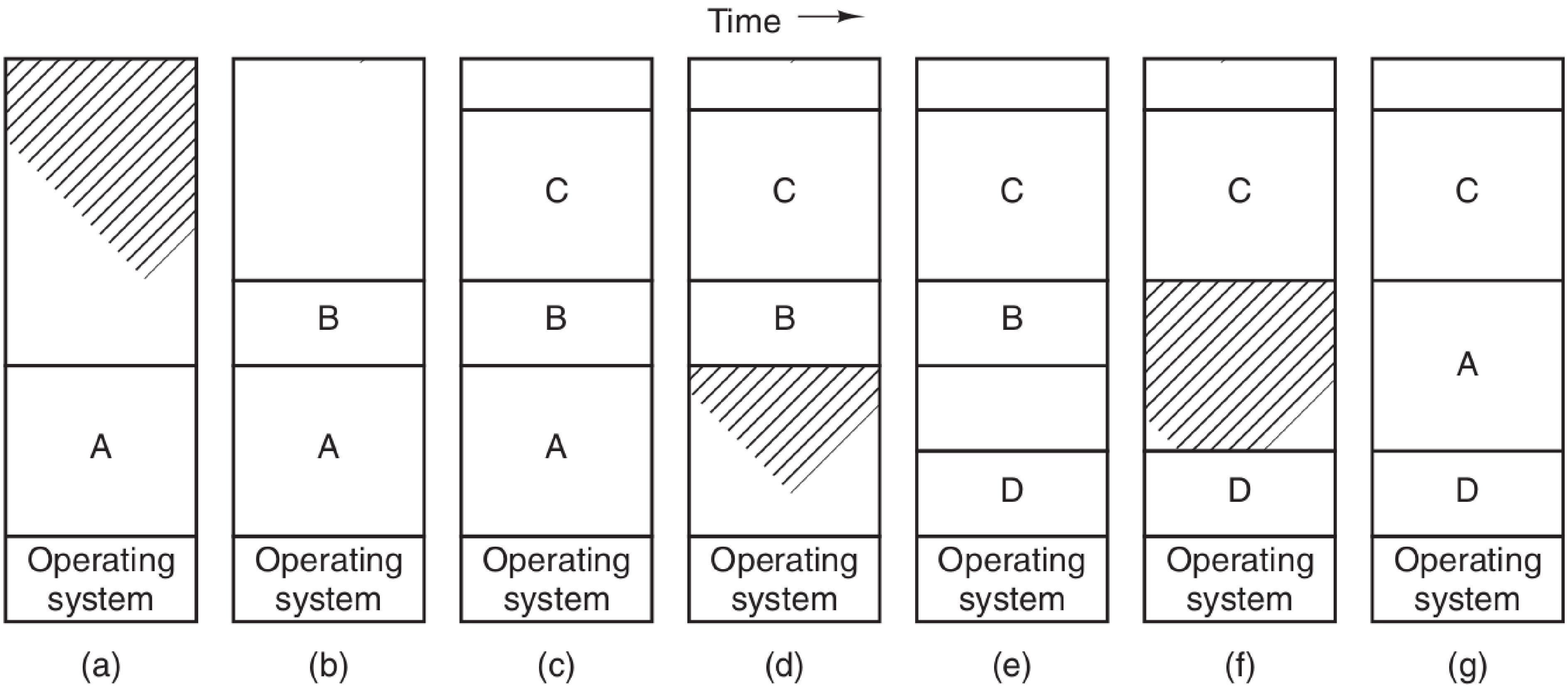


(c)

Not Enough Memory

- Typically, memory is smaller than the needs of all processes (Ex: about 50 – 100 processes start at startup).
- Solution:
  - Swapping
  - Virtual Memory

