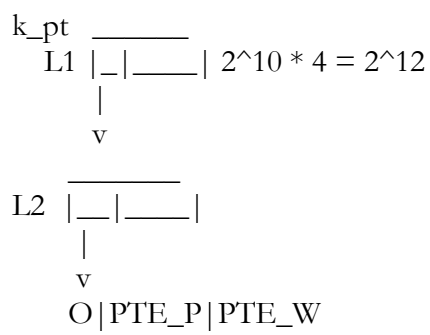
CS61 Scribe Notes — 11/7/13 — David Kaufman

Update pset to get new code!!!



- pagetable belonging to the process must allow the kernel to run
- kernel pagetable replaces the garbage pagetable that the process had

Let's do fork:

- fork ( )
- what is it and why is it
- a process believes that it has complete control over all of the system's resources
- processes have direct analogies to hardware devices
- with fork, a process can duplicate a computer

fork ( )

- create new processes
- duplicates the current process & its resources
- memory (+ registers + processes status) resources are duplicated and I/O device resources (files) are shared
- return to parent - child ID, to child: 0

Historical

- Originally, the original version of unix had support for one process at a time
- still a shell
- eventually hacked the system from supporting one processes to 1.5
  - shell main process and it would save the shell into memory and start executing what command you typed
  - overwrote the exit command to take the state of the shell and putting that state into memory to replace the current process
  - infinite loop -> reboot computer because the shell would be suspended in memory
  - turned into multi process system with 20 lines of assembly

kernel.c:fork

- how to do?
- need to copy the registers
- memcpy into process 2's registers from the current process's registers (or just do by assignment)
- need to return something to each process
- change eax to 0 for process 2

- set eax to 2 (the process id) for the current process

doesn't look like the process is being run

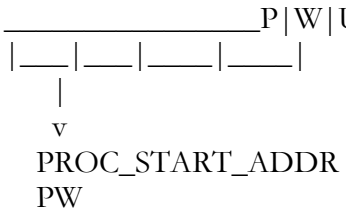
- in order to set up the process's state, set to runnable
  - not getting far (the system instantly reboots)
  - no pagetable!!!
  - use page_alloc_unused

VOTE: skip to end and explain

- copy registers, set return values
- copy the address space
- what do we want the pagetable to look like for the new process?

PROCESS 1

L1 -> L2



PROCESS 2

L1-> L2

give processes different pagetable addresses (to maintain process isolation)

the process can modify its own memory (not another process's memory)

to preserve process isolation, anything that can be written is copied

- copy address space for kernel
- for the user address space, go over and copy every page
- code loops through user address space
- lookup, allocate, copy, map
- assumption that that page is mapped (memcpy)
- casting vam.pa assumes that the current pagetable has an identity mapping for the physical address
  - that the virtual address maps to the same physical address
  - the only thing you can do with physical addresses is put them into page tables
  - everything else is a virtual address
  - so we can only pass a physical address into memcpy if an only if this physical address works as a virtual address

where do we copy the kernel?

- in pagetable alloc (marks everything as kernel memory)

BREAK

Virtual memory tricks

- faulting instructions
- faulting instruction and address are often not the same
- need to typedef a function pointer type
- no OS makes process code writable by default
- the process itself could accidentally write over its own code

- stack smashing attack

MMAP

- PROT_READ
- PROT_WRITE
- MAP_SHARED
- MAP_PRIVATE - process has illusion of own private copy of file (if it modifies it, a new file is created)

500mb of memory

how much of the process's memory are we using?

- turn off all of virtual memory space

1. box per day
2. calculate P(box used on that day)
3. #active boxes per day = P(box used per day) x #boxes

- VMware - people who bought machines wanted to use each one for multiple services
- which OS should get control of the CPU?
- what VMware does is it detects whether an operating system is doing useful work by detecting memory it is accessing (sampling strategy)
  - borrows memory back and runs that OS less often