Virtual Memory

Amir H. Payberah
payberah@kth.se
2022

Motivation

- A program needs to be in memory to execute.

Motivation

- A program needs to be in memory to execute.
- But, entire program rarely used (e.g., unusual routines, large data structures).

Motivation

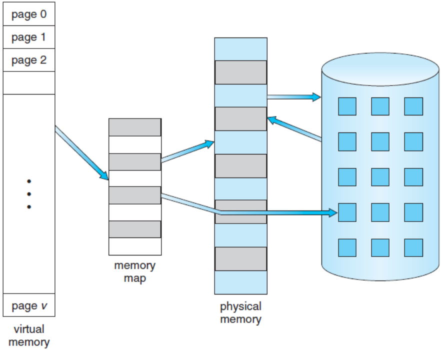
- A program needs to be in memory to execute.
- But, entire program rarely used (e.g., unusual routines, large data structures).
- Only part of the program needs to be in memory for execution.

Motivation

- A program needs to be in memory to execute.
- But, entire program rarely used (e.g., unusual routines, large data structures).
- Only part of the program needs to be in memory for execution.
- More programs running concurrently.

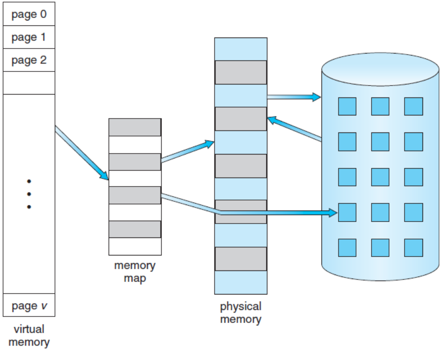
Virtual Memory

- Separation of user logical memory from physical memory.



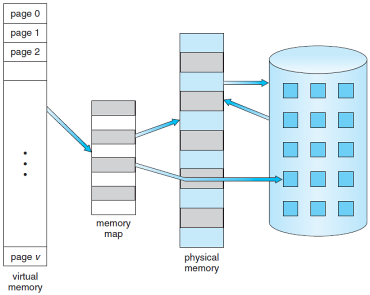
Virtual Memory

- Separation of user logical memory from physical memory.
- Logical address space can be much larger than physical address space.



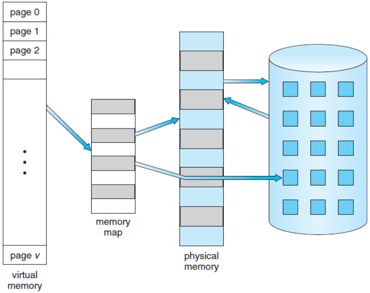
Virtual Address Space (1/2)

- Virtual address space: logical view of how process is stored in memory.



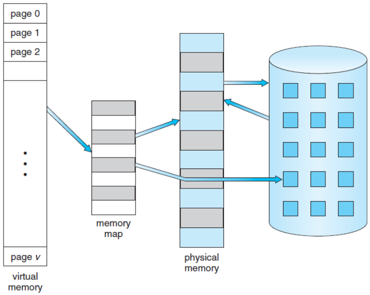
Virtual Address Space (1/2)

- Virtual address space: logical view of how process is stored in memory.
- Meanwhile, physical memory organized in page frames.



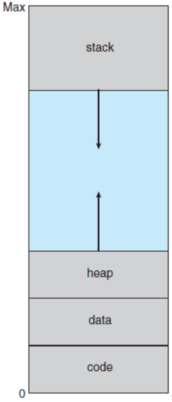
Virtual Address Space (1/2)

- Virtual address space: logical view of how process is stored in memory.
- Meanwhile, physical memory organized in page frames.
- MMU must map logical to physical.