Swapping

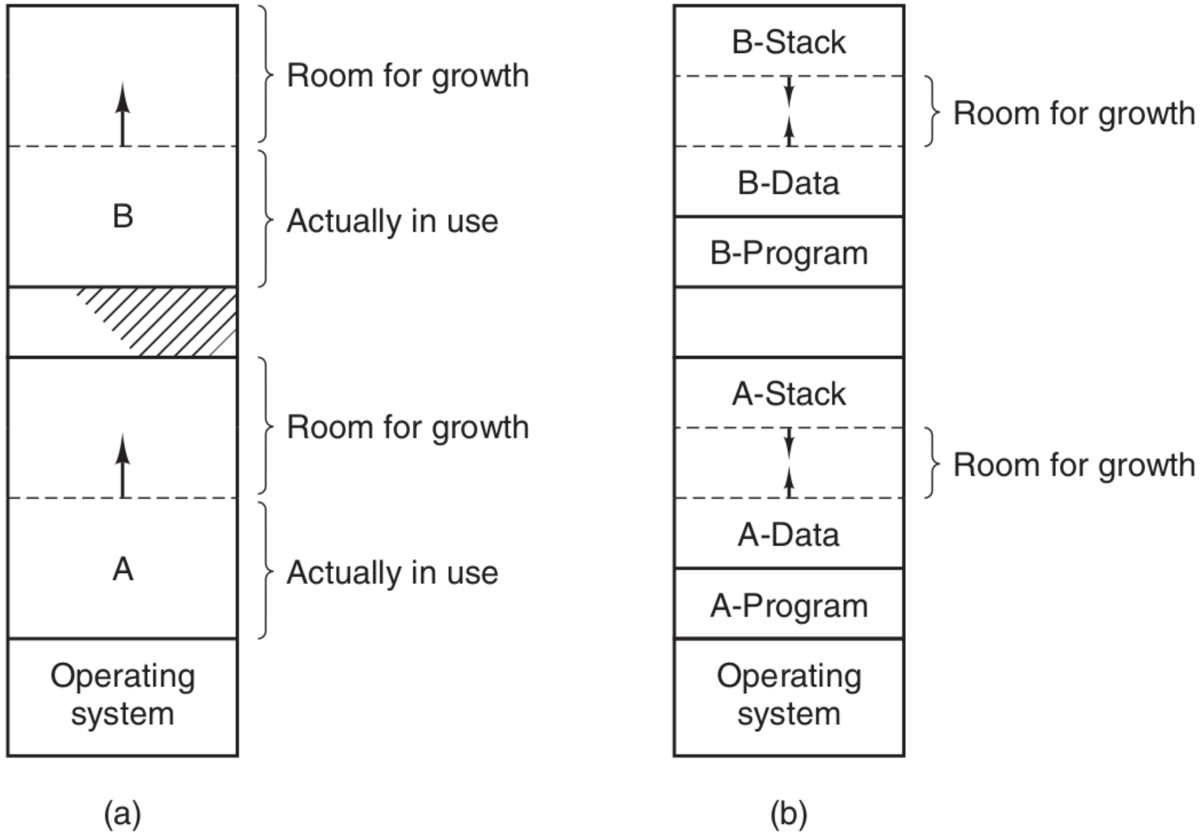


Time
C
C
C
C
C
B
B
B
B
A
A
A
A
A
D
D
D
Operating system
Operating system
Operating system
Operating system
Operating system
Operating system
Operating system
(a)
(b)
(c)
(d)
(e)
(f)
(g)

Swapping (cont.)

- When too many holes → memory fragmentation
  - **memory compaction**.
  - Takes a lot of time → not done often. (Ex: 16 seconds to copy 16GB at 8nsec/8 bytes.
- If process size is fixed that would be great.
- What happens when a process wants to grow?
  - Move, swap out or kill.
- Add extra space for growth at swapping in (not at swapping out).

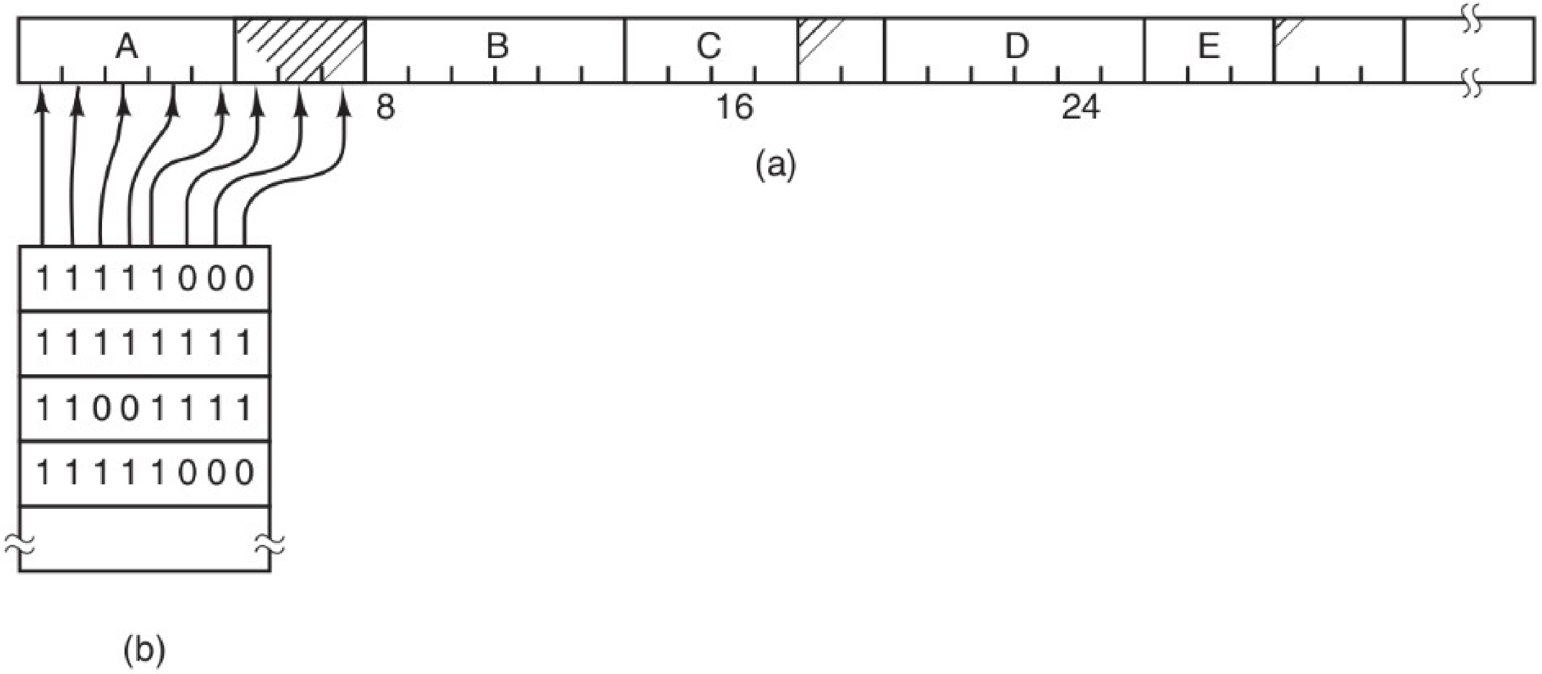
Swapping (cont.)



