

CS162, Spring 2002
Section 6
Steve Martin

More Project Info

- Congrats on finishing your first Nachos project!
 - Project 2 will be released very soon.
- **DESIGN DOCUMENT REVISIONS AND QUESTIONS DUE TOMORROW.**
 - Unless you used slip days. . .

What's in a Final Design Document?

- Revisions of your first design document
 - If you had to change anything, you should write a REVISION SECTION for your changes inside the design for that section.
- Boat problem
 - If you checked for your end condition using variables init'd in begin inside of an itinerary. . .
 - Write a revision section that discusses how you would design a solution that doesn't do this.
- Design Questions
 - Answer the two questions at the bottom of the project spec.
 - Don't have to write pages—if you did the project and came to section, these will be trivial.

Questions From Lecture?

Today's Menu

- Kernel Space vs. OS Space: System Calls
- How are processes created?
- Address Translation
- Quiz

Kernel Space vs. OS Space: System Calls

- Kernel Mode vs. OS Mode: what is it?
 - This is review. . . you tell me.
- How do we cross this border?
 - Unintentionally
 - Errors!
 - Also known as exceptions
 - Examples:
 - Bus Error (bad address, basically)
 - Segmentation Fault (c

- Intentionally: System Call

QUESTION: If the System Calls are OS functions, how does the program know about them?

ANSWER: Have to link in OS services on compile. In Unix, the interface file is sys.h

QUESTION: What happens when a System Call occurs?

ANSWER:

- Program triggers System Call exception!
- Set processor status to kernel mode
- Save program state
- Switch to handler routine in OS
 - How do we know which one?
 - There is a table of exception values in the OS that we need to look at!
 - Will tell us, based on exception type, where to go.
 - In linux: entry.s
- Execute routine

- What about arguments--i.e. open(char*)?

- Registers
 - Simple values
- Translate memory addresses!
 - Pass OS an address in user space
 - OS must copy whatever the arguments are across boundaries.

QUESTION: See anything dangerous here? What else should the OS do?

ANSWER: Heck yeah it's dangerous . . . OS must carefully check all addresses!

- OS must translate this to a physical address in order to access arguments.
 - How?
 - Base and Bounds, Paging, Segmenting, etc
 - More on this later

How are Processes Created?

- As was mentioned in class, we use fork and exec
 - These are both system calls (why?)

QUESTION: What does fork do?

ANSWER: Stops the current process, create an exact copy, change register/set PID, put new process on ready list, resume original.

- Important: how to distinguish forked copies?
 - PID, register

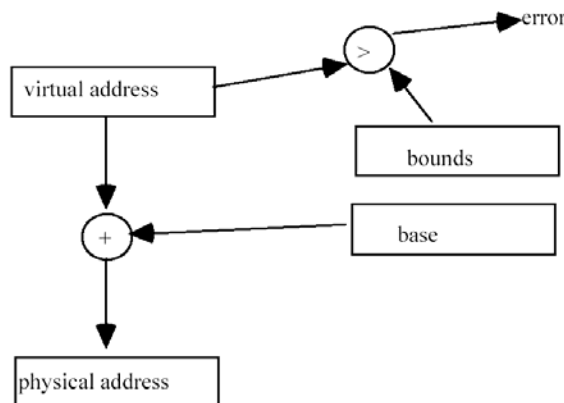
QUESTION: What does exec do?

ANSWER: Replaces the program in the current process and restarts from 0.

