

No. 10

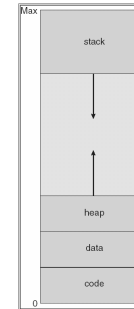
Virtual Memory

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

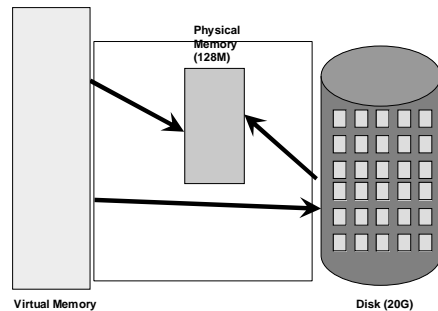
Background

- Memory-management methods 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 program takes less memory → CPU utilization and throughput up.
 - Less I/O to load program → run faster.
- Overlay and dynamic loading can ease the restriction, but require extra work by the programmer.

Logical Memory Space (review)



Virtual Memory: concepts



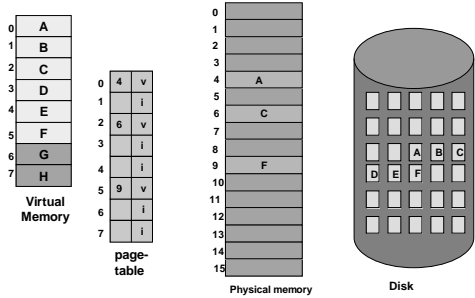
Virtual Memory

- Virtual memory can be implemented via:
 - Demand paging
 - Demand segmentation
 - Hard since segments have variable size

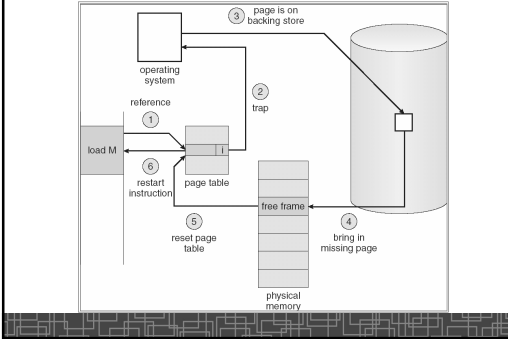
Demand Paging(1)

- Demand paging:
 - A paging system with 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 service routine 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)

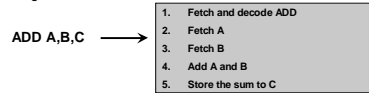
- ...
- Correct the page table and other tables to show the desired page is now in memory.
- Wake up the original waiting process.
- Wait for the CPU to be allocated to this process again.
- Restore the user registers and process state and new page table.
- Resume the interrupted instruction.

Pure Demand Paging

- Never bring a page into memory until it is referred.
- Start executing a process with no pages in memory
- OS set instruction pointer to the first instruction
- Once run, it causes a page fault to load the first page
- Faulting as necessary until every page is in memory

Some Architecture Concerns in demand paging

- Straightforward in most cases:



- But some instructions which may modify something are not easy to handle:
 - IBM 360/370: MVC (move 256 bytes)
 - PDP-11: auto-decrement or auto-increment addressing mode
`mov (R2)++, -(R3)`

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:

$$\text{Effective Access Time} = (1-p) * ma + p * \text{page fault time}$$

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

$$\text{Effective Access Time} = 100 + 24,999,900 * p$$

If $p=1/1000$, $EAT = 25$ microsecond (slow down a factor of 250)
If requiring only 10% slow down, $p < 4/10000000$ (one out of 2.5 million)

- How to achieve low page fault rate??

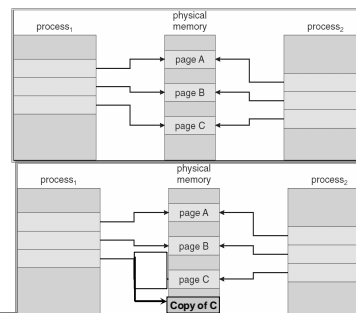
Handling Swap Space on Disk

- For fast speed:
 - Use swap space, not file system
 - Swap space: in larger blocks, no file lookup and indirect allocation.
 - Copying an entire file image into swap space at process startup and then perform demand paging from the swap space.
 - First load pages by file system, then write to swap space.

Copy-on-Write

- For quick process Creation: `fork()`
- Traditionally, `fork()` copies parent's address space for the child.
- **Copy-on-Write**: without copying, the parent and child process initially share the same pages, and these pages are marked as copy-on-write.
 - If either process needs to write to a shared page, a copy of the shared page is created and stop sharing this page.
- Advantages of copy-on-write:
 - Quick process creation (no copying, just modify page table for page sharing)
 - Eventually, only modified pages are copied. All non-modified pages are still shared by the parent and child processes.
 - Better memory utilization

Copy-on-Write



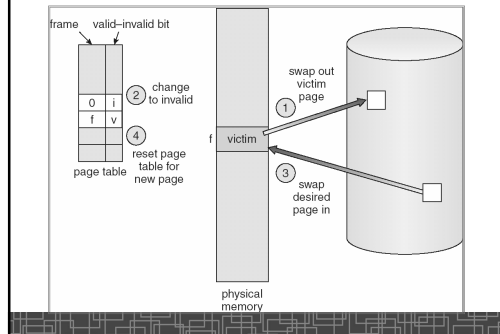
Page Replacement(1)

- 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 frames are available
 - Stop the process
 - Swap out a process to free some frames
 - Page replacement
 - Replacing in page level

Page Replacement(2)

- If no frame is free, find one 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).
- Use a *modify bit* to reduce overhead:
 - Each frame has a modify bit associated in hardware.
 - Any write in page will set the 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.
- Page-replacement algorithm: how to select the frame to be replaced
- Frame-allocation algorithm: how many frames to allocate to each process

Page Replacement



Page-Replacement Algorithm

- To achieve the lowest page-fault rate.
- Common schemes:
 - Optimal page replacement.
 - FIFO page replacement.
 - LRU page replacement.
 - LRU approximation page replacement.
 - Additional-reference-bits algorithm
 - Second-chance page-replacement algorithm
 - Counting-based page replacement.
 - Page-buffering algorithm.
- Evaluate with a reference string:
 - e.g., 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Optimal Page-Replacement

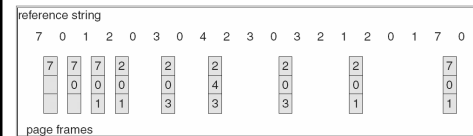
- An optimal page-replacement has the lowest page-fault rate among all possible replacement algorithms.
- OPT: replace the page that will not be used for the longest period of time.
- Guarantee the lowest page-fault rate
- Not feasible since future knowledge is required.
- Used for performance comparison of algorithms.

Reference String: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
4-frame example:

1	4
2	
3	
4	5

6 page faults

Optimal Page Replacement



FIFO (first-in-first-out) Page-Replacement

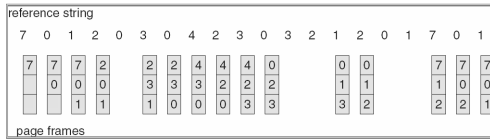
- Always replace the oldest page in memory
- Implement FIFO queue to hold all pages in memory. Replace the page at the head. When a page is brought into memory, it is inserted at the tail of the queue.
- Simple and easy to implement.
- Performance is not always good.
 - The replaced page may be a heavily used one
 - increasing page-fault rate

Reference String: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
4-frame example:

1	1	5	4
2	2	1	5
3	3	2	
4	4	3	

10 page faults

FIFO Page Replacement



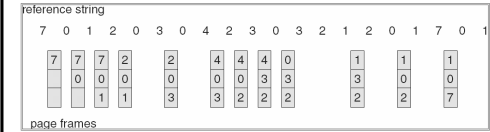
Least-recently-used (LRU) Page Replacement (1)

- Replace the page that has not been used for the longest period of time
- LRU has to associate with each page the time of last use.
- LRU chooses the oldest page based on the time stamp for replacement.
- The performance of LRU is considered to be good.
- Example: 4-frame
 - Reference String: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	5
2	
3	5 4
4	3

8 page-faults

LRU Page Replacement



LRU Page Replacement (2)

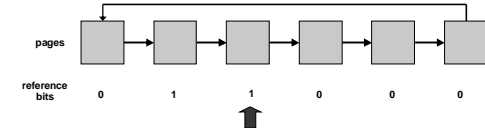
- LRU implementations
 - Counters:
 - CPU maintains a memory reference counter.
 - Add time-of-use in each entry in page-table.
 - Search the oldest page based on time-of-use.
 - Stack:
 - Keep a stack of all page numbers.
 - When one page is referenced, it is moved to the stack top.
 - The stack tail is always the LRU page.
- LRU implementation with hardware is very expensive.
- Few computers provide sufficient hardware for true LRU.

LRU Approximation Replacement(1)

- Reference bit:
 - Initially cleared by OS.
 - set by the hardware whenever the page is referenced.
- Additional-reference-bits algorithm:
 - We gain additional ordering information by recording the reference bits at regular intervals.
 - Keep an 8-bit byte for each page in memory
 - A timer interrupts at regular intervals (every 100 milliseconds)
 - Shift all bits right 1 bit and discard the low-order bit
 - OS copies the reference bit into the high-order bit and clear reference bit
 - Interpret the 8-bit byte as unsigned integer, the page with the lowest number is the LRU page.

