Project 4, Part II

- User address space changes
  - also implement Copy_To_User, Copy_From_User
- Demand paging: let the user stack grow
- Paging to disk

User address space changes

- Changes in uservm.c; copy from userseg.c
- Most changes in Load_User_Program
- Also need to write Destroy_User_Context and Switch_To_Address_Space
- Set up new paging directory, reference from User_Context
- Segmentation changes: base at 0x80000000
- Code and data (read from .exe file) at bottom of user space
- Stack and argument block (constructed in kernel) at top of user space

Demand paging

- User stack at top of address space; there is tons of virtual memory between bottom of stack and top of code
- With nested function calls, stack (easily) grows beyond 4K
- Modify page fault handler to allocate an extra stack page when fault is within a page of user stack pointer
- What if a function has 8K of local variables?

Paging to disk

- Much work is already done in Alloc_Pageable_Page()
- GeekOS paging file contains only pages that have been paged out, not those resident in memory
- You write the functions that manage the paging file

Paging file management functions (1)

- Initialization needs
  - find size and location of paging file (Get_Paging_Device)
  - initialize data structure for free space
- Functions in bitset.c are useful to track free space (and will come back in Project 5)
- Read and write paging file, using Block_Read and Block_Write
  - These handle one 512-byte sector at a time; 8 sectors to a page
  - Interrupts must be on to do disk i/o!

Paging file management functions (2)

- Decide what page to page out when physical memory is full
  - "Pseudo-LRU" (least recently used); algorithms described in text
  - Use accessed bit to determine what pages have been used (CPU sets it, you have to clear it)
  - Clock algorithm is good but name is misleading: it doesn't have anything to do with time
- Page fault handler must recognize (and bring back) pages that have been paged out