

Memory Management

Carsten Griwodz
University of Oslo

(includes slides from A. Tanenbaum and M. van Steen)

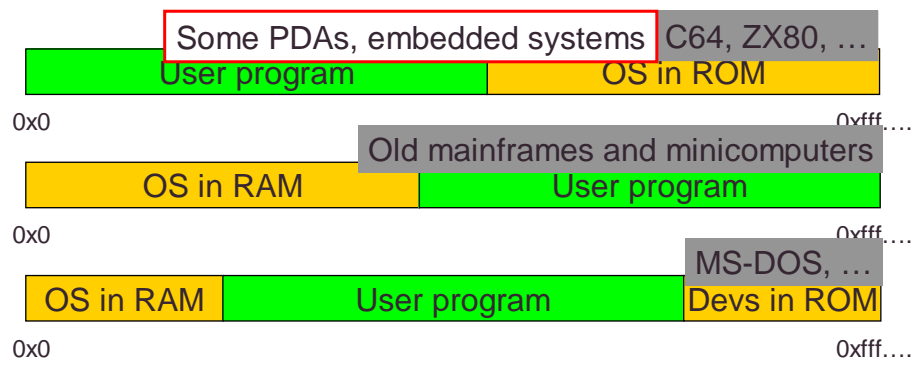
Motivation

- In project assignments so far
 - Program code is linked to kernel
 - Physical addresses are well-known
 - Not realistic

- In the real world
 - Programs are loaded dynamically
 - Physical addresses it will get are not known to program
 - Program size at run-time is not known to kernel

Memory Management for Monoprogramming

- Only one user program loaded
 - Program is entirely in memory
 - No swapping or paging
- Three simple ways of organizing memory



Multiprogramming

- Processes have to wait for I/O
- Goal
 - Do other work while a process waits
 - Give CPU to another process
- Processes may be concurrently ready
- So
 - If I/O waiting probability for all processes is p
 - Probable CPU utilization can be estimated as
$$CPU\ utilization = 1 - p^n$$

Multiprogramming

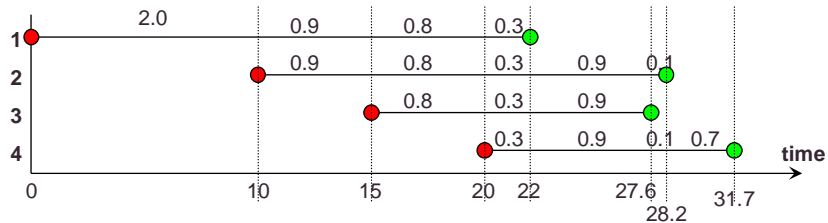
- Arrival and work requirements of 4 jobs
- CPU utilization for 1-4 jobs with 80% I/O wait

Job#	Arrival time	CPU use time	I/O wait time
1	10:00	4	16
2	10:10	3	12
3	10:15	2	8
4	10:20	2	8

Remaining CPU time

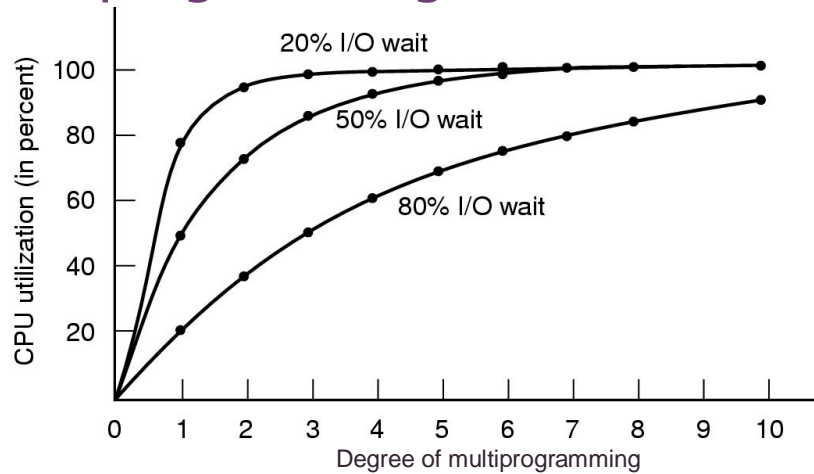
1	0.0
2	0.0
3	0.0
4	0.0

%	# processors			
	1	2	3	4
CPU idle	80	64	51	41
CPU busy	20	36	49	59
CPU per process	20	18	16	15



