OPERATING SYSTEMS

Memory Management

Suresh Jamadagni

Department of Computer Science

OPERATING SYSTEMS

Slides Credits for all the PPTs of this course

- The slides/diagrams in this course are an **adaptation**, **combination**, and **enhancement** of material from the following resources and persons:

1. Slides of Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin, Greg Gagne - 9th edition 2013 and some slides from 10th edition 2018
2. Some conceptual text and diagram from Operating Systems - Internals and Design Principles, William Stallings, 9th edition 2018
3. Some presentation transcripts from A. Frank – P. Weisberg
4. Some conceptual text from Operating Systems: Three Easy Pieces, Remzi Arpaci-Dusseau, Andrea Arpaci Dusseau

OPERATING SYSTEMS

Swapping, Memory Allocation , Fragmentation

Suresh Jamadagni

Department of Computer Science

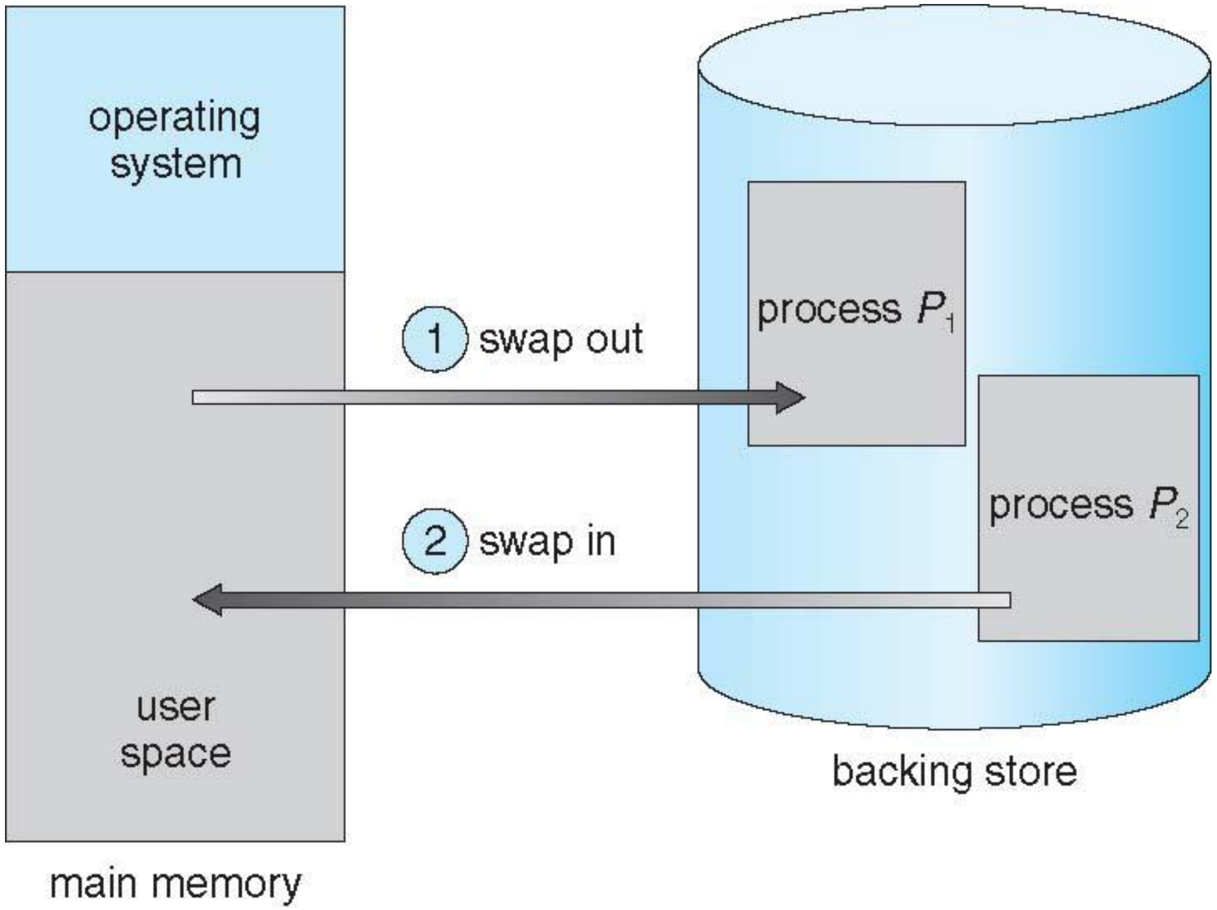
Swapping

- A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Swapping (Cont.)

