Virtual Memory and Demand Paging

Hui Chen [a]

[a] CUNY Brooklyn College

May 4, 2020

Outline

1. Virtual Memory and Demand Paging
2. Page Table and Page Fault
3. Copy-on-Write
4. Page Replacement and Workset
5. Page-Buffering Algorithms
6. Exercise and Experiment
7. Working-set Model and Thrashing

Outline

1. Virtual Memory and Demand Paging
2. Page Table and Page Fault
3. Copy-on-Write
4. Page Replacement and Workset
5. Page-Buffering Algorithms
6. Exercise and Experiment
7. Working-set Model and Thrashing

Policy vs. Reality

Recall

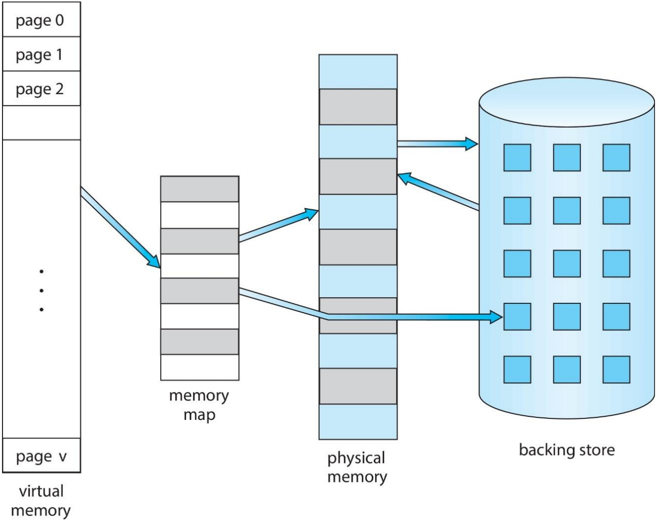
- Policy. Each process has a separate memory space.
    - Protection. Keep each process isolated.
    - Sharing. Allow memory to be shared between processes.
    - Virtualization. Provide applications with the illusion of "infinite" memory.

  However, we have only limited physical memory ...
    - Collectively in a multiprogramming system, the programs exceed physical memory.
    - A single program may become too large to fit in physical memory.
- Mechanism.
    - Demand paging.
    - ...

Demand Paging

Demand paging. Bring a page into memory only when it is needed and create an illusion of "infinite" memory.

Demand Paging



Benefits of Demand Paging

- Less I/O needed, no unnecessary I/O
- Less memory needed
- Faster response
- More users

Why & how?

Benefits and Observations

Demand paging gives the OS the ability to execute partially-loaded program.

- For each run of the program, not the entire program is used for each run of the program, e.g.,
    - Error code, unusual routines, large data structures
- The code and data structures for arun of the program are not needed at the same time.
- Each program takes less memory while running
    - More programs run at the same time (increased degree of multiprogramming)
    - Increased CPU utilization and throughput with no increase in response time or turnaround time
    - Less I/O needed to load or swap programs into memory
    - Each user program runs faster

Virtual Memory

Program no longer constrained by limits of physical memory by separating logical memory from physical memory.

$$\text{Virtual Memory} \equiv \text{Logical Memory}$$
$$\text{Virtual Memory Address} \equiv \text{Logical Memory Address}$$
$$\text{Virtual Memory Address Space} \equiv \text{Logical Memory Address Space}$$

Outline

1. Virtual Memory and Demand Paging
2. Page Table and Page Fault
3. Copy-on-Write
4. Page Replacement and Workset
5. Page-Buffering Algorithms
6. Exercise and Experiment
7. Working-set Model and Thrashing

Page Fault and Page Table Entry (PTE)

- Add a valid-invalid bit to each page table entry
  `v`. in-memory, or memory resident (Present in physical memory)
  `i`. not-in-memory (Not present in physical memory)
- For pure demand paging, initially valid–invalid bit is set to `i` on all entries
- During MMU address translation, if valid–invalid bit in page table entry is `i`, a page fault

