EECS 3221.3
Operating System Fundamentals

No. 10

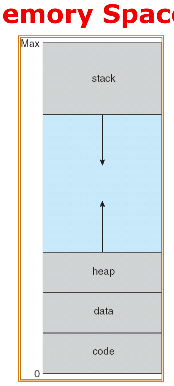
Virtual Memory

Prof. Hui Jiang
Department of Electrical Engineering and Computer Science, York University

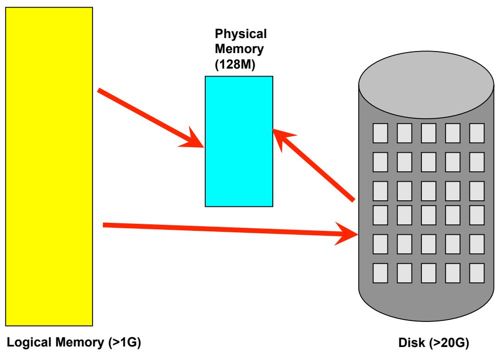
Background

- Memory-management methods normally requires the entire process to be in memory before the process can execute.
- Better not to load the whole process in memory for execution:
  – Programs often have code to handle unusual error conditions.
  – Arrays, lists, and tables are often allocated more memory than they actually need.
  – Certain options and features of a program may be used rarely.
  – Even all codes are needed, they may not all be needed at the same time.
- Our goal: partially load a process.
  – No longer be constrained by the amount of physical memory.
  – Each process takes less memory → CPU utilization and throughput up.
  – Less I/O to load program → run faster.

Logical Memory Space (review)



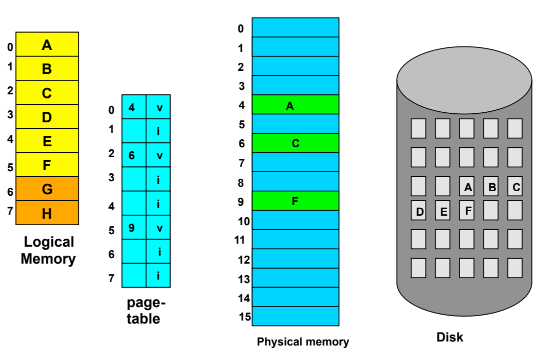
Virtual Memory: concept



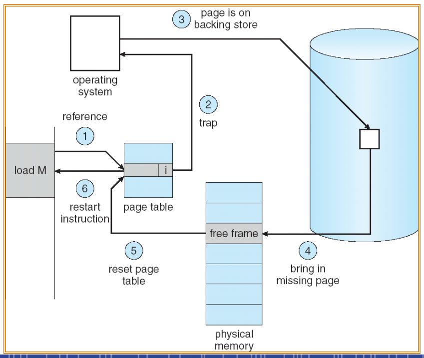
Demand Paging(1)

- Demand paging:
  - A paging system with a lazy page swapper.
  - A lazy swapper: never swap a page into memory unless the page will be used.
- In demand paging:
  - When a process is executed,
  - The pager guess which pages are needed. (optional)
  - The pager brings only these necessary pages into memory. (optional)
  - When referring a page not in a memory, the pager bring it in as needed and possibly replace an old page when no more free space.
- Hardware support: to distinguish those pages in memory and those pages in disk.
  - Use valid-invalid bit.

An example: Demand Paging



Handle a page fault



Handle a Page Fault

The interrupt handler program to handle page fault in virtual memory:

- Check an internal table to see if the reference was a valid or invalid memory access.
- If invalid, terminate the process; If valid, this page is on disk. Need page it into memory.
- Find a free frame from the free-frame list. (if no free frame, need replace an old page)
- Schedule a disk operation to read the desired page into the newly allocated frame.
- When the disk read is complete, modify the internal table and page table to set the bit as valid to indicate this page is now in memory.
- Restart the instruction that was interrupted. The process can now access the page as though it had always been in memory

Handle a Page Fault (more details)

- Trap to the OS
- Save the user registers and process status.
- Determine the interrupt was a page fault.
- Determine the location of the page on the disk.
- Find a free frame from the free-frame list.
  - If no free frame, page replacement.
- Issue a read from the disk to the free frame:
  - Wait in a queue for the disk until serviced.
  - Wait for the disk seek and latency time.
  - Begin the transfer of the page to the free frame.
- While waiting, allocate the CPU to other process.
- Interrupt from the disk (I/O completed).
- Save the registers and process state for other running process.
- Determine the interrupt was from the disk.