Virtual Address Space (2/2)

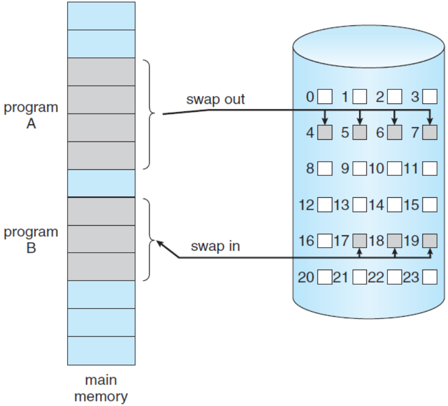
- The hole between heap and stack is part of the virtual address space, but will require actual physical pages only if the heap or stack grows.



Demand Paging

Demand-Paging

- Demand-paging: bring a page into memory only when it is needed.
- lazy swapper.

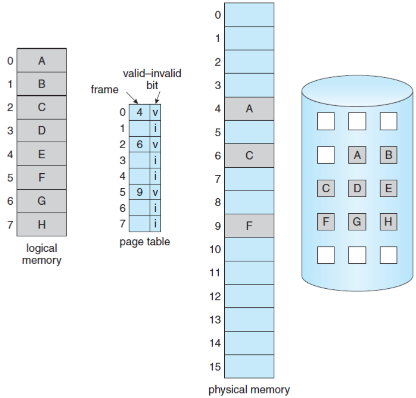


Basic Concepts

- The pager guesses which pages will be used again before swapping out.

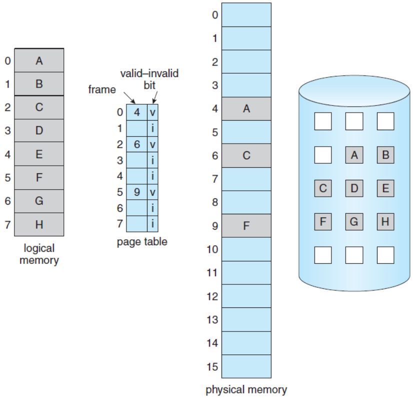
Basic Concepts

- The pager guesses which pages will be used again before swapping out.



Basic Concepts

- The pager guesses which pages will be used again before swapping out.
- Valid-invalid bit: distinguish between the pages in memory and on disk.
  v: memory resident
  i: not in memory

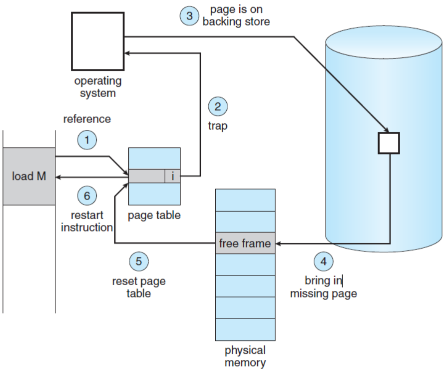


Page Fault

- Access to a page marked invalid causes a page fault.
- Causing a trap to the OS: brings the desired page into memory.

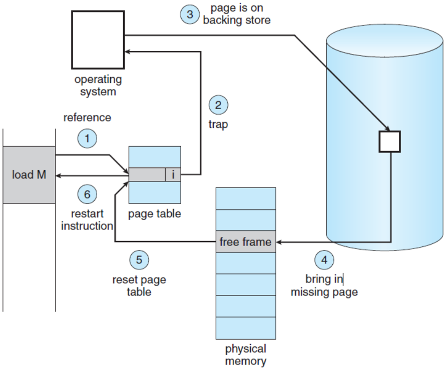
Handling Page Fault (1/6)

- Check an internal table for the process to determine whether the reference was a valid or an invalid memory access.



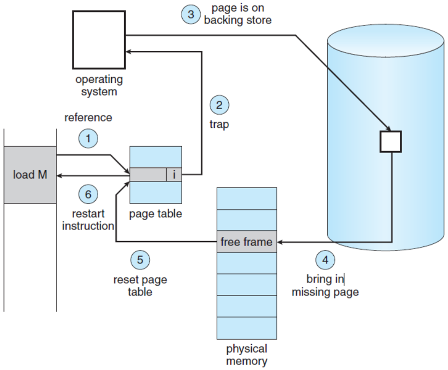
Handling Page Fault (2/6)

- If the reference was invalid, we terminate the process.



