5. Least Recently Used (LRU) Page Replacement Algorithm

- Assume pages used recently will used again soon. ~~through~~
  - Throw out page that has been unused for longest time when page faults occurs
- Must keep a linked list of pages
  - most recently used at front, least at rear.
  - Update this list every memory reference!!
- This is theoretically realizable but not cheap!
  - linked list is needed to be maintained with all the pages in memory. It is difficult to update on every memory reference.
  - Finding a page in the list, deleting it, and then moving it to the front is a very time consuming operation even in hardware level.
- Alternatively keep counter in each page table entry
  - Counter track the number of references to a page
  - Choose page with lowest value counter
  - Periodically zero the counter.

[redacted] self study

Not Frequently Used (NFU) Algorithm

- simulates LRU in software
- It requires a software counter associated with each page in memory.
- At each clock interrupt, the OS scans all the pages in memory
- For each page, the referenced bit, which is 0 or 1, is added to the counter
- the counters keep track of how often each page has been referenced.
- When a page fault occurs, the page with the lowest counter is chosen for replacement

Disadvantage: it never forgets anything consequently the OS may remove useful pages instead of pages no longer in used

Aging (Approximation of LRU) Algorithm

- Periodically
  1. shift each counter to right by 1 bit
  2. Add value of corresponding referenced bit to left most bit
  3. Reset all reference bits to 0

A. Suppose after first clock tick the reference bit for pages 0 to 5 have values 1, 0, 1, 0, 1 and 1, respectively (page 0 is 1, page 1 is 0, ...).

B. Between tick 0 and 1, pages 0, 2, 4, and 5 were referenced setting their reference bit to 1, while other ones remain 0.

R bits for pages 0-5
Clock tick 0

1	0	1	0	1	1

→

Page	
0	10000000
1	00000000
2	10000000
3	00000000
4	10000000
5	10000000

→

C. After six corresponding counters have been shifted and the referenced bit inserted at the left, they have the following values

←

C. When a page fault occurs, the page whose counter is the smallest is removed

Bits for pages 0-5
Clock tick 4 | 0 | 1 | 1 | 0 | 0 | 0 |

0	01111000
1	10110000
2	10010000
3	00100000
4	01011000
5	00101000

- It is clear that a page that has not been referenced for, say, four clock ticks will have 4 leading zeros in its counter and thus will have a lower value than a counter that has not been referenced for three clocks

[redacted] Self Study

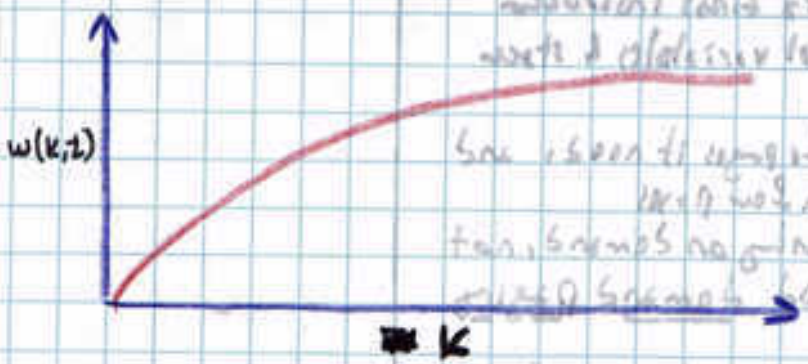
Aging & Working Set Page Replacement Algorithm

A. Pages started up without pages in memory
B. CPU tries to fetch first instruction getting a page fault
C. OS brings in the page containing first instruction
D. Other page faults for global variables & stack follow quickly
E. the process has most of the pages it needs and settle down to run with few page faults
F. Because pages are loaded only on demand, not in advance, this is called demand paging

- However most processes exhibit a locality of reference which means that during any phase of execution the process references only small fraction of its pages.
- Working set: the set of all pages that a process is currently actively using.
  - If the entire working set is in memory, the process will run without causing many faults until moves into another execution phase.
  - If the available memory is too small to hold the entire working set, the process will cause many page faults and run slowly (since reading from the disk)
  - ~~A program causing many few instructions~~
  - A program causing page faults every few instructions is said to be thrashing

Working set page replacement Algorithm (continued)

- the working set is the set of pages used by the k most recent memory references
- w(k,t) is the size of working set at time, t



key observation:

- working set size (and membership) for typical processes is constant
- If we can identify and keep the working set in memory then we will minimize page faults

Current virtual time: is the amount of CPU time a process has actually used since it started. This is an approximation in which the working set of a process is the set of pages it has referenced during the past t seconds of virtual time.