LRU Approximation Replacement(2): Second-Chance Algorithm (clock)

- Based on FIFO policy, but check the reference bit of the selected page.
- If reference bit is 0, the page is replaced.
- If reference bit is set to 1, the page is given the second chance
 - The reference bit is cleared.
 - Its arrival time is reset to the current time.
- Second-chance (clock) algorithm can be implemented as a circular queue:



Other Replacement Algorithms

- Counting-based page replacement:
 - Keep a counter of the number of references made to each page.
 - The *least frequently used (LFU)* page-replacement: replace the page with the smallest count.
 - The *most frequently used (MFU)* page-replacement algorithm (the page with small count was just brought in and has yet to be used).
- Page-Buffering Algorithm:
 - Keep a pool of free frames.
 - Select a victim frame, but the desired page is read into one free frame in the pool without waiting for write-out. The victim is written out later on and is added to free pool.
 - Remember which page was in each frame of free pool. When a page is needed, check if it is in the free pool.

Frame Allocation

- In single-user system, user process compete free frames with OS
- In multi-programming system, how to allocate the fixed amount of free memory among various processes??
- Minimum number of frames: a minimum number of frames must be allocated to the process (depending on instruction-set architecture)
- Allocation algorithms:
 - Equal allocation: free frames are equally allocated to all processes.
 - Proportional allocation: allocate available frames to each process according to its size, its priority, or a combination.
- Global versus local allocation in replacement
 - Global allocation*: allow a process to select a replacement frame from the set of all frames. (can take frames from others)
 - Local allocation*: require a process to select from only its own set of allocated frames.

Example: proportional frame allocation

s_i = size of process p_i

$S = \sum s_i$

m = total number of frames

a_i = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 64$

$s_1 = 10$

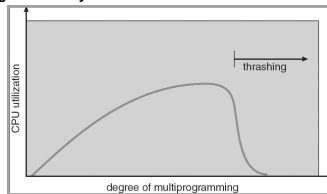
$s_2 = 127$

$a_1 = \frac{10}{137} \times 64 = 5$

$a_2 = \frac{127}{137} \times 64 = 59$

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.



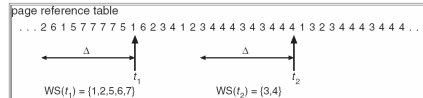
Locality Model of Programs

- A locality is a set of pages that are currently in an active use.
- A process moves from locality to locality.
- A program is generally composed of several different localities.
- The localities are defined by the program structure and its data structures.
- Locality model is the basic principle for caching as well as demand paging.
 - We only need a small number of frames to hold all pages in the current locality in order to avoid further page faults.

Working-set Model

- The model define a working-set window, say Δ page references, e.g., 10,000 page references.
- The set of all referenced pages in the most recent Δ page references is the working set.
- How to choose the window ?
 - if Δ too small will not encompass entire locality.
 - if Δ too large will encompass several localities.
 - if $\Delta = \infty \Rightarrow$ will encompass entire program.
- If WSS_i = working-set size of process P_i
 $\rightarrow D = \sum WSS_i$ total demand frames
- if $D > m$ (m : total available frames) \Rightarrow Thrashing.
- Policy:
 - CPU monitors working sets of all processes and allocate enough frames for the current working set.
 - if $D > m$, then suspend one of the processes.

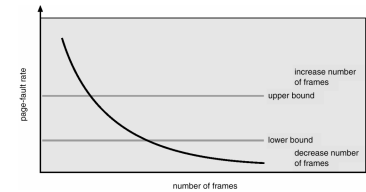
Working-Set Model



Keeping Track of the Working Set

- Approximate with interval timer + a reference bit.
- Example: $\Delta = 10,000$ references
 - Timer interrupts after every 5000 references.
 - Keep in memory 2 bits for each page.
 - Whenever a timer interrupts copy and sets the values of all reference bits to 0.
 - If one of the bits in memory = 1 \Rightarrow page in working set.
- The cost to service these frequent interrupts is high.

Page-Fault Frequency



- Establish "acceptable" page-fault rate.
 - If actual rate too low, process loses frame.
 - If actual rate too high, process gains frame.

Other Considerations in demand-paging

- Pre-paging:
 - To prevent high page-fault rate at the beginning.
 - Try to bring more pages at once.
- Page size: in powers of 2 ($2^{12} - 2^{22}$)
 - Small page size \rightarrow large page-table.
 - Small page size \rightarrow less internal fragmentation.
 - Small page size \rightarrow more I/O overhead in paging.
 - Small page size \rightarrow more page-faults.
 - Small page size \rightarrow less I/O amount (better resolution).
less total allocated memory
 - A historical trend is toward larger page sizes.

Other Considerations in demand-paging (cont'd)

- 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