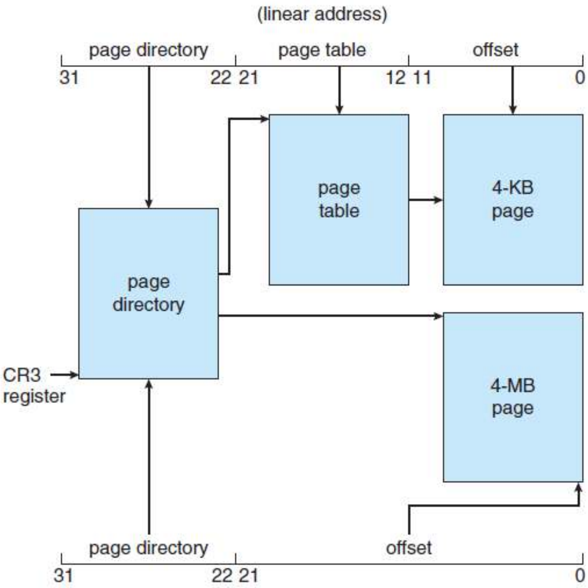
Example Page Table

Frame Number	Valid-Invalid bit
. . .	1 (v)
. . .	1 (v)
. . .	0 (i)
. . .	0 (i)
. . .	1 (v)
. . .	0 (i)

Intel IA-32 Page Table

Intel IA-32 uses a 2-level hierarchical paging.

- Intel IA-32's "Page Directory" Table is the outer table
- Intel IA-32's "Page Table" is the inner table


(linear address)
page directory
page table
offset
31
22 21
12 11
0
page table
4-KB page
page directory
4-MB page
CR3 register
page directory
offset
31
22 21
0

Intel IA-32 Outer Table Entry (Intel IA-32 PDE)

Frame Number	...	G	S	0	A	D	W	U	R	P
31 – 12	11 – 9	8	7	6	5	4	3	2	1	0

- P. If set, present (same as valid bit in other architectures) in physical memory.
- R. If set, readable and writable; otherwise, read only
- U. User or supervisor accessible
- W. If set, page write transparent, i.e., external cache write-through; if 0, write-back
- D. If set, page cache disabled (page cannot be cached)
- A. If set, accessed, i.e., page has been accessed (read or written) recently
- 0.
- S. Page size. 0 for 4KB, 1 for 4MB (huge) page size.
- G. Ignored.

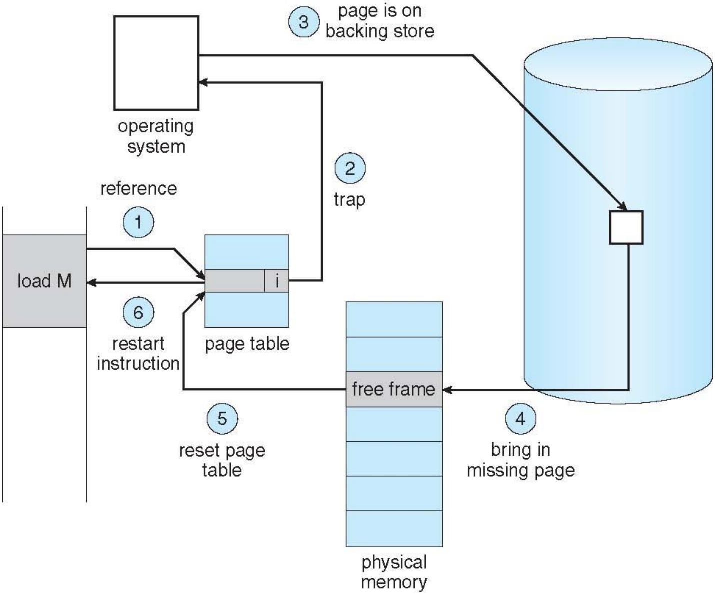
Intel IA-32 Inner Table Entry (Intel IA-32 PTE)

Frame Number	...	G	0	D	A	C	W	U	R	P
31 – 12	11 – 9	8	7	6	5	4	3	2	1	0

- P. Present (same as valid bit in other architectures)
- R. Writeable or read-only.
- U. User or supervisor accessible
- W. Page write transparent, i.e., external cache write-through
- C. Page cache disabled (page cannot be cached)
- A. Accessed. page has been accessed recently
- D. Dirty. if set, page has been modified recently
- G. Global flag. If set, prevents the TLB from updating the address in its cache if Control Register CR3 is reset. Note, that the page global enable bit in Control Register CR4 must be set to enable this feature.

Handling Page Fault

1. If there is a reference to a page, first reference to that page will trap to operating system, i.e., a Page fault
2. Operating system looks at the page table to decide:
   2.1 if invalid reference, abort
   2.2 if just not in memory
3. Find free frame
4. Swap page into frame via scheduled disk operation
5. Reset tables to indicate page now in memory, an set validation bit to v
6. Restart the instruction that caused the page fault


3
page is on backing store
operating system
2
trap
reference
1
load M
i
6
restart instruction
page table
free frame
5
reset page table
4
bring in missing page
physical memory

Performance of Demand Paging

Three major activities

- Service the interrupt – careful coding means just several hundred instructions needed
- Read the page – lots of time
- Restart the process – again just a small amount of time

Page Fault Rate $0 \leq p \leq 1$

- if p $=$ 0 no page faults
- if p $=$ 1, every reference is a fault

$$\begin{aligned}&\text{Effective Access Time (EAT)} = \\ &\quad (1-p)(\text{memory access}) + \\ &\quad p(\text{page fault overhead} + \text{swap page out} + \text{swap page in})\end{aligned}$$

Outline

1 Virtual Memory and Demand Paging

2 Page Table and Page Fault

3 Copy-on-Write

4 Page Replacement and Workset

5 Page-Buffering Algorithms

6 Exercise and Experiment

7 Working-set Model and Thrashing

Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially share the same pages in memory
- If either process modifies a shared page, only then is the page copied
- COW allows more efficient process creation as only modified pages are copied
- `fork()` vs. `vfork()`

Outline

1 Virtual Memory and Demand Paging

2 Page Table and Page Fault

3 Copy-on-Write

4 Page Replacement and Workset

5 Page-Buffering Algorithms

6 Exercise and Experiment

7 Working-set Model and Thrashing

What if there is no more free frame?

- Page replacement – find some page in memory, but not really in use, page it out
- Algorithm – terminate? swap out? replace the page?
- Performance – want an algorithm which will result in minimum number of page faults
- Same page may be brought into memory several times

Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement
- Use modify (dirty) bit to reduce overhead of page transfers only modified pages are written to disk
- Page replacement completes separation between logical memory and physical memory
- Large virtual memory can be provided on a smaller physical memory

Basic Algorithm

1. Find the location of the desired page on disk
2. Find a free frame: - If there is a free frame, use it - If there is no free frame, use a page replacement algorithm to select a victim frame- Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap
5. Note now potentially 2 page transfers for page fault – increasing EAT

Frame Allocation and Replacement

- Frame-allocation algorithm determines how many frames to give each process
- Page-replacement algorithm which frames to replace
- Want lowest page-fault rate on both first access and re-access

Evaluting Page Replacement Algorithms

- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
- String is just page numbers, not full addresses
- Repeated access to the same page does not cause a page fault
- Results depend on number of frames available
- In all our examples, the reference string of referenced page numbers is 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

Algorithms

- Optimal Algorithm
- First-In-First-Out (FIFO) Algorithm
- Least Recently Used (LRU) Algorithm
- LRU Approximation Algorithms
- Second-Chance Algorithm
- Enhanced Second-Chance Algorithm
- Counting Algorithm
- Page-Buffering Algorithms

FIFO Page Replacement

- Straightforward application of Queue
- Select the head of the queue as the victim
- How many page faults are there?

LRU Page Replacement

- Order pages based on access time
  - An access may change the order of the pages
- Select the oldest page (based on access time, i.e., the “least recently used”) as the victim
- How many page faults are there?

LRU Page Replacement

- Order pages based on access time
    - An access may change the order of the pages
- Select the oldest page (based on access time, i.e., the “least recently used”) as the victim
- How many page faults are there?

LRU Implementation

How do we implement LRU?

- Use a counter to save the clock of each access.
- Use a stack of page numbers. Move page referenced to the top of the stack.
- How many memory accesses for each strategy?
    - at each access?
    - for selecting a victim?

LRU Approximation

LRU is slow. Use approximation.

- Second Chance algorithm (also called, the Clock algorithm)
- Enhanced Change algorithm

Second Chance (Clock) Page Replacement

- Storage page numbers in a “circular” FIFO queue
- Associate each page with a hardware-provided reference bit, initialized as 0
  - When page is referenced, set bit to 1
- Maintain a pointer, initially pointing to the first page referenced (like a clock hand/arm).
- To select a victim, iterate over pages starting from the one pointed by the pointer (the clock hand/arm) in the following manner,
  1. if the reference bit of the page pointed by the pointer is 0, select it as victim
  2. otherwise, set the page's reference bit to 0, advance the pointer to next page
  3. go to 1

How many page faults are there?