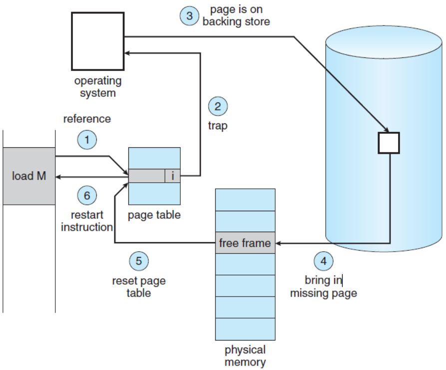
Handling Page Fault (2/6)

- If the reference was invalid, we terminate the process.
- If it was valid but we have not yet brought in that page, we now page it in.



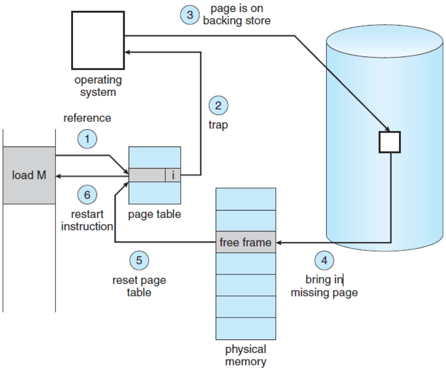
Handling Page Fault (3/6)

- We find a free frame.



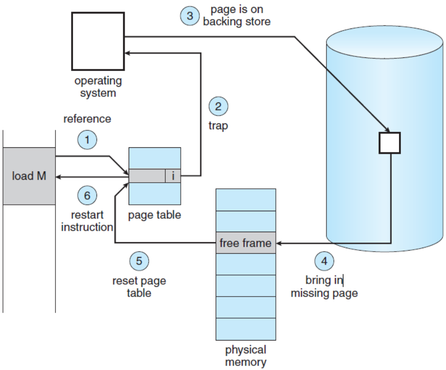
Handling Page Fault (4/6)

- We schedule a disk operation to read the desired page into the newly allocated frame.



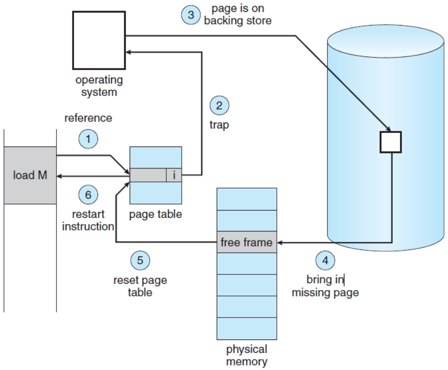
Handling Page Fault (5/6)

- When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.



Handling Page Fault (6/6)

- We restart the instruction that was interrupted by the trap.



Page Replacement

What Happens if There is no Free Frame?

- Assume, we had 40 frames in physical memory.
- And, we run 6 processes, each of which is 10 pages in size, but actually uses only 5 pages.

What Happens if There is no Free Frame?

- Assume, we had 40 frames in physical memory.
- And, we run 6 processes, each of which is 10 pages in size, but actually uses only 5 pages.
- It is possible that each of these processes may suddenly try to use all 10 of its pages: resulting in a need for 60 frames when only 40 are available.

What Happens if There is no Free Frame?

- Assume, we had 40 frames in physical memory.
- And, we run 6 processes, each of which is 10 pages in size, but actually uses only 5 pages.
- It is possible that each of these processes may suddenly try to use all 10 of its pages: resulting in a need for 60 frames when only 40 are available.
- Increasing the degree of multiprogramming: over-allocating memory

Over-Allocation of Memory

- While a user process is executing, a page fault occurs.