- Sequence of events as jobs arrive and finish
 - Note numbers show amount of CPU time jobs get each interval

Multiprogramming



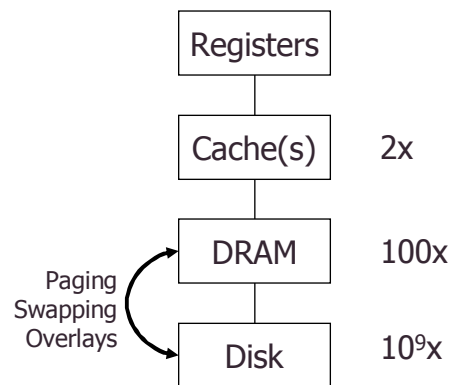
- CPU utilization as a function of number of processes in memory

Multiprogramming

- Several programs
 - Concurrently loaded into memory
 - OS must arrange memory sharing
 - Memory partitioning
- Memory
 - Needed for different tasks within a process
 - Shared among processes
 - Process memory demand may change over time
- Use of secondary storage
 - Move (parts of) blocking processes from memory
 - Higher degree of multiprogramming possible
 - Makes sense if processes block for long times

Memory Management for Multiprogramming

- Process may not be entirely into memory
- Reasons
 - Other processes use memory
 - Their turn
 - Higher priority
 - Process is waiting for I/O
 - Too big
 - For its share
 - For entire available memory
- Approaches
 - Swapping
 - Paging
 - Overlays



Memory Management for Multiprogramming

- Swapping
 - Remove a process from memory
 - With all of its state and data
 - Store it on a secondary medium
 - Disk, Flash RAM, other slow RAM, historically also Tape
- Paging
 - Remove part of a process from memory
 - Store it on a secondary medium
 - Sizes of such parts are fixed
 - Page size
- Overlays
 - Manually replace parts of code and data
 - Programmer's rather than OS's work
 - Only for very old and memory-scarce systems

How to use these
with
Virtual Memory

Memory Management Techniques

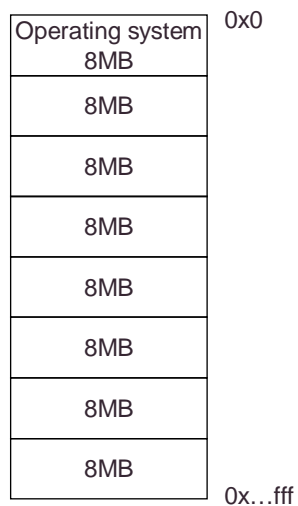
- Before details about moving processes out
 - Assign memory to processes
- Memory partitioning
 - Fixed partitioning
 - Dynamic partitioning
 - Simple paging
 - Simple segmentation
 - Virtual memory paging
 - Virtual memory segmentation

Fixed Partitioning

- Divide memory
 - Into static partitions
 - At system initialization time (boot or earlier)
- Advantages
 - Very easy to implement
 - Can support swapping process in and out

Fixed Partitioning

- Two fixed partitioning schemes
 - Equal-size partitions
 - Unequal-size partitions
- Equal-size partitions
 - Big programs can not be executed
 - Unless program parts are loaded from disk
 - Small programs use entire partition
 - A problem called "internal fragmentation"



Fixed Partitioning

- Two fixed partitioning schemes
 - Equal-size partitions
 - Unequal-size partitions

- Unequal-size partitions
 - Bigger programs can be loaded at once
 - Smaller programs can lead to less internal fragmentation
 - Advantages require assignment of jobs to partitions

Operating system
8MB
8MB
8MB
8MB
8MB
8MB
8MB
8MB
8MB
8MB

Operating system
8MB
2MB
4MB
6MB
8MB
8MB
12MB
16MB

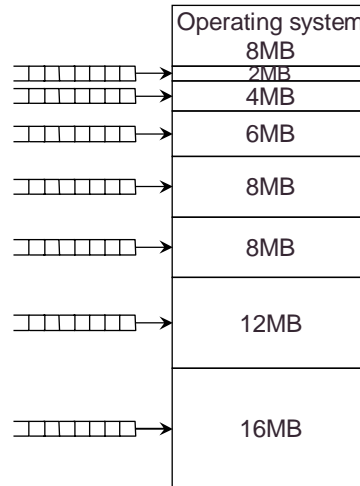
Fixed Partitioning

- Approach
 - Has been used in mainframes
 - Uses the term job for a running program
 - Jobs run as batch jobs
 - Jobs are taken from a queue of pending jobs
- Problem with unequal partitions
 - Choosing a job for a partition

Operating system
8MB
2MB
4MB
6MB
8MB
8MB
12MB
16MB

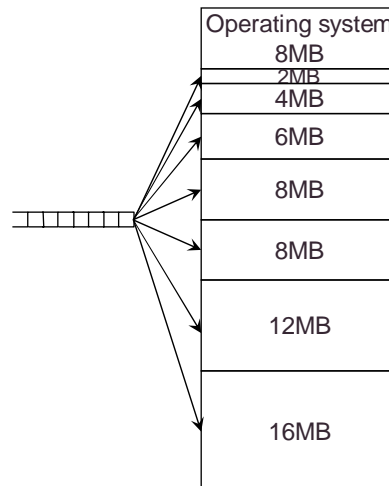
Fixed Partitioning

- One queue per partition
 - Internal fragmentation is minimal
 - Jobs wait although sufficiently large partitions are available



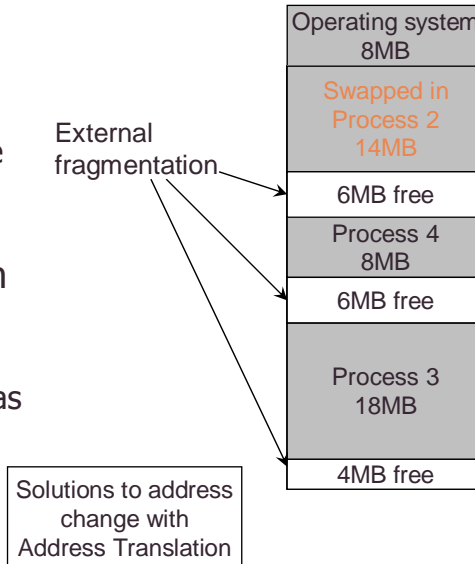
Fixed Partitioning

- Single queue
 - Jobs are put into next sufficiently large partition
 - Waiting time is reduced
 - Internal fragmentation is bigger
- A swapping mechanism can reduce internal fragmentation
 - Move a job to another partition



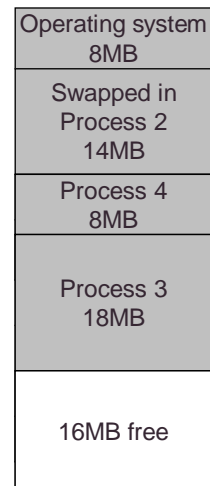
Dynamic Partitioning

- Divide memory
 - Partitions are created dynamically for jobs
 - Removed after jobs are finished
- External fragmentation
 - Problem increases with system running time
 - Occurs with swapping as well
 - Addresses of process 2 changed



Dynamic Partitioning

- Reduce external fragmentation
 - Compaction
- Compaction
 - Takes time
 - Consumes processing resources
- Reduce compaction need
 - Placement algorithms

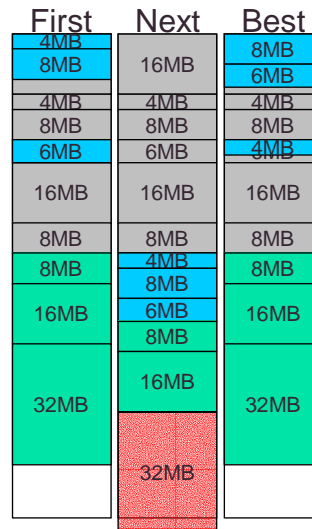


Dynamic Partitioning: Placement Algorithms

- Use most suitable partition for process

- Typical algorithms

- First fit
- Next fit
- Best fit



Dynamic Partitioning: Placement Algorithms

- Use most suitable partition for process

- Typical algorithms

- First fit
- Next fit
- Best fit

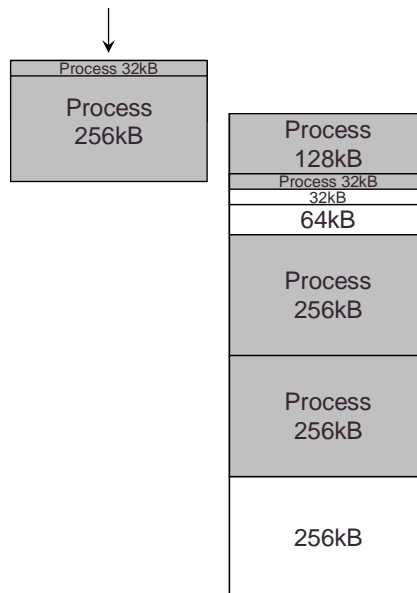


Dynamic Partitioning: Placement Algorithms

- Comparison of First fit, Next fit and Best fit
- Example is naturally artificial
 - First fit
 - Simplest, fastest of the three
 - Typically the best of the three
 - Next fit
 - Typically slightly worse than first fit
 - Problems with large segments
 - Best fit
 - Slowest
 - Creates lots of small free blocks
 - Therefore typically worst

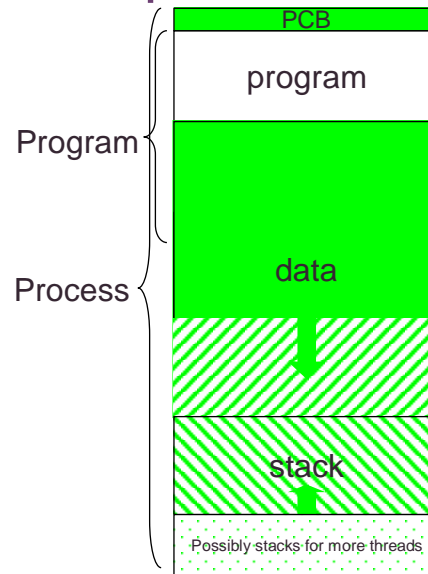
Buddy System

- Mix of fixed and dynamic partitioning
 - Partitions have sizes 2^k , $L \leq k \leq U$
- Maintain a list of holes with sizes
- Assign a process
 - Find smallest k so that process fits into 2^k
 - Find a hole of size 2^k
 - If not available, split smallest hole larger than 2^k
 - Split recursively into halves until two holes have size 2^k



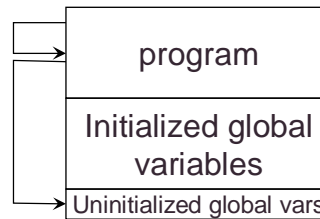
Memory use within a process

- Memory needs of known size
 - Program code
 - Global variables
- Memory needs of unknown size
 - Dynamically allocated memory
 - Stack
 - Several in multithreaded programs



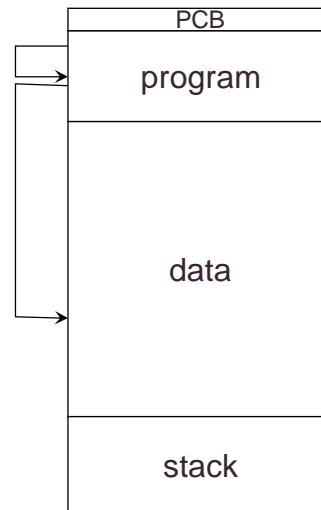
Memory Addressing

- Addressing in memory
 - Addressing needs are determined during programming
 - Must work independently of position in memory
 - Actual physical address are not known



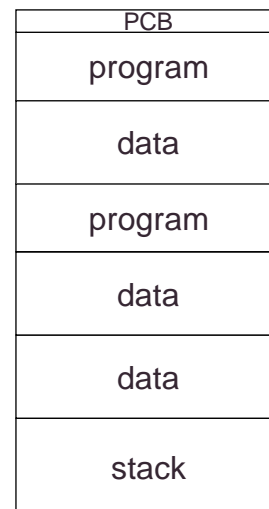
Memory Addressing

- Addressing in memory
 - Addressing needs are determined during programming
 - Must work independently of position in memory
 - Actual physical address are not known