[redacted] self study

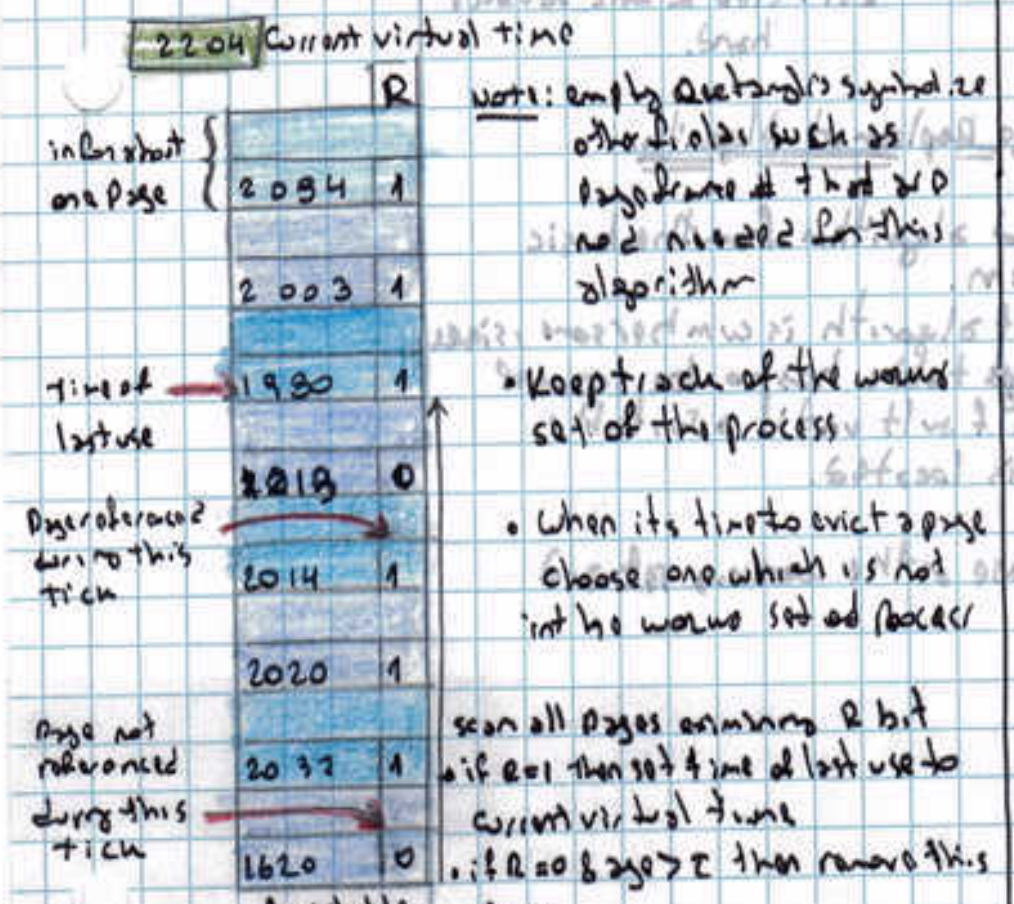
Working set Page Replacement Algorithm

Page replacement Algorithm:

A. Find a page that is not in the working set and is evicted it.

B. Because only pages that are in memory are considered as candidates for eviction, pages that are absent from memory are ignored.

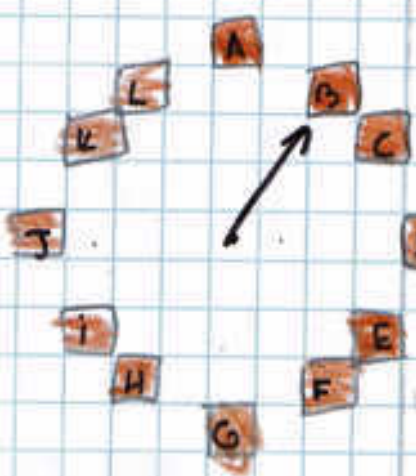
C. Each entry contains (at least) two key items of info:
   1. Approximate time the page was last used
   2. Reference bit



9. Clock Page Replacement Algorithm

- same as second chance algorithm; however, it doesn't keep moving pages around the linked list.

- We keep all the page frames on a circular list in the form of a clock



- When a page fault occurs the page the hand is pointing to is inspected
  - the action depends on the Reference Bit:
    - R = 0 : Evict the Page; put new page in its position
    - R = 1 : Clear R and advance hand.

10. The WSClock Page Replacement Algorithm

- This is an improved algorithm from the basic working set algorithm.
  - → the working set algorithm is cumbersome, since the entire page table has to be scanned at each page fault until a suitable candidate is located.