- Does the swapped out process need to swap back in to same physical addresses?
- Depends on address binding method
  - Plus consider pending I/O to / from process memory space
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)
  - Swapping normally disabled
  - Started if more than threshold amount of memory allocated
  - Disabled again once memory demand reduced below threshold

Schematic View of Swapping



Context Switch Time including Swapping

- If next processes to be put on CPU is not in memory, need to swap out a process and swap in target process
- Context switch time can then be very high
- Let us consider a user process with size 100MB
- hard disk transfer rate of 50MB/sec
  - Swap out time=100MB/50MB per sec= 2 sec
  - Swap in in time is same as swap out time
  - Total context switch swapping component time = 4 seconds =swap in time + swap out time
- What will be the swap time for process with size 3GB? – 60 secs
- Is it right to swap complete process?
- Can be reduced if size of memory to be swapped is reduced – by knowing how much memory really being used
  - System calls to inform OS of memory use via request_memory() and release_memory()

Context Switch Time and Swapping (Cont.)

- Other constraints as well on swapping
  - Pending I/O – can’t swap out as I/O would occur to wrong process
  - Or always transfer I/O to kernel space, then to process memory space
    - Known as **double buffering**, adds overhead
- Standard swapping not used in modern operating systems
  - But modified version common
    - Swap only when free memory extremely low

Swapping on Mobile Systems

- Not typically supported
  - Flash memory is used rather than hard disks
    - Storage is a constraint for swapping
    - Limited number of writes are tolerated by flash memory
    - Poor throughput between flash memory and CPU on mobile platform
- Instead of swapping, other methods to free memory if memory is low
  - *iOS* asks apps to voluntarily relinquish allocated memory
  - Read-only data are removed system and are loaded if needed
  - Data that have been modified (such as the stack) are never removed.
  - Applications that fail to free up sufficient memory may be terminated by the operating system.

Swapping on Mobile Systems

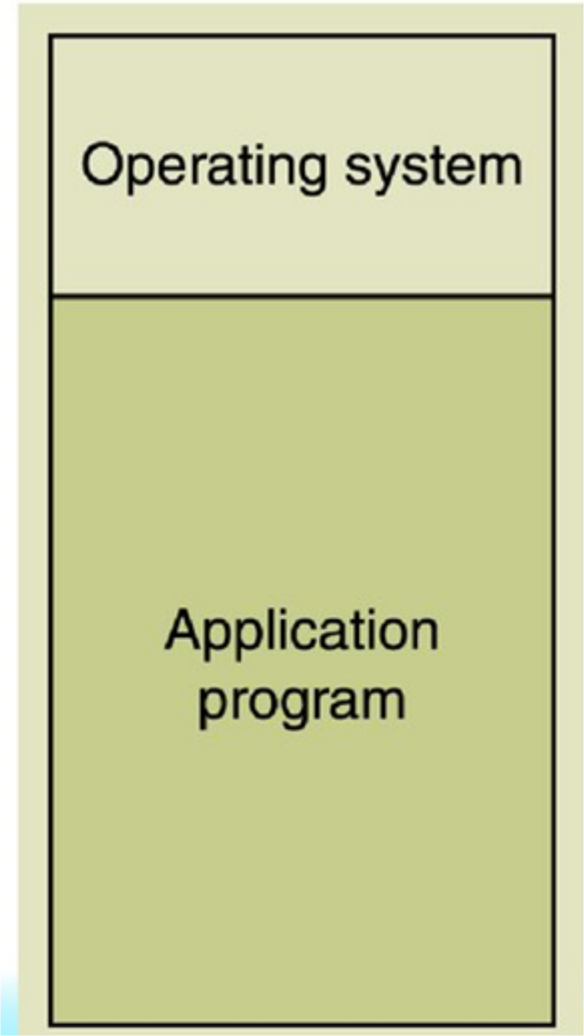
Android OS

- Does not support swapping
- ***Android*** terminates apps if low free memory, but first writes **application state** to flash for fast restart
- Both OSes support paging which will be discussed later

Contiguous Allocation

- Main memory must accommodate both OS and user processes
- Memory needs to be allocated 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 section of memory that is contiguous to the section containing the next process.

