بسم الله الرحمن الرحیم

۲۲۲

«سیستم عامل»

جلسه ۲۱: مدیریت حافظه (۹)

Working set page replacement

- **Demand paging**
  - Pages are only loaded when accessed
  - When process begins, all pages marked INVALID

Working set page replacement

- **Demand paging**
  - Pages are only loaded when accessed
  - When process begins, all pages marked INVALID
- **Locality of reference**
  - Processes tend to use only a small fraction of their pages

Working set page replacement

- **Demand paging**
  - Pages are only loaded when accessed
  - When process begins, all pages marked INVALID
- **Locality of Reference**
  - Processes tend to use only a small fraction of their pages
- **Working Set**
  - The set of pages a process needs
  - If working set is in memory, no page faults
  - What if you can't get working set into memory?

Working set page replacement

- **Thrashing**
  - If you can't get working set into memory page faults occur every few instructions
  - Little work gets done
  - Most of the CPU's time is going on overhead

Working set page replacement

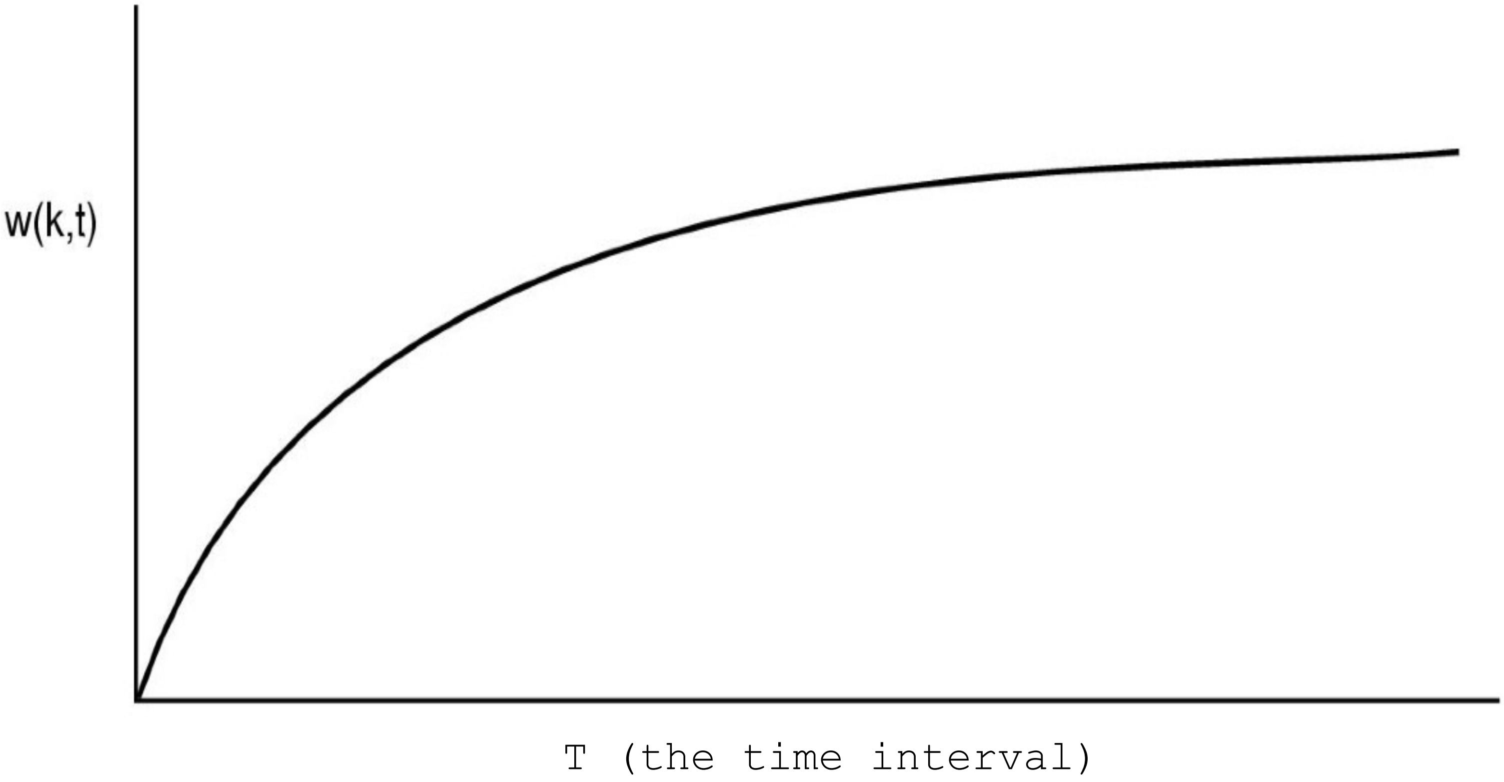
- **Based on prepaging (prefetching)**
  - Load pages before they are needed

- **Main idea:**
  - Try to identify the process's "working set"

- **How big is the working set?**
  - Look at the last K memory references
  - As K gets bigger, more pages needed.
  - In the limit, all pages are needed.

Working set page replacement

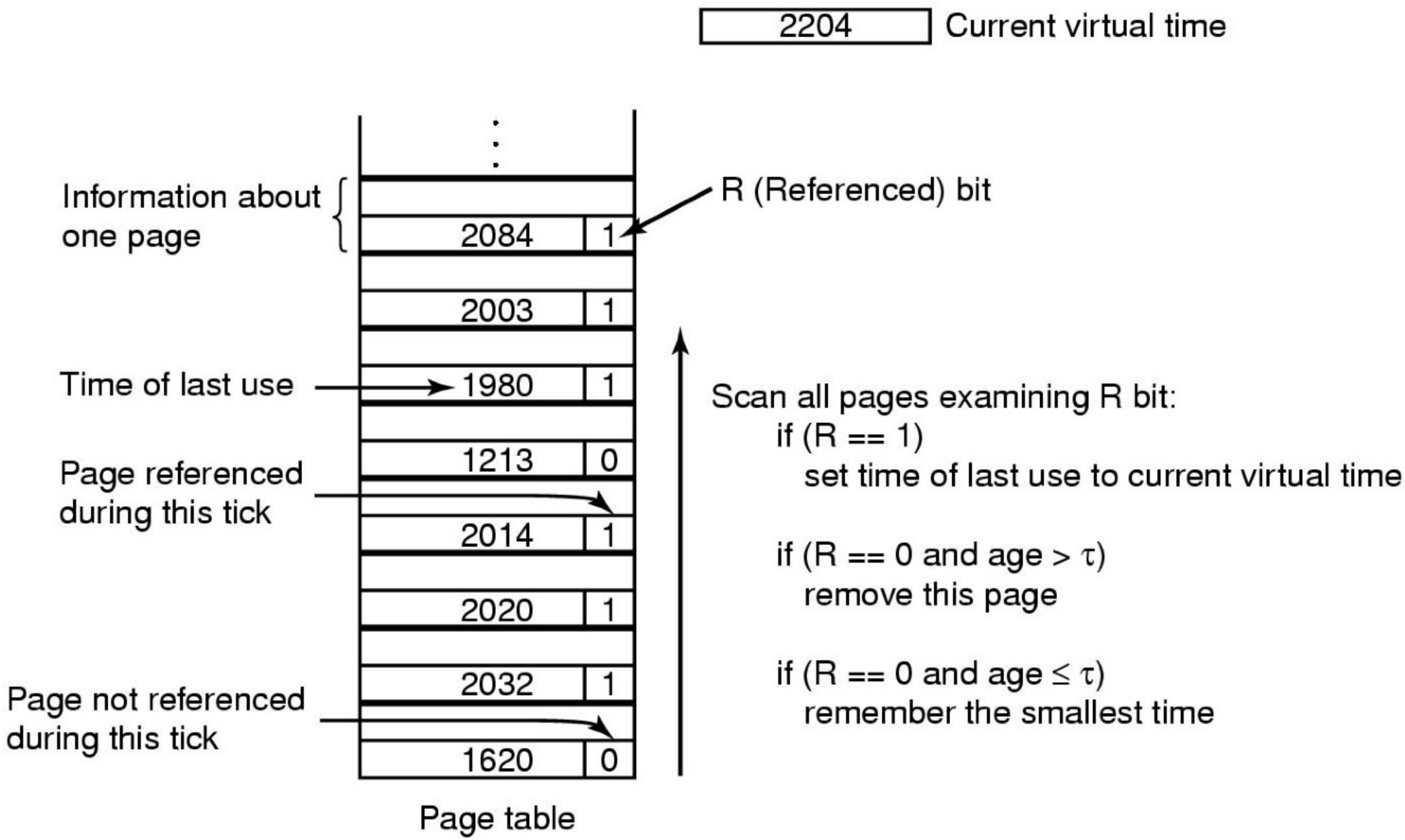
- The size of the working set for last k time intervals at time t:



Working set page replacement

- **Idea:**
  - Look back over the last T msec of time
  - Which pages were referenced?
    - **This is the working set.**
- **Current Virtual Time**
  - Only consider how much CPU time this process has seen.
- **Implementation**
  - On each clock tick, look at each page
  - Was it referenced?
    - **Yes: Make a note of Current Virtual Time**
  - If a page has not been used in the last T msec,
    - **It is not in the working set!**
    - **Evict it; write it out if it is dirty.**

Working set page replacement



WSClock page replacement algorithm

- An implementation of the working set algorithm
- All pages are kept in a circular list (ring)
- As pages are added, they go into the ring
- The "clock hand" advances around the ring
- Each entry contains "time of last use"
- Upon a page fault...
  - If Reference Bit = 1...
    - Page is in use now. Do not evict.
    - Clear the Referenced Bit.
    - Update the "time of last use" field.

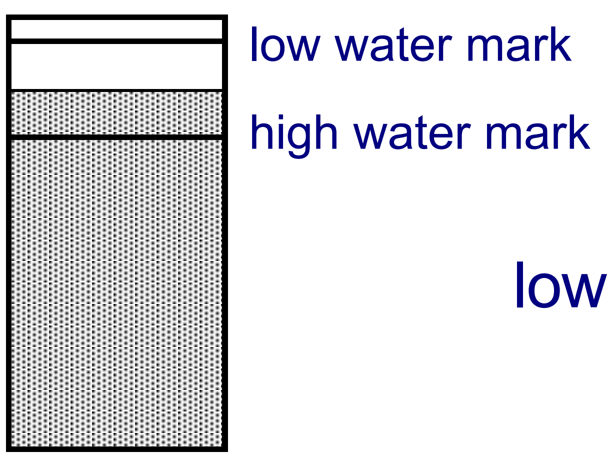
WSClock page replacement algorithm

- **If Reference Bit = 0**
  - If the age of the page is less than T...
    - **This page is in the working set.**
    - **Advance the hand and keep looking**
  - If the age of the page is greater than T...
    - **If page is clean**
      - **Reclaim the frame and we are done!**
    - **If page is dirty**
      - **Schedule a write for the page**
      - **Advance the hand and keep looking**

Proactive use of replacement algorithm

- Replacing victim frame on each page fault typically requires two disk accesses per page fault
- Alternative → the O.S. can keep several pages free in anticipation of upcoming page faults.

In Unix: low and high water marks



low < # free pages < high

Free pages and the clock algorithm

- **The rate at which the clock sweeps through memory determines the number of pages that are kept free:**
  - Too high a rate --> Too many free pages marked
  - Too low a rate --> Not enough (or no) free pages marked

- **Large memory system considerations**
  - As memory systems grow, it takes longer and longer for the hand to sweep through memory
  - This washes out the effect of the clock somewhat

The UNIX memory model

- **UNIX page replacement**
  - clock algorithm for page replacement
    - **If page has not been accessed move it to the free list for use as allocatable page**
      - **If modified/dirty → write to disk (still keep stuff in memory though)**
      - **If unmodified → just move to free list**
  - High and low water marks for free pages
  - Pages on the free-list can be re-allocated if they are accessed again before being overwritten