Nov 27, 2006
Lecture 14: Virtual Memory 2, and File System

December 4, 2006

1 ADMIN

ANNOUNCEMENT: the final exam is on Monday, December 18, from 2-3:30 in our usual classroom.

PLEASE start to read chapter 11 on File Systems.

2 REVIEW

- Q: What is the difference between Policy, Mechanism and Implementation? Illustrate all three concepts with a single example.

  A: Consider page replacement rule in demand paging. Policy is LRU Rule.

  Mechanism is the use of reference bit.

  Implementation is the second chance rule.

3 Demand Paging – LRU Implementations (contd.)

- We consider enhancements of the Reference Bit and the Second Chance Algorithm.
- Aging Of Bits. Another idea is to have automatic aging of reference bits. Every 100 ms or so, a timer interrupt will cause the OS to refresh (reset) the reference bits of each page.
- Another idea is to have a **reference byte** (i.e., 8 reference bits). Of course more bits is conceivable.

  When combined with the aging idea, we simply shift the reference byte to the right (hence losing the lowest order bit).

  E.g., a reference byte of 0000,0000 shows the page has not been referenced in the last 8 periods. Thus the reference byte is a historical sample of page references.

  So we can use the numerical value of this byte as the time stamp (the smallest numerical value is the one to be replaced).

- We can store a PAIR of bits, the usual reference bit and a new **modify bit**. The pair (r, m) has four possible values:

  $(0, 0)$: not recently used or modified. Best for replacement.

  $(0, 1)$: not recently used but modified. Need to write out the page if we replace it.

  $(1, 0)$: recently used but not modified.

  $(1, 1)$: recently used and modified. Least likely candidate for replacement.

  We view (r, m) as an integer between 0 and 3, representing the "class" of the page.

  We search for a page in the lowest class for replacement.

4 Frame Allocation Among Processes

- So far, in demand paging, we considered each process in isolation.
- In a multiprocessing environment, assuming that all the frames are allocated to the running process, it means that we must swap out ALL the pages in a frame when we do a context switch. This is very expensive.
- But first, let us review the basic SCHEDULING MODEL (§4.2., p.109) that we assume. We assume that processes are partitioned into two pools:

  – **active processes** which are partially loaded in main memory. These processes may be further subdivided into queues (Fig.4.5, p.109): a **ready queue**, and feeding into this ready queue are several queues (I/O queue, fork queue, etc). The fork queue might be waiting for a child to terminate.

  – **swapped processes** which are spooled on the disk.

  There are two schedulers:

  – **long-term scheduler** or job scheduler: chooses a process from the swapped process to make it active. This scheduler controls the **degree of multiprogramming**.

  – **short-term scheduler** or CPU scheduler: chooses a process from active processes to make it the current running process.

  The short-term scheduler runs relatively frequently (say, every 100 ms). as compared to the long-term scheduler (say, every 10 second).

  The longterm scheduler tries to get a good mix of I/O-bound and CPU-bound processes in the active list. In some systems, the longterm scheduler is missing or minimal.

- We may want to allocate a certain number of frames per process.

  A simple choice is equal allocation of frames per process.

  Another is proportional allocation – larger processes is allocated more frames.

- This leads to the distinction between LOCAL versus GLOBAL page replacement:

  Local is when we choose replacement page from the current allocated frames.

  Global is when we choose a replacement page from the entire pool of frames.

- **Thrashing** is the phenomenon of a process spending more time doing paging than executing its code.

  Thrashing can occur if we do global replacement without regard to actual performance (p.394 and also Fig 10.16.)

  By using only local replacement, we can limit thrashing to each process, but it also affects the performance of other processes.

- To avoid thrashing, we want to have a way to estimate the frames needed by each process.

  The fundamental idea to be exploited is the **locality assumption**. It says that each process executes in a locality (which is a small set of pages that are used together). This locality moves over the course of the process's execution.

  If we can estimate this locality, we can try to provide the necessary pages for the process. Then the process will not page fault until it goes to the next locality.

5 Working Set Model

- The **Working Set Model** is a method of estimating locality.

  We use a parameter Δ to define a **working set window**. The most recent Δ page references defines the set of pages called the **working set**.

- Suppose $S = (r_1, r_2, \ldots)$ is the reference string. Let S_t denote the prefix of S of length t $(t = 1, 2, 3, \ldots)$. E.g., $S_1 = (r_1)$, $S_2 = (r_1, r_2)$, etc. Let $WS(t)$ denote the last Δ page references in S_t.
- E.g., let $S = (2615, 7777, 5162, 3412, 3444, \ldots)$.

  If $\Delta = 5$ then $WS(8) = \{5, 7\}$ and $WS(12) = \{1, 2, 5, 6, 7\}$.
- Once the Δ is fixed, let WWS_i denote the **working set size** of the ith process. Then the demand is $D = \sum_i WWS_i$.

  If D exceeds the number of pages, we might get thrashing, and the degree of multiprogrogramming ought to be reduced.

  We could also allow the Δ be a function of the process: Δ_i for the ith process.

6 OS Examples of Demand Paging

CONSIDER demand paging in Windows XP.

- When a process is created, it is given a minimum and maximum for its working set.

  Typically, minimum is 50, maximum is 345.
- The Virtual Memory manager maintains a list of free frames.

  When a page fault occurs for a given process, and its maximum allocation is not reached, and there is a free frame, we allocate a new frame.

  Otherwise we use a local replacement algorithm.
- If the free memory falls below a threshold, it uses a **automatic working-set trimming** to restore this threshold: for each process, see if we can deallocate some pages so that it reaches its worksing-set minimum.

7 File Systems

- This is the most visible part of the OS! Files is the non-volatile part of computer memory, and hence reside in secondary storage.
- The Physical Memory in the disk is divided into logical user-defined units called **files**.
- Files are organized into structures, which are usually **hierarchies**.

  Files have data of various types: E.g., source, binary, graphics, sound, etc.
- **Files attributes**

  1. Name: human readable form
  2. Identifier: unique tag within whole system
  3. Type: ascii, binary, graphics, etc
  4. Location: physical address
  5. Size
  6. Protection:
  7. Time, Date, Ownership: creation, last modification, last use, etc. Useful for protection, security, monitoring and search.

  Remark: file name has 2 parts (second part is the extension, indicating type for some systems)
- There is a file directory, also in secondary storage, for this info.

- File Operations:

  1. Create: find space, create directory entry
  2. Write: depends on current location
  3. Read: depends on current location
  4. Seek: change the current location
  5. Delete:
  6. Truncate:

- Open and Close:

  Since read/write/seek, etc, has nontrivial initialization cost, most OS requires that we must first OPEN a file before we can do these operations. Hence we also need to CLOSE a file when done.

  There is a list of OPENED files (per process, and system-wide).

  The per-process list of OPENED files points to the system-wide list.

  We need these info for the per-process opened files:

  1. File pointer: current location
  2. File-open count: closes file when it reaches 0
  3. Memory location: location in main memory about file
  4. Access rights: