Slide 1

Memory Management

COMP3231 Operating Systems

2004 S2

Slide 2

PROCESS

- One or more threads of execution
- Resources required for execution
  - Memory (RAM)
    - program code ("text")
    - data (initialised, uninitialised, stack)
    - buffers etc held by kernel on behalf of process
  - others
    - CPU time
    - files, disk space
    - ...

Slide 3

MEMORY MANAGEMENT

➜ Subdividing memory to accommodate multiple concurrent processes
(*multiprogramming*, *multitasking*)

➜ Goals:

- Maximise memory utilisation
- Maximise processor utilisation
- Ensure minimum response time
- Ensure timely execution of "important" processes

➜ Conflicting goals ⇒ tradeoffs

Slide 4

MEMORY MANAGEMENT REQUIREMENTS

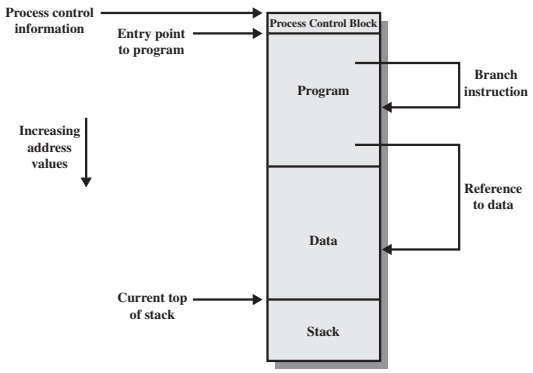
1. Address binding/relocation::

➜ Actual program location in memory unknown at the time the program is written

- Programs use various forms of symbolic references to data and instructions
- These must be bound to actual physical memory addresses
- Can happen:
  - at compile/link time,
  - at load time,
  - at run (execution) time.

Slide 5

Example logical address-space layout::



Slide 6

Compile/load-time binding::

- ➜ Can generate absolute addresses at compile/link time
  - Must recompile/relink code if starting address changes

Slide 7

Load-time binding::

- ➜ Compiler/linker generates relocatable addresses
- ➜ Loader replaces by absolute addresses once starting address is known
  - Must relink if starting address changes

Slide 8

Run-time binding::

- ➜ Compiler/linker/loader produce logical addresses
- ➜ Hardware translates addresses during execution
  - Allows dynamic relocation (moving) of program

Dynamic linking::

- ➜ Libraries not linked (copied) into executable file
- ➜ Libraries are linked to program at load time
- ➜ Library entry points are accessed via jump table initialised by dynamic linke
- ➜ Supports sharing of library code between programs

Slide 9

Dynamic loading::

Slide 10

- ➜ Library code is not loaded until actually invoked
- ➜ Entrypoint table initially points to dynamic loader
- ➜ After loading library, loader resets entrypoint addresses.

2. Protection::

Slide 11

- ➜ Processes should not be able to reference memory locations in another process without permission
  - Impossible to check absolute addresses in programs since the program could be relocated
  - Checks must be done at run-time
    - Requires hardware

3. Sharing::

Slide 12

- ➜ Allow several processes to access the same portion of memory
  - Shared code ⇒ better memory utilisation
  - Communication via shared data
- ➜ Selective sharing requires hardware support

Slide 13

4. Logical Organisation:

➜ Software engineering:
- Programs are written in modules
- Modules can be written and compiled independently
- Different degrees of protection given to modules (read-only, execute-only)
- Share modules

➜ Needs OS support

Slide 14

5. Physical Organisation:

➜ Memory available for a program plus its data may be insufficient
- Overlaying allows various modules to be assigned the same region of memory

➜ Programmer does not know how much space will be available
- Memory size of system?
- How many active processes?

➜ OS should abstract physical organisation

SIMPLE MM APPROACH: FIXED PARTITIONING

Slide 15

Equal-size partitions::

➜ Any process $\leq$ partition size can be loaded into any partition

➜ If all partitions are full, swap out some process

➜ A program may not fit in a partition.
- The programmer must design the program with overlays

➜ Any unused space within a partition is wasted:
- Called internal fragmentation

Slide 16

Unequal-size partitions::

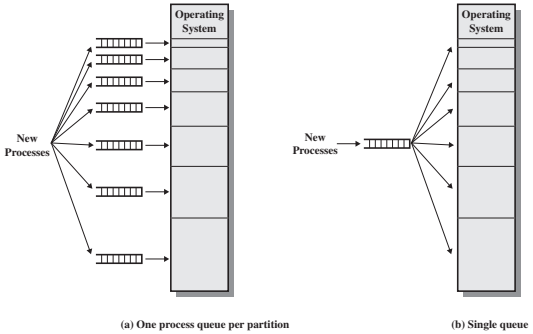
➜ Assign process to the smallest partition within which it will fit

➜ Reduces internal fragmentation

➜ May have contention for some partitions while others are unused
- reduces memory and CPU utilisation
- can allocate bigger partition (increases internal fragmentation)

Slide 17

Memory allocation for fixed partitioning::



Slide 18

Fixed partitioning summary:

- ➜ Simple
- ➜ Low CPU overhead
- ➜ Poor memory utilisation
- ➜ limits number of processes
- ➜ no support for
  - sharing
  - logical organisation
  - abstracting physical organisation

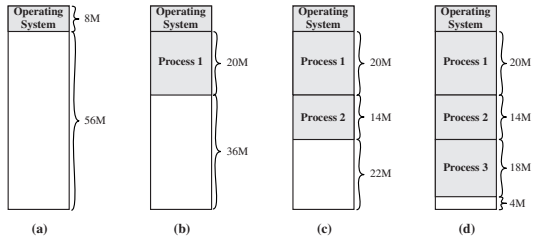
Slide 19

SIMPLE MM APPROACH: DYNAMIC PARTITIONING

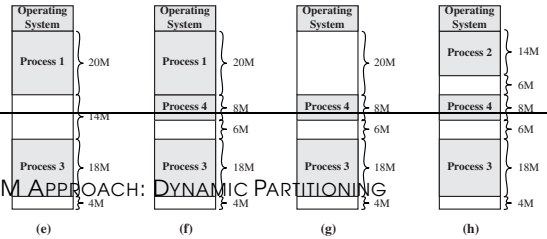
- ➜ Partitions are of variable length and number
- ➜ Process is allocated exactly as much memory as required
- ➜ Eventually get unusable holes in the memory.
  - Called external fragmentation
- ➜ Must use compaction to free up memory
  - shift processes so they are contiguous and all free memory is in one block

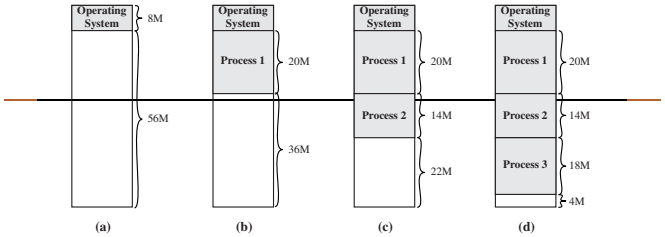
Slide 20

External fragmentation::

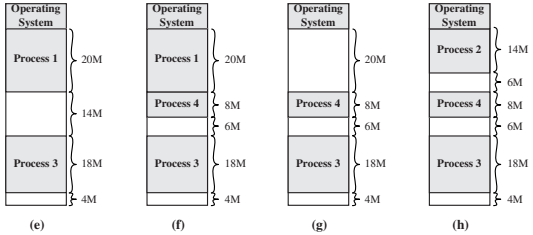


Now swap out process 2 to make space for process 4:





External fragmentation...:



Slide 21

Slide 22

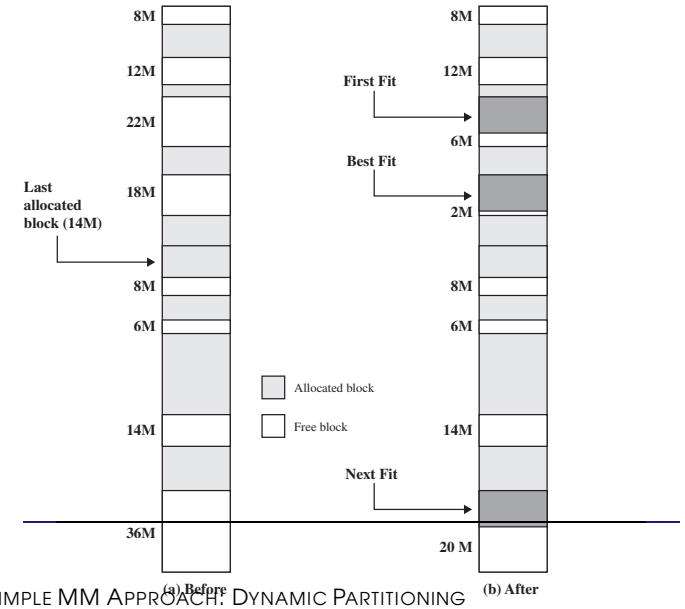
Dynamic Partitioning Placement Algorithms: OS must pick free block to allocate to a process

- Best-fit algorithm:
  - Chooses the block that is closest in size to the request
  - Maintain block list in size order
  - Leaves small fragments, unlikely to be useful
  - Tends to be slow
- First-fit algorithm:
  - Use first block big enough
  - Maintain block list in address order
  - May have to search frequently past same allocated blocks
- Next-fit algorithm:
  - Continue search from where last allocation was made

Slide 23

Example:: Allocate 16M

Slide 24



Slide 25

Dynamic partitioning: buddy system:

- ➜ Entire space available is treated as a single block of 2^U
- ➜ If a request of size s such that $2^{U-1} < s \leq 2^U$, entire block is allocated
- ➜ Otherwise:
  - Block is split into two equal buddies
  - Process continues until suitable size block of size 2^b is generated, so that $2^{b-1} < s \leq 2^b$
- ➜ Useful also for dynamic heap management (`malloc()`)