Handle a Page Fault (more details) (cont'd)

- …
- Correct the page table and other tables to show the desired page is now in memory.
- Wake up the original waiting process.
- Wait for the CPU to be allocated to this process again.
- Restore the user registers and process state and new page table.
- Resume the interrupted instruction.

Performance of Demand Paging

- To service a page fault is very time-consuming:
  - Service the page-fault interrupt.
  - Read in the page.
  - Restart the process.
- Effective access time for a demand-paged system:

Effective Access Time = (1-p) * ma + p * page fault time

- One example: memory access 100 nanosecond
  page fault 25 millisecond

Effective Access Time = 100 + 24,999,900 * p

If **p**=1/1000, **EAT** = 25 microsecond (slow down a factor of 250)
If requiring only 10% slow down, **p**<4/10,000,000 (one out of 2.5 million)

- How to achieve low page fault rate??

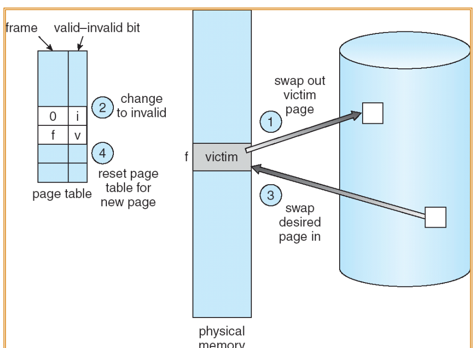
Page Replacement(1)

- **In demand paging, when increasing multiprogramming level, it is possible to run out of all free frames.**
- **How about if a page fault occurs when no free frame is available?**
  - **Stop the process.**
  - **Swap out another process to free some frames.**
  - **Page replacement:**
    - **Replacing in page level.**

Page Replacement(2)

- If no frame is free, find one frame that is not currently being used and free it.
  - Write the page into swap space and change page-table to indicate that this page is no longer in memory.
  - Use the freed frame to hold the page for which the process faulted.
- Use a page-replacement algorithm to select a victim frame.
- In this case, two disk accesses are required (one write one read).

Page Replacement

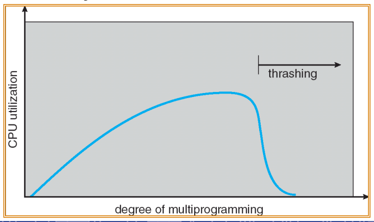


Page Replacement(3)

- Use a ***modify bit (dirty bit)*** to reduce overhead:
  - Each frame has a modify bit associated in hardware.
  - Any write in page will set this bit by hardware.
  - In page replacement, if the bit is not set, no need to write back to disk.
- For read-only pages, always no need to write back.
- With page replacement, we can run a large program in a small memory.
- Two important issues:
  - Page-replacement algorithm: how to select the frame to be replaced?
  - Frame-allocation algorithm: how many frames to allocate to each process?

Thrashing

- Thrashing: a process is spending a significant time in paging.
- Thrashing results in severe performance problem. The process is spending more time in paging than executing.
- Cause of thrashing:
  - The process is not allocated enough frames to hold all the pages currently in active use.



Other Considerations in demand-paging

- Page size: in powers of 2 (2**12 – 2*22)
  - Small page size → large page-table.
  - Small page size → less internal fragmentation.
  - Small page size → more I/O overhead in paging.
  - Small page size → more page-faults.
  - Small page size → less I/O amount (better resolution) and less total allocated memory.
  - A historical trend is toward larger page sizes.

Other Considerations in demand-paging

- Program structure: a careful selection of data structure and programming structure
  - To increase locality and hence lower the page-fault rate.
  - To reduce total number of memory access.
  - To reduce total number of pages touched.
- Also compiler and loader can improve.
- Example: Array `A[1024][1024]` of integer
  - Each row is stored in one page
  - Program 1

```
for j = 1 to 1024 do
    for i = 1 to 1024 do
        A[i][j] = 0;
```

    1024 x 1024 page faults

  - Program 2

```
for i = 1 to 1024 do
    for j = 1 to 1024 do
        A[i][j] = 0;
```

    1024 page faults