Memory management

Paging:

- Memory is split up into partitions of blocks of fixed size
- Physical memory blocks are known as frames
- Fixed-sized logical memory blocks are known as pages
- A program is allocated a number of pages that is usually just larger than what was actually needed
- The logical address stores the page number in the first part of address and the page offset in the remaining part of the address.
  - Page number: number of bits required to represent the pages in Logical address space.
  - Page offset: Number of bits required to represent a particular `word` in page.

    > 如果一共有16个`pages`，每个page的大小是1024`words` 那么则需要address的前四个bits来表示page number, 后 10个bits 来表示page offset. 这样的话 通过address可以精准定位到某一个`page`的某一个`word`. e.g: 01010000000001代表第5个`page`的第2个`word`.

- `--`
- Page table: the page map table shows the mapping to pages to page frames
- Page: virtual memory divided into blocks of fixed size
- Page frame: the main memory is divided into page frames of same size as page

How paging is used to handle virtual memory:

- Divide main memory/RAM into frames
- Divide virtual memory into blocks of same size called pages
- Frame/pages are a fixed size
- Set up a page (map) table that translates logical to physical address
- Keeps track of all free frames
- Swap pages in memory with new pages from disk when needed

The page frame's address entry is 245, state what the values of 245 could represent

- The 245th page frame from the start of memory

Process X executes until the next instruction is the first instruction in page4. Page 4 is not current main memory

State a hardware device that could be storing this page

- Flash memory // magnetic disk // hard drive

Segmentation

- Logical address space is broken up into variable-size memory blocks called segments
- Segment can be of varying size
- Each segment has name and size
- During execution
    - Segments from the logical memory are loaded into physical memory
    - Address = segment number + offset value; found in segment map table
    - Offset value decides the size of the segment

