Slide 1

Administrivia

- Reminder: Homework 4 due Wednesday.

Slide 2

Minute Essay From Last Lecture

- (Most people were more or less in the ballpark.)

Paging — Operating System Versus MMU

- Some aspects of paging are dealt with by hardware (MMU) — translation of program addresses to physical addresses, generation of page faults, setting of R and M bits.
- Other aspects need O/S involvement. What/when?

Slide 3

Paging — Operating System Involvement

- Process creation requires setting up page tables and other data structures. Process termination requires freeing them.
- Context switches require changing whatever the MMU uses to find the current page table.
- And of course it's the operating system that handles page faults!
- Some details . . .

Slide 4

Slide 5

Processing Memory References — MMU

- Does cache contain data for (virtual) address? If so, done.
- Does TLB contain matching page table entry? If so, generate physical address and send to memory bus.
- Does page table entry (in memory) say page is present? If so, put PTE in TLB and as above.
- If page table entry says page not present, generate page fault interrupt. Transfers control to interrupt handler.

Slide 6

Processing Memory References — Page Fault Interrupt Handler

- Is page on disk or invalid (based on entry in process table, or other o/s data structure)? If invalid, error — terminate process.
- Is there a free page frame? If not, choose one to steal. If it needs to be saved to disk, start I/O to do that. Update process table, PTE, etc., for "victim" process. Block process until I/O done.
- Start I/O to bring needed page in from swap space (or zero out new page). If I/O needed, block process until done.
- Update process table, etc., for process that caused the page fault, and restart it at instruction that generated page fault.

Slide 7

Processing Memory References — Details Still To Fill In

- How to keep track of pages on disk.
- How to keep track of which page frames are free.
- How to “schedule I/O” (but that’s later).

Slide 8

Keeping Track of Pages on Disk

- To implement virtual memory, need space on disk to keep pages not in main memory. Reserve part of disk for this purpose (“swap space”); (conceptually) divide it into page-sized chunks. How to keep track of which pages are where?
- One approach — give each process a contiguous piece of swap space. Advantages/disadvantages?
- Another approach — assign chunks of swap space individually. Advantages/disadvantages?
- Either way — processes must know where “their” pages are (via page table and some other data structure), operating system must know where free slots are (in memory and in swap space).

Slide 9

Paging — Other Design Issues/Choices

- Demand paging versus prepaging.
- Global versus local allocation.
- “Paging daemon” that tries to keep a supply of free page frames.
- What to do if page to be replaced is waiting for I/O — probably trouble if we replace it anyway. (Solutions include “locking” pages, or doing all I/O to o/s pages and then moving data to user pages.)

Slide 10

Modeling Page Replacement Algorithms

- Intuitively obvious that more memory leads to fewer page faults, right? Not always!
- Counterexample — “Belady’s anomaly”, sparked interest in modeling page replacement algorithms.
- Modeling based on simplified version of reality — one process only, known inputs. Can then record “reference string” of pages referenced.
- Given reference string, p.r.a., and number of page frames, we can calculate number of page faults.
- How is this useful? can compare different algorithms, and also determine if a given algorithm is a “stack algorithm” (more memory means fewer page faults).

Slide 11

Sharing Pages

- Shared pages can be useful, but can also present problems.
- Multiple processes running the same program is relatively easy (why?) but has one potential downside (what?)
- UNIX `fork` system call is — interesting in this context. POSIX definition says that child process's address space is basically a copy of the parent's address space. What's the easy-to-implement way to do this? What downside does that have in current systems? Is there a way to reduce its impact? And why duplicate in the first place?

Slide 12

Sharing Pages and `fork`

- Duplicating pages is easy but inefficient, especially if the child process is going to call `execve` or something similar right away. Some systems use "copy-on-write" to improve efficiency.
- Why did the people who designed UNIX require this duplication . . . Possibly because it makes some things easy (such as setting up parent/child pipes) and wasn't very costly when designed. Windows's system call for creating processes takes a different approach. Maybe that's better!

Slide 13

Sharing Pages, Continued

- One use for shared pages is multiple processes running the same program.
- What about sharing code at a level below whole programs (UNIX “shared libraries”, Windows DLLs)? Seems attractive; are there potential problems?

Slide 14

Shared Libraries

- One attraction is somewhat obvious — if code for library functions (e.g., `printf`) is statically linked into every program that uses it, programs need more memory — seems wasteful if processes can share one copy of code in memory.
- Another attraction is that library code can be updated independently of programs that use it. (Is there a downside to that?)
- How to make this happen . . . At link time, programs get “stub” versions of functions. References to real versions resolved at load time.

Slide 15

Minute Essay

- Another story from long ago: Once upon a time, a mainframe computer was running very slowly. The sysadmins were puzzled, until one of them noticed that one of the disk drives seemed to be very busy and asked “which disk are you using for paging?” The answer made everyone say “aha!” What was wrong (to make the system so slow)?
- Could anything like this still happen?

Slide 16

Minute Essay Answer

- The disk being used for paging was the one that was very busy. So, mostly likely the system was spending so much time paging (“thrashing”) that it wasn’t able to get anything else done. Usually this means that the system isn’t able to keep up with active processes’ demand for memory.
- Memory sizes have increased to a point where the odds aren’t as good as they were. But a few years ago we did run into problems with the machines in one of the classrooms trying to run both Eclipse and a Lewis simulation, and then more recently with some of them attempting to run a background program that asked for more memory than its author intended.

Slide 17

Minute Essay

- FIXME

Slide 18

Minute Essay Answer

- FIXME