Contiguous Allocation



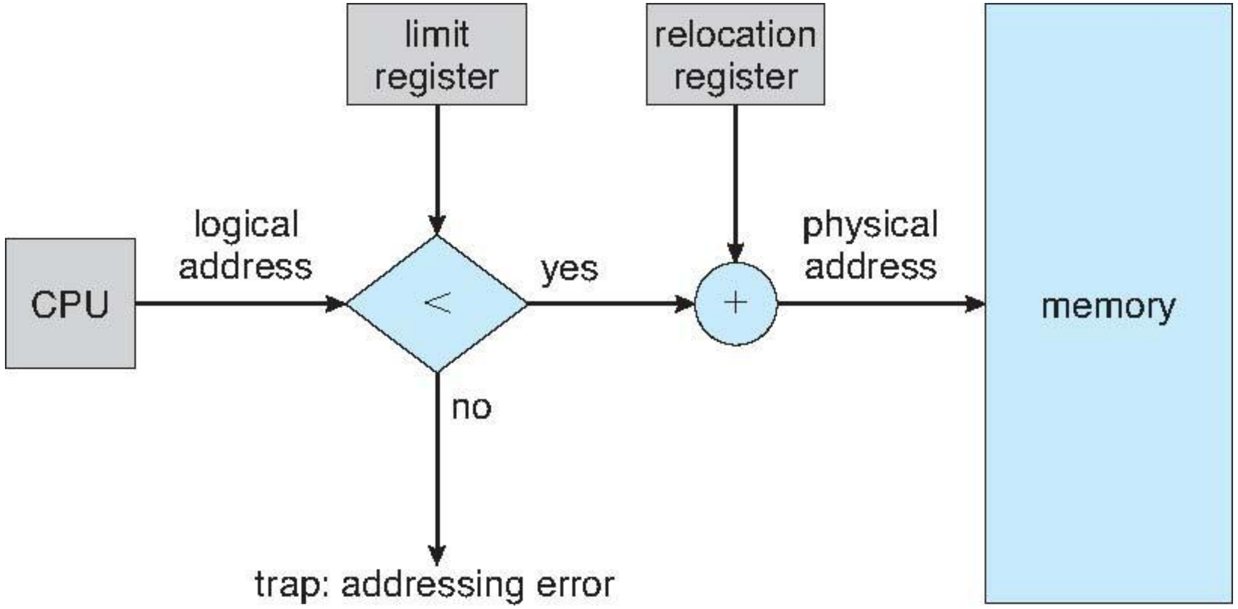
Lower address

higher address

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** (it comes and goes as needed) and kernel/OS changing size during program execution.

Hardware Support for Relocation and Limit Registers



When the CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with the correct values as part of the context switch.

