

CS162 – Section # 7, 03/11/2003

Rodrigo Fonseca

Announcements

Midterm !

Nachos phase 2: implement system calls to support multi-programming

Design doc is due next Tuesday

Next week: design review sessions, there will be a Web signup form for times,

Code is due right after SP (April 3rd)

Today: review of paging/segmenting/...

Virtual Memory

Goals for a (virtual) memory system:

- Multiprogramming
- Relocation
- Protection
- Sharing (code, memory)
- Illusion of large memory

How to allocate main memory to programs?

1. One program at a time, at fixed position
 - Multiprogramming expensive
 - No relocation (not needed)
 - No protection (can damage OS)
 - No sharing
 - Memory constrained to size of physical mem - OS

Other comments:

+ simple

How to have more programs at a time?

2. Multiprogramming with static software relocation
 - Multiprogramming achieved (possible without need for swap)
 - Relocation on loading (always same location after that)
 - No protection (can damage other programs and OS now)
 - Sharing difficult/impractical
 - Memory still constrained to size of physical memory

Other:

- External fragmentation

Virtual versus Physical Address Spaces
--

3. Dynamic relocation: base + bound

Virtual vs Real (physical) address space

Base and Bound Registers

Part of the PCB

Set by the OS! (inaccessible to programs)

- Multiprogramming cheap (just change base+bound)

- Relocation greatly simplified (by hardware provisions)
- Protection can be enforced
- Sharing still difficult (impractical)
- Memory still constrained to size of physical memory

Other:

- Memory allocation complex (where to put a new program? How to 'grow' a program?)
- External fragmentation
- Overhead for each access to memory

4. Pure Segmentation

Variable sized region of contiguous memory.

Generalization of base and bound -> Segment table

Correspond to logical divisions of programs

Hardware provides STBR, seg table (may or may not), protection

Address Space:

Segment table is *per-process*

- Multiprogramming easy
- Relocation
- Protection (even better granularity – R/W/E per segment)
- Sharing (of code, of data) possible
- Illusion of memory greater than physical memory

Other:

- + Efficient for sparse address spaces
- Complex memory allocation (best fit, first fit, re-shuffling)
- Each segment has to be:
 - Contiguous
 - Smaller than memory
- Overhead for access
- Space for segment table

5. Paging

Solve the complex allocation in segments: divide memory in equal sized blocks – page frames.

Allocation is simple: just find *any* free page frames

2 registers: PTBR and Ptsize

Multiprogramming easy

Relocation

Protection

Sharing possible

Illusion of memory greater than physical memory

Other:

- Internal fragmentation
- Large address spaces: large page table!

Tradeoffs in page size.

5. Multilevel translation

Divide the address space in multiple levels.

If your table gets very large, you may subdivide it.

One way to combine the best of both worlds.

Segments are nice because they have a direct relationship with logical divisions of programs (nice granularity for protection and sharing for example). Are also efficient for sparse address spaces.

Paging is nice for it is simple, but page table can get huge.

Nachos Phase 2 – Multiprogramming/System Calls/Lottery Scheduler

System call: a function called by an application to invoke a kernel service. Provides the interface between programs and the OS.

The system calls are documented in syscall.h

Part 1: File system calls.

Part 2: Support for multiprogramming. How can different programs share (virtual) memory? Each user process will have its own page table.

Part 3: Exec, join, exit, for process control

Part 4: Lottery scheduler. Should require a few changes to Priority Scheduler.

Remember, you must make your syscall implementations bullet proof, in the sense that user processes cannot do any damage to the OS or to other processes, by passing any kind of bad or corrupt arguments. It is your responsibility to do a sanity check on all arguments, and also **test** it thoroughly, so as to guarantee the robustness of the system.