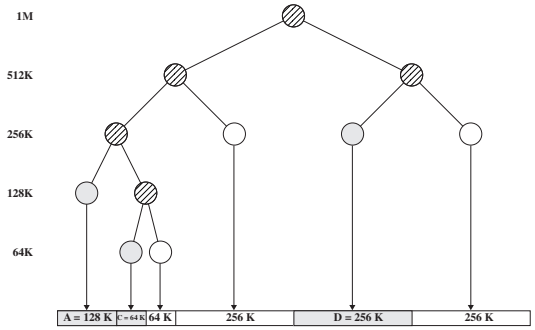
Slide 26

Buddy system example::

1 Mbyte block	1 M					
Request 100 K	A = 128 K	128 K	256 K	512 K		
Request 240 K	A = 128 K	128 K	B = 256 K	512 K		
Request 64 K	A = 128 K	C = 64 K	64 K	B = 256 K	512 K	
Request 256 K	A = 128 K	C = 64 K	64 K	B = 256 K	D = 256 K	256 K
Release B	A = 128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
Release A	128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
Request 75 K	E = 128 K	C = 64 K	64 K	256 K	D = 256 K	256 K
Release C	E = 128 K	128 K	256 K	D = 256 K	256 K	
Release E	512 K			D = 256 K	256 K	
Release D	1 M					

Slide 27

Buddy system representation::



Slide 28

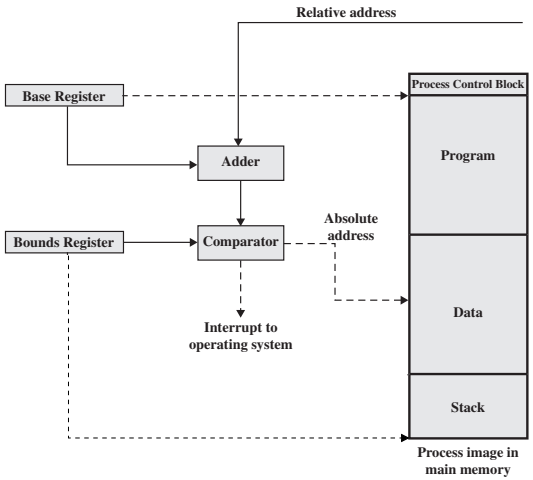
Relocation::

- ➜ Program uses logical (or virtual) addresses
- ➜ Actual (absolute or physical) addresses are determined at load time
- ➜ Addresses change at run time due to
  - swapping
  - compaction
- ➜ Requires address translation at run time (by hardware)
- ➜ This approach to memory management is called virtual memory

Slide 29

Minimal hardware support for relocation:

Slide 30



Slide 31

Registers used during execution::

- Base register
  - starting address for the process
  - added to logical address to obtain absolute address
- Limit (bounds) register
  - ending location of the process
  - compared to absolute address to detect address-range violation
- Set at load or relocation time
- Part of process context
- Implies contiguous allocation of physical memory
- Cannot support partial sharing of address spaces

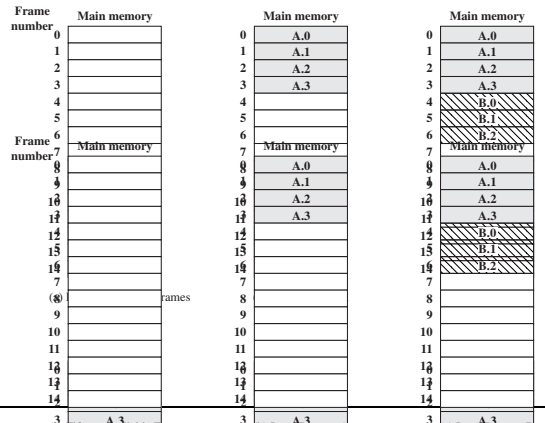
Slide 32

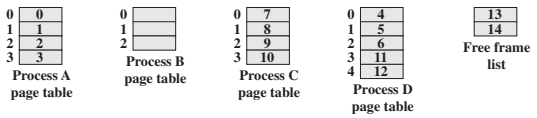
PAGING

- Partition physical memory into small equal-size chunks
  - called frames
- divide each process' (virtual) address space into the same size chunks
  - called pages
  - virtual memory address consist of page number and offset within the page
- OS maintains a page table for each process
  - contains the frame location for each page in the process
- Process' physical memory does **not** have to be contiguous

Slide 33

Page assignment::





Slide 34

Slide 35

Paging::

- ➜ No external fragmentation
- ➜ Small internal fragmentation
- ➜ Allows sharing by mapping several pages to the same frame
- ➜ Abstracts physical organisation
- ➜ Moderate support for logical organisation

Slide 36

SEGMENTATION

- ➜ Instead of equal-size pages use arbitrary-sized segments
- ➜ Address consist of two parts: segment number and offset
- ➜ Properties:
  - Supports sharing by mapping several segments to same PM
  - Supports logical organisation
  - Abstracts physical organisation
- ➜ Since segments are not equal get similar issues as with dynamic partitioning
  - no internal fragmentation
  - significant external fragmentation