Memory Protection

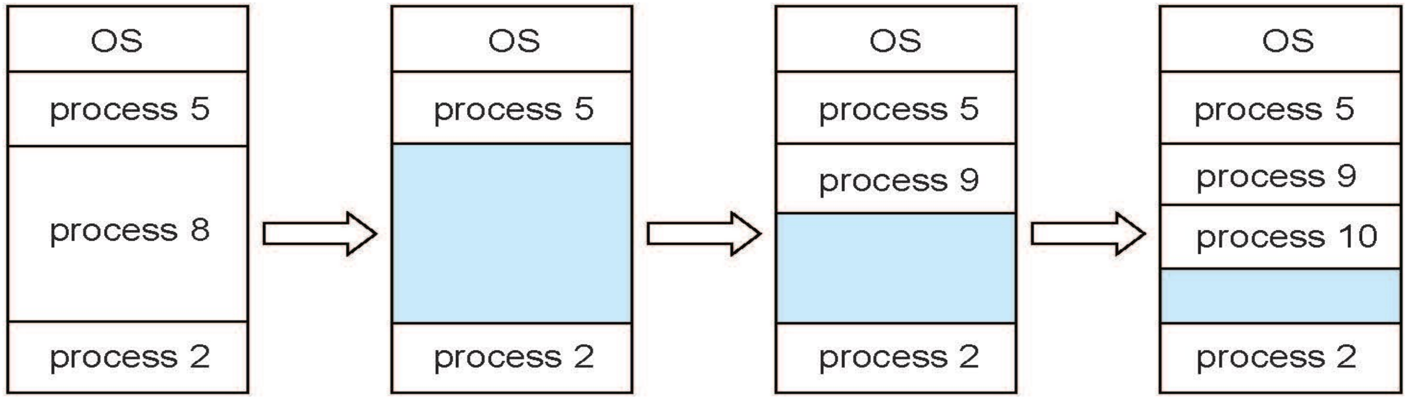
- Every address generated by a CPU is checked against these registers, so possible to protect both the operating system and the other users' programs and data from being modified by this running process .
- The relocation-register scheme provides an effective way to allow the operating system's size to change dynamically.
  - This flexibility is desirable in many situations. For example, the operating system contains **code** and **buffer space** for **device drivers**. If a device driver is not currently in use, it makes little sense to keep it in memory; instead, it can be loaded into memory only when it is needed. Likewise, when the device driver is no longer needed, it can be removed and its memory allocated for other needs.

Multiple-partition allocation

- Multiple-partition allocation
  - Divide memory into several **fixed-size partitions**.
    - Each partition may contain exactly one process
    - Degree of multiprogramming is bounded by number of partitions.
    - when a partition is free, a process is selected from the input queue and is loaded into the free partition
    - used by the IBM OS/360 operating system,but is no longer in use
  - **Variable-partition** sizes for efficiency (sized to a given process' needs)
    - This scheme keeps a table indicating which parts of memory are available/occupied
  - **Hole** – block of available memory; holes of various size are scattered throughout memory

Multiple-partition allocation (Cont.)

- 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)
- Considering the size of process as,
- Process 5=100MB,Processes 8=400MB, Process 2=200MB,Process 9=200MB, process 10=100MB, can the process 11=200MB be accommodated after process 5 terminates?



- **Advantages**

1. No Internal Fragmentation
2. No restriction on Degree of Multiprogramming
3. No Limitation on the size of the process

- **Disadvantages**
  - Causes External Fragmentation
  - Difficult to implement

Dynamic Storage-Allocation Problem and Strategies

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

- **First-fit**: Allocate the ***first*** hole that is big enough (generally faster)
- **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

Allocations strategies: Examples

PES UNIVERSITY ONLINE

Example: Given six memory partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB, and 125 KB (in order), how would the first-fit, best-fit, and worst-fit algorithms place processes of size 115 KB, 500 KB, 358 KB, 200 KB, and 375 KB (in order)? Rank the algorithms in terms of how efficiently they use memory.

Soln:

Partitions	First-Fit
M1=300 KB	P1
185KB	
M2=600 KB	P2
100KB	
M3=350 KB	P4
150KB	
M4=200 KB	
M5=750 KB	P3
392KB	P5
17KB	
M6=125 KB	

Partitions	Best Fit
M1=300 KB	
M2=600 KB	P2
100KB	
M3=350 KB	
M4=200 KB	P4
M5=750 KB	P3
392KB	P5
17KB	
M6=125 KB	P1
10KB	

Partitions	Worst-Fit
M1=300 KB	
M2=600 KB	P3
242KB	
M3=350 KB	P4
150KB	
M4=200 KB	
M5=750 KB	P1
635KB	P2
135KB	
M6=125 KB	

P5 must wait

Processes and their sizes

P1=115 KB P2=500 KB

P3=358 KB P4=200 KB

P5=375 KB

Fragmentation

- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used
- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
  - First fit statistical analysis reveals that given *N* allocated blocks (66% for example), another 0.5 *N* blocks (33% for example) lost to fragmentation
  - 1/3 of memory may be unusable -> **50-percent rule**
    - Unusable memory = (0.5N)/(N+0.5N)= 1/3

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

THANK YOU

Suresh Jamadagni
Department of Computer Science and Engineering
sureshjamadagni@pes.edu