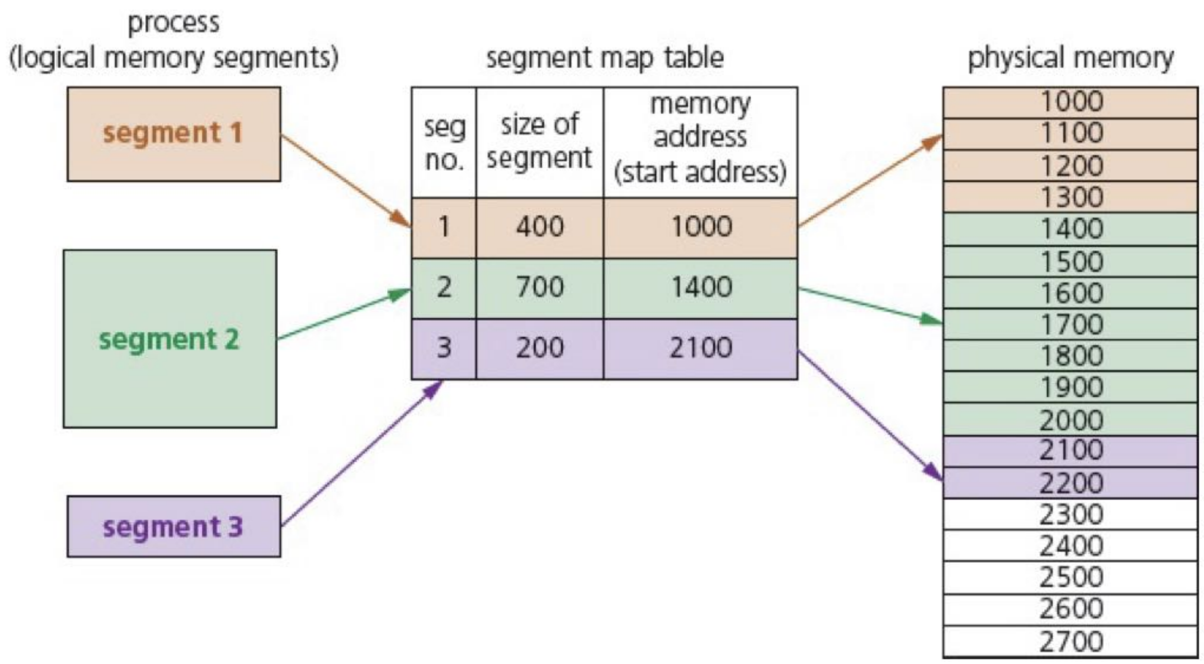

Figure 16.11

Differences between paging and segmentation

Paging	Segmentation
A page is a fixed-size block of memory	A segment is a variable-size block of memory
Since the block size is fixed, it is possible that call blocks may not be fully used - this can lead to internal fragmentation	Because memory blocks are a variable size, this reduces risk of internal fragmentation but increase the risk of external fragmentation.
The user provide a single value - this means that the hardware decides the actual page size	The user will supply the address in two values
A page table maps logical address to physical addresses(this contains the base address of each page stored in frame in physical memory space)	Segmentation uses a segment map table containing segment number + offset(segment size); it maps logical addresses to physical addresses
The process of paging is essentially invisible to the user/programmer	Segmentation is essentially a visible process to user/programmer
Procedures and any associated data cannot be separated when using paging	Procedures and any associated data can be separated when using segmentation
Paging consists of static link and dynamic loading	Segmentation consists of dynamic linking and dynamic loading
Pages are usually smaller than segments	Segments are usually larger than pages

Virtual memory

- If the available RAM is exceeded due to multiple processes running, it is possible to corrupt the data used in some of the program being run
- This can be solved by separately mapping each program's virtual memory space to RAM and utilizing the hard disk drive/solid state drive if we need more memory. This is the basis behind virtual memory.
- RAM is the physical memory and virtual memory is RAM+swap space on the hard disk
- Virtual memory is usually implemented using 'in demand paging'

State what is meant by virtual memory

- Disk/Secondary storage is used to extend the RAM/memory available
- … so CPU can access more memory space than available RAM
- Only part of the program/data in use needs to be in RAM
- Data is swapped between RAM and disk

Advantages of using virtual memory

- Not all pages needs to be in memory at one time
- More than one processes can be in ready state
- … because memory addresses can be mapped
- A process can have an address space larger than that of physical memory

Disk thrashing

The main drawback of using HDD in virtual memory is that as main memory fills, more and more data/pages need to be swapped in and out of virtual memory. This leads to high rate of hard disk read/write head movement. if **more time is spent on moving pages in and out of memory than actually doing any processing**, then the processing speed of the computer will be reduced. A point can be reached when the execution of a process comes to a halt since the system is so busy paging and in and out of memory; this is known as the thrash point.

- Pages are requested back into RAM as soon as they are moved to the disk
- There is continuous swapping (of the same page)
- No useful processing happens (deadlock)
  - Pages that are in RAM and disk are interdependent
  - All processing times are used for swapping pages

Page replacement

- Page replacement occurs when a requested page is not in memory. When paging in/out from memory, it is necessary to consider how the computer can decide which page to replace to allow the requested page to be loaded.
- When a new page is requested but is not in memory, a page fault occurs and the OS replaces one of the existing pages with the new page. How to decide which page to replace is done in a number of different ways, but all methods have the same goal of minimizing the number of page faults.

- A page fault is a type of interrupt raised by hardware. When a running program accesses a page that is mapped into virtual memory address space, but not yet loaded into physical memory, then the hardware needs to raise this page fault interrupt.

Page replacement algorithms:

- **First in first out (FIFO)**
  - The OS keeps track of all the pages in memory using a queue structure
  - The oldest page is at the front of the queue and is the first to be removed when a new page needs to be added
- FIFO algorithms don't consider page usage when replacing pages; a page may be replaced simply because it arrived earlier than another page
- It suffers from, what is known as, Belady's Anomaly where it is possible to have more page faults when increasing the number of page frames.
- Additional data required: time of entry
- Drawbacks:
  - Page in for lengthy period of time may be often accessed
  - … so not a good candidate to be removed
- **Optimal page replacement(OPR)**
  - Optimal page replacement looks forward in time to see which frame it can replace in the event of a page fault.
    - The algorithm is actually impossible to implement; at the time of a page fault, the OS has no way of knowing when each of the pages will be replaced next.
    - It tends to get used for comparison studies but has the advantage that it is free of Belady's Anomaly and also has the fewest page faults.
- **Least recently used page replacement (LRU)**
  - With least recently used page replacement (LRU), the page which has not been used for the longest time is replaced.
  - To implement this method, it is necessary to maintain a linked list of all pages in memory with the most recently used page at the front and the least recently used page at the rear.
- **Least used page replacement**
  - Additional data required: Number of times the page has been accessed
  - Drawbacks:

  - A page just entered has a low least-used value
  - … so likely to be a candidate for immediately being swapped out
- **Clock page replacement/second-change page replacement**