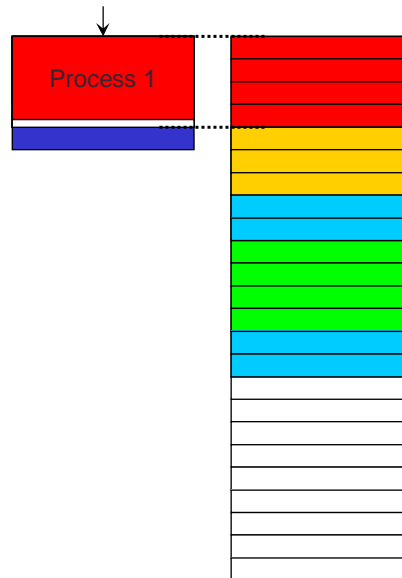
Memory Management

- Addressing
 - Covered address translation and virtual memory
- Important now
 - Translation is necessary
 - Therefore possible to have several parts
 - Pages
 - Segments



Paging

- Paging
 - Equal lengths
 - Determined by processor
 - One page moved into one memory frames
- Process is loaded into several frames
 - Not necessarily consecutive
- No external fragmentation
- Little internal fragmentation
 - Depends on frame size

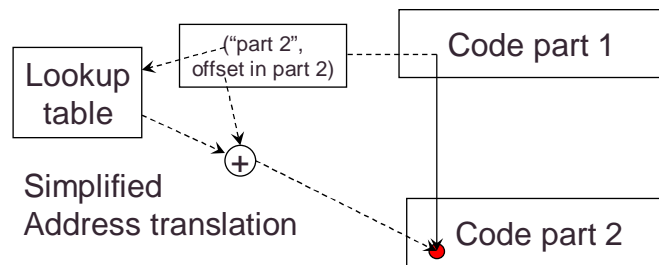


Segmentation

- Segmentation
 - Different lengths
 - Determined by programmer
 - Memory frames
- Programmer (or compiler toolchain) organizes program in parts
 - Move control
 - Needs awareness of possible segment size limits
- Pros and Cons
 - Principle as in dynamic partitioning
 - No internal fragmentation
 - Less external fragmentation because on average smaller segments

Paging and Segmentation

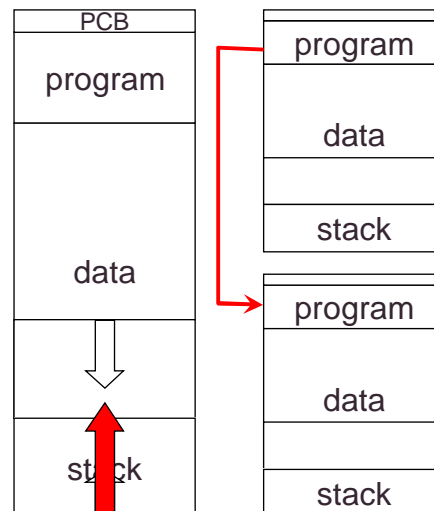
- Typical for paging and swapping
 - Address translation
 - At execution time
 - With processor support
- Simple paging and segmentation
 - Without virtual memory and protection
 - Can be implemented
 - by address rewriting at load time
 - by jump tables setup at load time



Other needs

- Protection of process from itself
 - (stack grows into heap)
- Protection of processes from each other
 - (write to other process)

Solutions to protection with Address Translation



Summary of Memory Management Algorithms

- Algorithms
 - Paging and segmentation
 - To be extended in address translation and virtual memory lectures
 - Placement algorithms for partitioning strategies
 - Mostly obsolete for system memory management
 - since hardware address translation is available
 - But still necessary for managing
 - kernel memory
 - memory within a process
 - memory of specialized systems (esp. databases)
- Address translation solves
 - Solves addressing in a loaded program
- Hardware address translation
 - Supports protection from data access
 - Supports new physical memory position after swapping in
- Virtual memory provides
 - Provide larger logical than physical memory
 - Selects process, page or segment for removal from physical memory