Slide 1

Administrivia

- Next homework on Web soon; probably will be due next Thursday.

Slide 2

Minute Essay from Last Lecture

- A couple of people mentioned calculating page table sizes. So, a short review ...

Slide 3

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 4

Paging — Other Design Issues

- In deciding which page to replace, consider all pages ("global allocation"), or just those that belong to the current process ("local allocation")?

  Generally, global approach works better, but not all page replacement algorithms can work that way (e.g., WSClock). Hybrid strategy — combine local approach with some way to vary processes' allocations.
- What happens if combined working sets of all processes don't fit into memory? "Thrashing".

  What to do? temporarily "swap out" some processes, or other forms of "load control".
- Maintaining a supply of free frames — desirable, could do by having a "paging daemon" in background.

Slide 5

Paging — Other Hardware Issues

- What if page to be replaced is waiting for I/O? probably trouble if we replace it anyway.
- One solution — allow pages to be “locked”.
- Another solution — do all I/O to o/s pages, then move to user pages.

Slide 6

One More MM Strategy — Segmentation

- Idea — make program address “two-dimensional” / separate address space into logical parts. So a virtual address has two parts, a segment and an offset.
- To map virtual address to memory location, need “segment table”, like page table except each entry also requires a length/limit field. (So this is like a cross between contiguous-allocation schemes and paging.)

Slide 7

Segmentation, Continued

- Benefits?
  - Nice abstraction; nice way to share memory.
  - Flexible use of memory — can have many areas that grow/shrink as required, not just heap and stack — especially if we combine with paging.
- Drawbacks?
  - External fragmentation possible (can offset by also paging).
  - More complex.
  - "Paging" in/out more complex — issues similar to with contiguous-allocation.

Slide 8

Memory Management in Windows

- Apparently very complex, but basic idea is paging.
- Intraprocess memory management is in terms of code regions (some shared — DLLs), data regions, stack, and area for o/s. "Virtual Address Descriptor" for each contiguous group of pages tracks location on disk, etc.
- Memory-mapped files can make I/O faster and allow processes to (in effect) share memory.
- Demand-paged, with six (!) background threads that try to maintain a store of free page frames. Page replacement algorithm is based on idea of working set.
- (Also see comment on p. 823.)

Slide 9

Memory Management in Unix/Linux

- Very early Unix used contiguous-allocation or segmentation with swapping. Later versions use paging. Linux uses multi-level page tables; details depend on architecture (e.g., three levels for Alpha, two for Pentium).
- Intraprocess memory management is in terms of text (code) segment, data segment, and stack segment. Linux reserves part of address space for o/s. For each contiguous group of pages, "vm_area_struct" tracks location on disk, etc.
- Memory-mapped files can make I/O faster and allow processes to (in effect) share memory.
- Demand-paged, with background process ("page daemon") that tries to maintain a store of free page frames. Page replacement algorithms are mostly variants of clock algorithm.

Slide 10

Minute Essay

- Consider the following partial program:

```
double a[N][N];
int i, j, k;
for (i = 0; i < N; ++i) {
    for (j = 0; j < N; ++j) {
        a[i][j] = i + j;
    }
}
```

- Reversing the order of the loops can have a big effect on execution time. Why? (Actually there are two explanations, depending on the size of N.)