Enhanced Second Chance (Clock) Page Replacement

- Still a “Clock” algorithm.
- Introduce a `modify` bit for each page, initialized it as 0
  - Set it to 1 if the process modify the page’s content (i.e., the process writes to page’s memory)
- Select a victim (iterating over pages starting from the page pointed by the clock hand) based on the order of preference below (a heuristics)
  1. (0, 0) neither recently used not modified – best page to replace
  2. (0, 1) not recently used but modified – not quite as good, must write out before replacement
  3. (1, 0) recently used but clean – probably will be used again soon
  4. (1, 1) recently used and modified – probably will be used again soon and need to write out before replacement

How many page faults are there?

```
R,R,R,R,R,R,R,W,W,R,W,R,W,W,W,R,R,R,R,R,R,R
7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1
```

Counting Algorithms

Not common.

- Keep a counter of the number of references that have been made to each page
- Lease Frequently Used (LFU) Algorithm: replaces page with smallest count
- Most Frequently Used (MFU) Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Outline

1. Virtual Memory and Demand Paging
2. Page Table and Page Fault
3. Copy-on-Write
4. Page Replacement and Workset
5. Page-Buffering Algorithms
6. Exercise and Experiment
7. Working-set Model and Thrashing

Page-Buffering Algorithms

- Always keep a pool of free frames
- Also keep a pool of modified pages
  - When backing store otherwise idle, write pages there and set to non-dirty
- *Possibly*, keep free frame contents intact and note what is in them
  - If referenced again before reused, no need to load contents again from disk
  - Generally useful to reduce penalty if wrong victim frame selected
- Minor vs. major page faults
  - Shared library?
  - Free frame (buffered), but not yet zeroed-out and allocated to other processes

Outline

1. Virtual Memory and Demand Paging
2. Page Table and Page Fault
3. Copy-on-Write
4. Page Replacement and Workset
5. Page-Buffering Algorithms
6. Exercise and Experiment
7. Working-set Model and Thrashing

Paging and Program Structure

Exercise 10.7. Consider the two-dimensional array A:

```
int A[][] = new int[100][100];
```

where A[0][0] is at location 200 in a paged memory system with pages of size 200. A small process that manipulates the matrix resides in page 0 (locations 0 to 199). Thus, every instruction fetch will be from page 0. For three page frames, how many page faults are generated by the following array-initialization loops? Use LRU replacement, and assume that page frame 1 contains the process and the other two are initially empty

- Program (a)

```
for (int j = 0; j < 100; j++)
  for (int i = 0; i < 100; i++)
    A[i][j] = 0;
```

- Program (b)

```
for (int i = 0; i < 100; i++)
  for (int j = 0; j < 100; j++)
    A[i][j] = 0;
```

Experiment

Let's conduct an experiment illustrating the above exercise ...

- `OSClassExamples/memory/linux/paging/arrayfiller`

Outline

1. Virtual Memory and Demand Paging
2. Page Table and Page Fault
3. Copy-on-Write
4. Page Replacement and Workset
5. Page-Buffering Algorithms
6. Exercise and Experiment
7. Working-set Model and Thrashing

Thrashing

A process is *thrashing* if the process experiences high paging activity, as the result, the process is spending more time paging than executing.

1. Paging activity increases, CPU utilization decreases.
2. When CPU scheduler sees low CPU utilization, schedule more processes to run to utilize the idle CPU.
3. Paging activity becomes even higher, CPU utilization becomes even lower.
4. CPU scheduler schedule even more processes to run.
5. Go to 3

Locality Model

- As a process executes, it moves from locality to locality.
- A locality is a set of pages that are actively used together.
- A running program is generally composed of several different localities, which may overlap.

Working-set Model

Based on the assumption of locality, define the set of pages in the most recent Δ page references as a working set where we call Δ the working-set window.

- Let Δ be 10, what is the working set at time immediately after the 10th page reference (counting from 1) and what immediately after page reference 26?

  `2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2`

Frame Allocation and Working-set Model

We ought to select Δ and allocate frames such that

$$D(t) = \sum_{i}^{N} |\mathbb{W}_i(\Delta, t)| \le m(t) \tag{1}$$

where

- N is the number of processes,
- $|\mathbb{W}_i(\Delta, t)|$ is the size of the working set of process i,
- $D(t)$ is the total demand for frames, i.e., the sum of the sizes of all working sets of all processes at time t, and
- $m(m)$ is the number of available frames at time t.