Managing Free Memory

- How to manage memory chunks during dynamic memory allocation?
  - Bitmap
  - Free lists

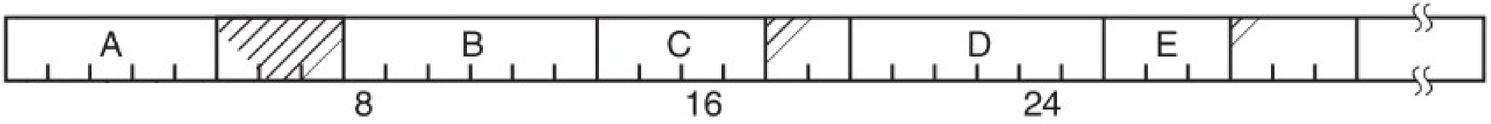
Memory Management with Bitmaps



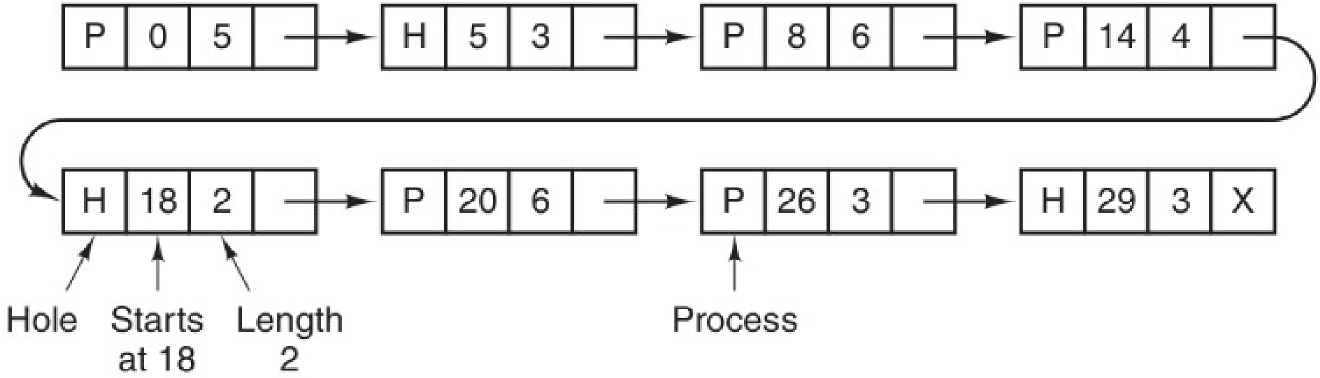
Memory Management with Bitmaps

- Memory is divided into units with size from few words to several kBs.
- For each unit, a bit is either 0 or 1.
- Smaller allocation unit → larger bitmap.
  - Ex: 4 bytes unit: 1 bit → 32n bits of memory: n bits. The map is 1/32 of memory.
- Larger allocation unit → possible waste in the last unit in process.
- Simple: map size depends on memory size and allocation unit size.
- To allocate a k-unit process: search the map for consecutive k 0 bits.
- Searching is slow: the run may stradle word boundaries.

Memory Management with Linked Lists



(a)



(c)

Memory Management with Linked Lists (cont.)

- Sorted by address. Easier when a process is terminated or swapped out.
- Double linked list is better, to find the previous.