Paging: Design Issues
Readings

- Silbershatz et al: 9.5-9.6
Outline

- Frame Allocation
- Case Studies
Thrashing

- A process causing page faults every few instructions is said to be **thrashing**
- If a process does not have “enough” pages, the page-fault rate is very high. This leads to:
  - low CPU utilization;
  - operating system thinks that it needs to increase the degree of multiprogramming;
  - another process added to the system;

- To minimize thrashing requires that we carefully consider issues related to frame allocation
Frame Allocation

- How do we allocate the fixed amount of memory among the various processes?
- What is the minimum number of frames?
- How many frames should a process get?
- Should a process replace a page being used by another process?
Minimum Number of Frames

- Each process needs a minimum number of pages
- Example: 6 pages needed by one instruction
  - Instruction might span 2 pages
  - 1st argument spans 2 pages
  - 2nd argument spans 2 pages
- We must have enough frames to hold all the different pages that any single instruction can reference
- The minimum number of frames is defined by the computer architecture
Global vs Local Allocation

- We have discussed several algorithms for choosing a page to replace when a page fault occurs.

- Should a process only be allowed to replace any page or only its pages?
  - Global vs Local
Global vs. Local Allocation

- **Global replacement**
  - Process selects a replacement frame from the set of all frames
  - A process can take a frame from another based on some priority scheme
  - Allows for the number of frames assigned to a process to dynamically vary
  - A process cannot control its own page-fault rate
Global vs. Local Allocation

- **Local replacement**
  - Each process selects from only its own set of allocated frames
  - Corresponds to allocating every process a fixed fraction of memory
  - This assumes a fixed allocation of frames to a process
  - A process that thrashes cannot cause another process to thrash by stealing its frames
Global vs. Local Allocation

- Which is better global or local?
- On the surface it may seem that local is better since thrashing problems can’t spread but ..... 
  - I/O Competition:
    - Other thrashing processes
    - Processes that may page fault but are not necessarily thrashing
  - This implies a average longer wait for the disk read
Global vs. Local Allocation

- The number of pages actually needed (working set) varies over time

- Global usually means better utilization
  - If the working set shrinks local algorithms waste memory

- Global algorithms are usually used but some operating systems use a mix
Global Allocation

- If a global algorithm is used the operating system must decide how many frames to assign to each process
  - Fixed vs proportional
**Fixed vs Proportional Allocation**

- All processes are started with the same number of pages *(fixed)*
- Does it really make sense to start a small process with the same number as a large process
  - Allocate based on process size *(proportional)*
Other Considerations

- Prepaging
  - A page fault occur when a page is not found in memory
  - You can bring that page in as well as adjacent pages
  - In reading from disk most of the overhead is in waiting for the disk to be in a position to read the data
  - The actual transfer of data is relatively slow
  - Why not then read several pages into memory?
Other Considerations

- **Copy-on-Write**
  - Remember that the `fork()` system call works by creating a copy of the parent’s address space for the child
  - You can allow the parent and the child to initially share the same pages
  - When a child needs a page then a copy is made.
Other Considerations

- I/O Interlock
  - Pages must sometimes be locked into memory
  - Consider I/O - Pages that are used for copying a file from a device must be locked from being selected for eviction by a page replacement algorithm
Case Study – Windows

- Virtual memory with demand page
- Can support 32 or 64 bit
- Has a pool of free frames
- Uses prepagging (called clustering)
- What happens if the amount of free memory falls below some threshold?
  - Each process has a minimum number of processes
  - Windows will take away pages that exceed that minimum
- Applies LRU Locally
Case Study - Window

- Each process is guaranteed to have a minimum number of frames.
- Each process has a maximum number of frames.
- If a page fault occurs for a process that has the maximum number of frames a local replacement policy is used.
- If a page fault occurs for a process that is below its working set maximum a free frame is used.
Case Study - Solaris

- Virtual memory with demand paging
- Can support 32 or 64 bit
- Global replacement
Case Study - Solaris

- Maintains a list of free frames
- A free frame is assigned to a process that page faults.
- If the number of free pages falls below a threshold value (lotsfree) a process called pageout starts.

**pageout process**
- Scans all pages and sets a reference bit to 0
- Later it checks the pages to determine if it has been written to.
- If not it is freed.

Pages that are shared are not freed.
Case Study - Linux

- Virtual memory with demand paging
- Can support 32 or 64 bit
- Global replacement

Replacement
  - Least recently used (LRU) policy
  - Different implementations for different systems
Case Study-Android, IoS

- PCs and Servers: Support some form of swapping
- Mobile devices -- Rely a lot on flash memory for persistent storage
  - It's fast
  - Flash memory can tolerate a limited number of writes before it becomes unreliable
- Support
  - Typically no swapping
  - Paged systems
Case Study-Android

- No swap space for memory
- Does have
  - Paging
  - Map pages to physical pages
- Implications
  - Modified data (e.g., stack is not removed)
  - Read-only data (e.g., code) can be removed from the system and reloaded from flash memory
Case Study -- Android

Sharing Memory

- Each app is forked from an existing process called Zygote
  - Zygote loads framework code
  - RAM pages allocated for framework code is shared by application processes

- Static data (e.g., code) is often mapped to specific pages

- Some dynamic memory is explicitly shared by Android and applications
  - Example: Window surfaces use shared memory between the app and screen compositor
Case Study - Android

- Switching applications
  - Android keeps processes that are not hosting a foreground ("user visible") app component in a least-recently-used (LRU) cache.
  - The system keeps the process cached, so if the user later returns to the app, the process is reused for faster app switching.
  - As the system runs low on memory, it may kill processes in the LRU cache beginning with the process least recently used.
Case Study - Android

- Implications
  - Developers must carefully allocate and release memory to ensure that their applications do not use too much memory or suffer from memory leaks
Summary

- This section studied how page tables are implemented
- Case Studies are presented