Over-Allocation of Memory

- While a user process is executing, a page fault occurs.
- The OS determines where the desired page is residing on the disk.

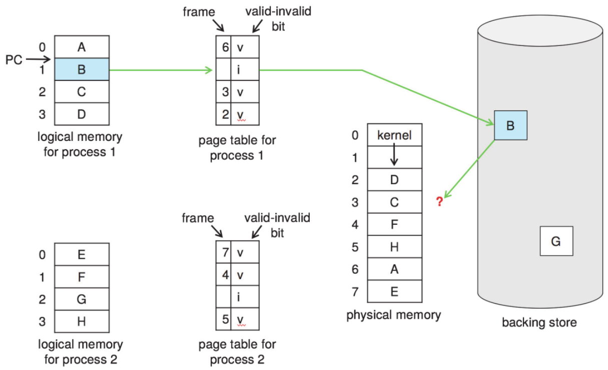
Over-Allocation of Memory

- While a user process is executing, a page fault occurs.
- The OS determines where the desired page is residing on the disk.
- But, it finds that there are no free frames on the free-frame list.

Over-Allocation of Memory

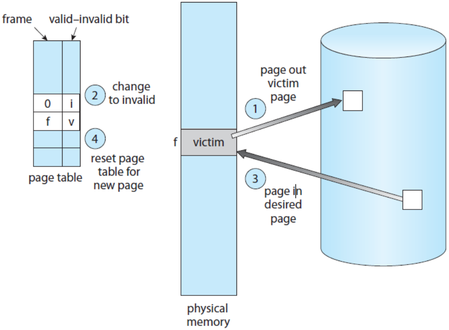
- While a user process is executing, a page fault occurs.
- The OS determines where the desired page is residing on the disk.
- But, it finds that there are no free frames on the free-frame list.
- Need for page replacement

Need For Page Replacement



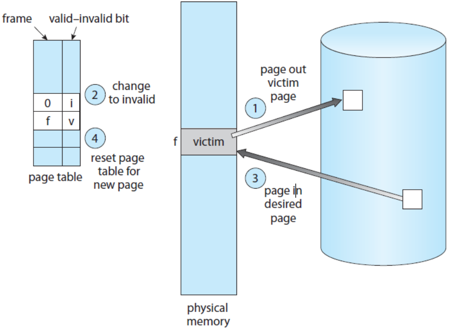
Page Replacement (1/4)

- Find the location of the desired page on disk.



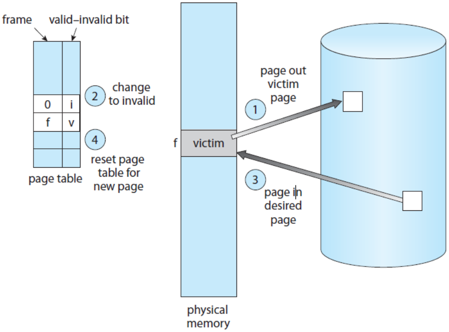
Page Replacement (2/4)

- Find a free frame.



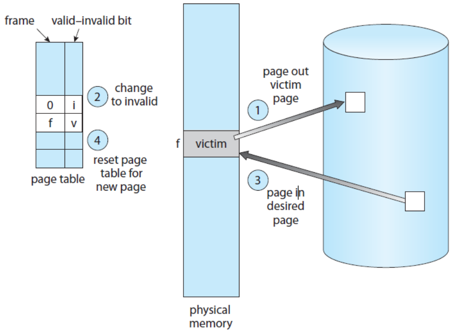
Page Replacement (2/4)

- Find a free frame.
  - If there is a free frame, use it.