- This algorithm is based on the working set and clock algorithm

[redacted] self study

the WSClock Page Replacement Algorithm (continued)

A. data structure is a circular list of page frames.

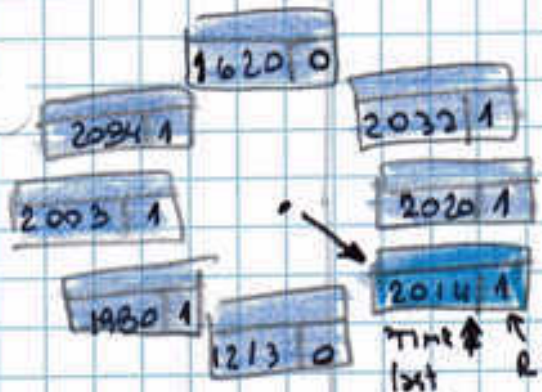
B. Initially this list is empty

C. when first page is loaded it is added to the list

D. list of pages forms a ring.

E. each entry contains the time of last use field (from working set) as well that Referenced bit and Modified bit (not shown)

2204 Current virtual Time



F. If R=0, the page has not been referenced during the current clock tick and may be candidate for removal.

G. to see if need to be removed its age (current virtual time minus time of last use) is compared with τ page is no longer in the working set and new page replaces it.

H. If R=0 & age ≤ τ then page still in the working set the page is spared but the page with greatest age is noted.

I. If entired table is scanned without finding a candidate to evict, it means that all pages are in the working set.
→ if one or more pages with R=0 were found, one with greatest age is evicted

[redacted] self study

Design Issues for Paging Systems

Local vs. Global Replacement Policies

- suppose we got a page fault

(Original Configuration)

	page
A0	10
A1	7
A2	5
A3	4
A4	6
A5	3
B0	9
B1	11
B2	6
B3	2
B4	5
B5	6
B6	12
C1	3
C2	5
C3	6

If we have only at A's pages, the page with lowest value is removed

↓ we obtain

(Local Page replacement)

A0
A1
A2
A3
A4
A6
B0
B1
B2
B3
B4
B5
B6
C1
C2
C3

If page w/ the lowest age value removed without regards to who page it is

↓ we obtain

(Global Page replacement)

A0
A1
A2
A3
A4
A5
B0
B1
B2
A6
B4
B5
B6
C1
C2
C3

Local vs Global Replacement Policies

- Local page replacement: Effectively correspond to allocating every process a fixed fraction of the memory
- Global page replacement: dynamically allocate page frames among runnable processes
- In general Global algorithm work better (especially when working set size can vary over the lifetime of a process)
- If local algorithm is used and the working set grows it will result in thrashing. If working set shrinks then local algorithms waste memory
- If global algorithm is used, the system must continually decide how many page frames to assign each proc
- One way to manage allocation when using global algorithm is to use Page Fault Frequency (PFF)
  → it tells when to increase or decrease a process' page allocation but says nothing about which page to replace on a fault. It just controls the size of the allocation set

[redacted] selfstudy

Internal vs. External Fragmentation

- **Internal Fragmentation:**
  - Occurs when part of memory allocated to a process remains unused
  - With n segments in memory a page size of p bytes, np/2 bytes will be wasted
    Eg: a process may allocate a 4K page but it only uses 1-byte in the page

- **External Fragmentation**
  - Occurs when none of the available free memory fragments is big enough to satisfy a new memory allocation request
  - Eg: a process may request 16KB contiguous physical memory allocation, but all available free memory is of size less than 16KB

- Can a virtual memory paging system have internal fragmentation?

  In paging there is internal fragmentation but not external

- Can a virtual memory system have external fragmentation?

  Only in segmentation there is external fragmentation

Page size

- Page size is a parameter chosen by the OS
- No overall optimum.
- two factors:
  1. A randomly chosen text, data, or stack segment will not fill an integral number of pages
  2. On average, half of the final page will be empty (internal fragmentation)
- Small page size:
- Advantages:
  1. less internal fragmentation
  2. better fit for various data structures, code sections
  3. less unused program in memory
- Disadvantages:
  1. large page tables.

Periodic Cleaning Policy

- Need for a background process, paging daemon
  - periodically inspect state of memory
- When too few page frames are free
  - select pages to evict using a replacement algorithm.

Thrashing

- Despite good designs, systems may still thrash when:
  1. some processes need more memory
  2. but no process needs less.

[redacted] selfstudy

Thrashing (continue)

- when the working set of all processes do not fit in main memory
- Hence, paging daemon has to constantly page-out pages to disk and bring them back in almost immediately.

Solution: Reduce the degree of multi-programming number of processes competing for memory

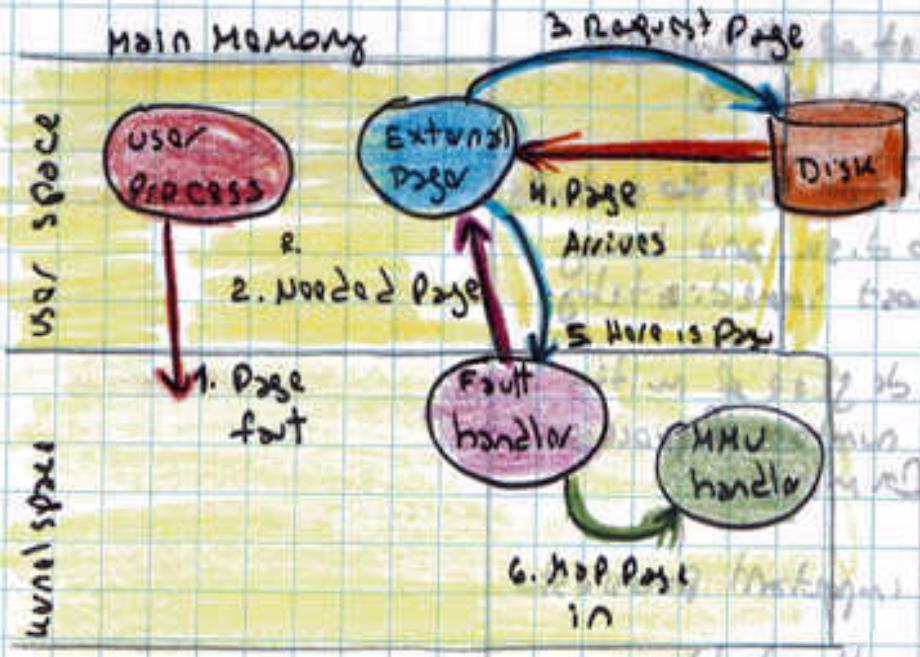
1. Swap some less important processes to disk
2. Divide up pages they hold.

Separation of Policy and Mechanism

- Important for managing the complexity of any system: separation of policy from mechanism
- can be applied to memory management by having most of the memory manager run as user level process
- The memory management can be divided into three parts
  1. low-level MMU handler
  2. A page fault handler that is part of the kernel
  3. An external pager running in user space.

Separation of Policy and Mechanism (Cont. two)

Page fault handling with an external pager



- MMU code encapsulated in the MMU handler
- Page fault handler is machine-independent & contains most mechanism of pager
- Policy is largely determined by the external pager (runs in user process)

A. When process starts up, external pager is notified ahead to set up the process' page map and allocate backing store on the disk if need be.

B. As process runs, map new objects into address space, so external pager is not. Used (again)

C. Once process starts running it may get page fault
i. fault handler figures out which virtual page is needed and sends message to the external pager, telling it the problem

[redacted] self study

Separation of Policy and Mechanism (Continued)

D. External pager then reads needed page in from the disk and copies it to a portion of its own address space

E. then it tells the fault handler where the page is

F. the fault handler then unmaps the page from the external pager address space and asks MMU handler to put it into the user's address space at right place

G. then the user process can be restarted

Notes:

Notes:

Concurrency, Race Conditions, Mutual Exclusion, Semaphores, Monitors, Deadlock

Concurrency vs. Parallelism

A. Concurrency:

1. when two or more control flows (threads) of execution share one or more CPUs
2. in such case: CPU scheduler is responsible for deciding when each thread gets to execute and on which CPU

ex: Even if there is only one CPU, but two or more threads share the CPU then ~~so~~ it's considered concurrent execution

3. ~~• If there are 2 flows / 2 cores execute at the same time~~

B. Parallelism:

- need two or more CPUs
- Is a subset of concurrency
- It's when two or more threads execute at the same time on two or more CPUs

ex: Three threads executing on three ~~threads~~ different CPUs

- note: I use the term "threads" above to refer to either threads or processes.

114 Critical Sections (crit. cal region)

- Prevent a troubles involving shared memory
- A section of code that modifies or accesses a shared resource which can be modified by another process concurrently
- two pieces of code that are sharing the same section is called critical section
- If two process (P1 and P2) access to the same critical section, there is a chance to have a corruption of Data; therefore identifying critical sections is important when working with concurrency

Ex A. A piece of code that reads from or writes to a shared memory region

B. A code that modifies or traverses a linked list that can be accessed concurrently by another thread

Solution:

1. We need mutual exclusion: some way of making sure that if one process is using a shared variable or file, the other process will be excluded from doing the same thing

- ~~Exclusion~~
- mutual exclusion is a major design issue