- Example: what happens when type 'exec ls' from your login. Why?
- Is this used?
 - you bet
 - UNIX: there is one root process. . .all other processes are forked from this process!
 - When you exec something, we fork the shell!
- How do we make this process more efficient?
 - Ideas?
 - Vfork();

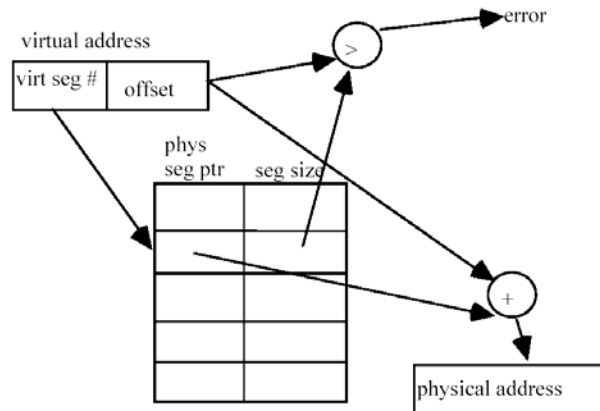
Address Translation

- Base and Bounds



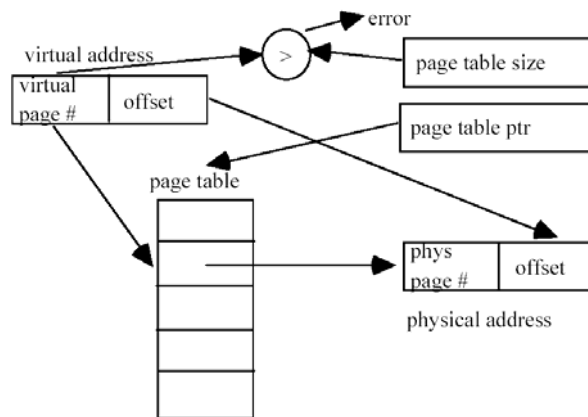
- Have a base and bounds value for the memory addresses of each program in physical memory.
- Use to change addresses on the fly from applications.
- If address out of range of bounds:
 - Segment fault!
- Pros:
 - Fast, easy
- Cons:
 - Hard to share memory!
 - Not enough abstraction here. . .how would you share just one piece of memory?
 - There can be external fragmentation in your memory.
 - What happens when you allocate and quit a lot of applications?
 - Fix: have fixed size bounds for everyone!

- Does this work?
- Fixed size to heap and stack
 - We run into problems when we need to resize our memory
- Segmenting (note that this segment table is missing the segment start column)



- Instead of having ONE set of base and bounds, why not have a table?
- Take virtual address, divide up into segment number and offset number
 - Do a lookup into table, get segment start
 - Add offset to segment start to form physical address.
 - Check to make sure we're within the segment size.
 - Get data from physical memory with new address!
- Pros:
 - Easy to share memory!
 - Need to add PERMISSIONS/PROTECTION BITS to your segment table.
 - Easy to resize memory. . . maybe
- Cons:
 - Still have external fragmentation
 - Need to save segment table for each process
 - Not too big, but still is work.

- Paging



- Segmenting with fixed size segments (plus more)
 - Divide up physical and virtual memory into pages OF THE SAME SIZE.
 - Note: physical and virtual addresses do NOT have to have the same number of address bits.
 - Why?
 - Doing actual address translation:
 - Each virtual address is divided up into page number and offset.
 - This is based on page size.
 - The page number is used to lookup into the page table.
 - The physical address of page is retrieved from the page table entry.
 - Physical address of page + offset = physical memory location.
 - Load from memory can now proceed with this address . . . or can it?
 - What if the physical memory is smaller than the virtual memory? (yeah, it will be!)
 - Not all pages in memory.
 - What if page is not in memory?
 - Page Fault! (exception!)
 - How do we know if a page is memory?
 - Need extra bits in page table. . .Valid Bit.
 - What happens on a page fault?
 - Load a page if memory free
 - If not, kick someone out! (more on this later)
 - Pros of paging:
 - Solves allocation—we do everything in fixed block
 - Easy sharing, with Protection Bits
 - Cons
 - Complex
 - Slow—have to look up every address!
 - Can fix with buffering! More on this later.
 - Page tables have to be saved somewhere!
 - Example: 32 bit addresses, 4k pages means 4meg per page table, PER PROCESS.
 - Adds to context switches
 - Internal fragmentation
 - Pages fixed size. . .if page size is X bytes, and we need X+1, how much memory is wasted?
- . . . Much, much more to come on this topic.

Quiz