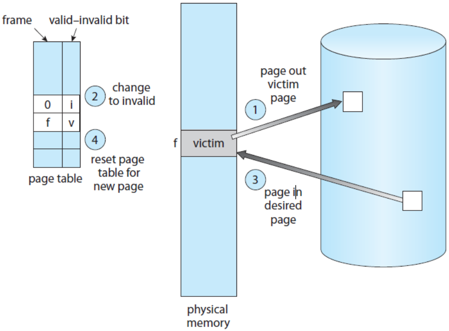
Page Replacement (2/4)

- 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.



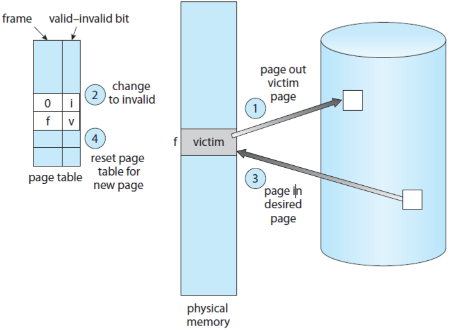
Page Replacement (3/4)

- Bring the desired page into the (newly) free frame; update the page and frame tables

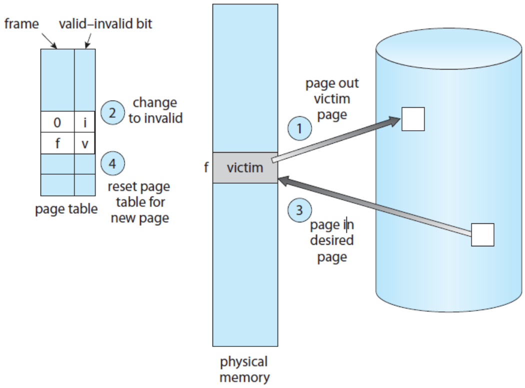


Page Replacement (4/4)

- Continue the process by restarting the instruction that caused the trap.



Dirty Bit



- Use modify (dirty) bit to reduce overhead of page transfers - only modified pages are written to disk.

Page Replacement Algorithms

Evaluate Page Replacement Algorithms

- Refernce string is a sequence of page numbers.

Evaluate Page Replacement Algorithms

- Refernce string is a sequence of page numbers.
- Assume a reference string could be
  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

Evaluate Page Replacement Algorithms

- Refernce string is a sequence of page numbers.
- Assume a reference string could be
  7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
- Evaluate algorithm by running it on a reference string and computing the number of page faults on that string.

Page Replacement Algorithms

- First-In-First-Out (FIFO)
- Optimal
- Least Recently Used (LRU)
- LRU-Approximation
- Counting-Based

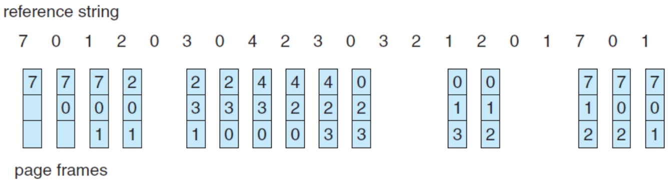
FIFO Page Replacement

FIFO Page Replacement

- Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

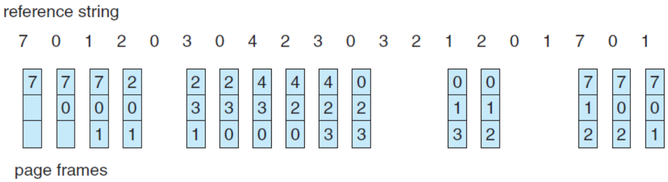
FIFO Page Replacement

- Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1
- 3 frames (3 pages can be in memory at a time per process)



FIFO Page Replacement

- Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1
- 3 frames (3 pages can be in memory at a time per process)



- 15 page faults

Optimal Page Replacement

Optimal Page Replacement

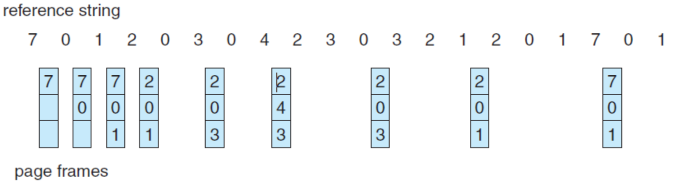
- Replace page that will not be used for longest period of time.

Optimal Page Replacement

- Replace page that will not be used for longest period of time.
- How do you know this? Can't read the future!

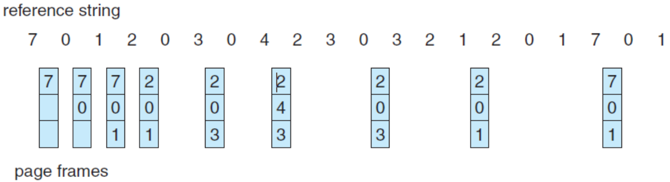
Optimal Page Replacement

- Replace page that will not be used for longest period of time.
- How do you know this? Can't read the future!
- 9 faults is optimal for this example.



Optimal Page Replacement

- Replace page that will not be used for longest period of time.
- How do you know this? Can't read the future!
- 9 faults is optimal for this example.



- Used for measuring how well your algorithm performs.

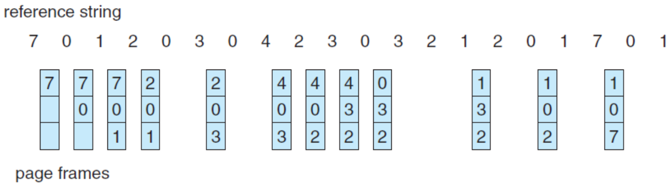
LRU Page Replacement

LRU Page Replacement

- Use past knowledge rather than the future.

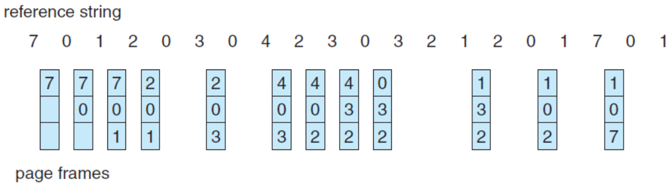
LRU Page Replacement

- Use past knowledge rather than the future.
- Replace page that has not been used in the most amount of time.



LRU Page Replacement

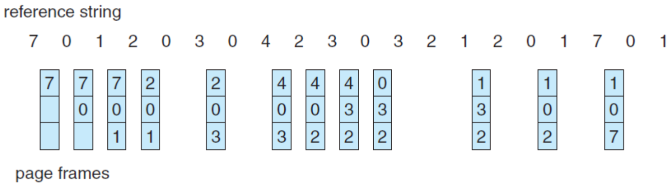
- Use past knowledge rather than the future.
- Replace page that has not been used in the most amount of time.



- 12 faults: better than FIFO but worse than OPT

LRU Page Replacement

- Use past knowledge rather than the future.
- Replace page that has not been used in the most amount of time.



- 12 faults: better than FIFO but worse than OPT
- Generally good algorithm and frequently used

LRU Implementation (1/2)

- Counter implementation

LRU Implementation (1/2)

- Counter implementation
- Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.

LRU Implementation (1/2)

- Counter implementation
- Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
- When a page needs to be changed, look at the counters to find smallest value.

LRU Implementation (1/2)

- Counter implementation
- Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
- When a page needs to be changed, look at the counters to find smallest value.
- Search through table needed.

LRU Implementation (2/2)

- Stack implementation

LRU Implementation (2/2)

- Stack implementation
- Keep a stack of page numbers in a double link form.

LRU Implementation (2/2)

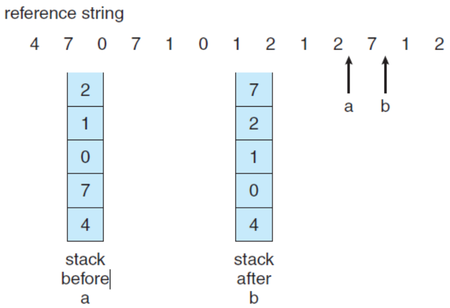
- Stack implementation
- Keep a stack of page numbers in a double link form.
- Page referenced: move it to the top

LRU Implementation (2/2)

- Stack implementation
- Keep a stack of page numbers in a double link form.
- Page referenced: move it to the top
- No search for replacement.

Stack Implementation

- Use of a stack to record most recent page references.



LRU-Approximation Page Replacement

LRU-Approximation Page Replacement

- LRU needs special hardware and still slow
- Improvements: LRU-Approximation

LRU-Approximation Page Replacement

- LRU needs special hardware and still slow
- Improvements: LRU-Approximation
    - Reference bit

LRU-Approximation Page Replacement

- LRU needs special hardware and still slow
- Improvements: LRU-Approximation
  - Reference bit
  - Second-chance algorithm

Reference Bit

- With each page associate a bit, initially $= 0$

Reference Bit

- With each page associate a bit, initially $= 0$
- When page is referenced, bit set to 1

Reference Bit

- With each page associate a bit, initially $= 0$
- When page is referenced, bit set to 1
- Replace any with reference bit $= 0$ (if one exists)

Reference Bit

- With each page associate a bit, initially $= 0$
- When page is referenced, bit set to 1
- Replace any with reference bit $= 0$ (if one exists)
- We do not know the order

Second-Chance Algorithm (1/2)

- Generally FIFO, plus hardware-provided reference bit

Second-Chance Algorithm (1/2)

- Generally FIFO, plus hardware-provided reference bit
- If page to be replaced has

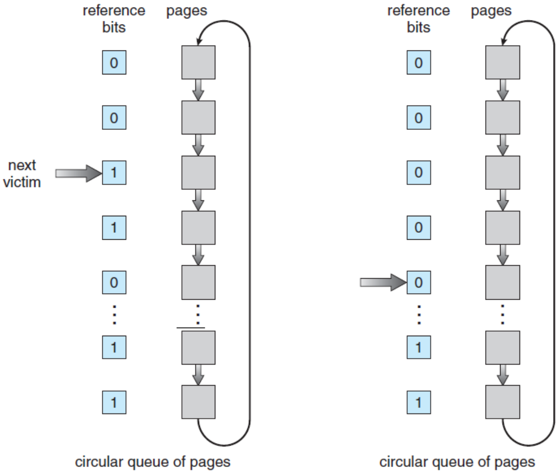
Second-Chance Algorithm (1/2)

- Generally FIFO, plus hardware-provided reference bit
- If page to be replaced has
  - Reference bit $= 0 \rightarrow$ replace it

Second-Chance Algorithm (1/2)

- Generally FIFO, plus hardware-provided reference bit
- If page to be replaced has
  - Reference bit $= 0 \rightarrow$ replace it
  - Reference bit $= 1$ then, set reference bit 0, leave page in memory, and replace next page, subject to same rules.

Second-Chance Algorithm (2/2)



Counting Page Replacement

Counting Page Replacement

- Keep a counter of the number of references that have been made to each page.

Counting Page Replacement

- 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.

Counting Page Replacement

- 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.

Summary

Summary

- Partially-loaded programs

Summary

- Partially-loaded programs
- Virtual memory: much larger than physical memory

Summary

- Partially-loaded programs
- Virtual memory: much larger than physical memory
- Demand paging similar to paging + swapping

Summary

- Partially-loaded programs
- Virtual memory: much larger than physical memory
- Demand paging similar to paging + swapping
- Page fault

Summary

- Partially-loaded programs
- Virtual memory: much larger than physical memory
- Demand paging similar to paging + swapping
- Page fault
- Page replacement algorithms:
  - FIFO, optimal, LRU, LRU-approximate, counting-based

Questions?

Acknowledgements

Some slides were derived from Avi Silberschatz slides.