CS 3204
Operating Systems

Lecture 6

Godmar Back

Virginia Tech

Announcements

- Posted stack.cc example on website
- Project 0 is due Feb 1, 11:59pm
- This project is done individually
- Curator instructions will be posted on website today
  - Vijay needs to upload class list

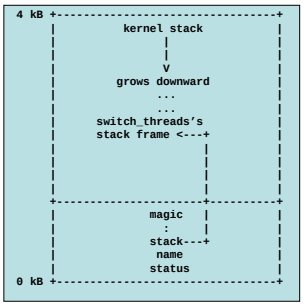
Overview for today

- Finish
  - How does OS execute processes?
  - How do kernel & processes interact
  - How does kernel switch between processes
  - Implementing Processes (PCB)
- Process States
- Next:
  - Scheduling of threads
  - Synchronization of threads

Pintos Kernel Stack

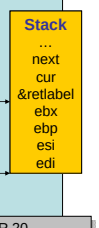
- One page of memory captures a process's kernel stack + PCB
- Don't allocate large objects on the stack:

```
void
kernel_function(void)
{
    char buf[4096]; // DON'T
    // KERNEL STACK OVERFLOW
    // guaranteed
}
```



Pintos Context Switch

```
switch_threads:  // switch_thread (struct thread *cur, struct thread *next)
        # Save caller's register state.
        # Note that the SVR4 ABI allows us to destroy %eax, %ecx, %edx,
        # but requires us to preserve %ebx, %ebp, %esi, %edi.
        pushl %ebx; pushl %ebp; pushl %esi; pushl %edi

        # Get offsetof (struct thread, stack).
        mov thread_stack_ofs, %edx

        # Save current stack pointer to old thread's stack.
        movl SWITCH_CUR(%esp), %eax
        movl %esp, (%eax,%edx,1)        } cur->stack = esp

        # Restore stack pointer from new thread's stack.
        movl SWITCH_NEXT(%esp), %ecx
        movl (%ecx,%edx,1), %esp        } esp = next->stack

        # Restore caller's register state.
        popl %edi; popl %esi; popl %ebp; popl
        ret
```




```
#define SWITCH_CUR 20
#define SWITCH_NEXT 24
```

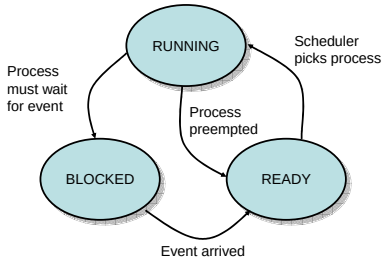

External Interrupts & Context Switches

```
intr_entry:
        /* Save caller's registers. */
        pushl %ds; pushl %es; pushl %fs; pushl %gs; pushal

        /* Set up kernel environment. */
        cld
        mov $SEL_KDSEG, %eax            /* Initialize segment registers. */
        mov %eax, %ds; mov %eax, %es
        leal 56(%esp), %ebp             /* Set up frame pointer. */

        pushl %esp
        call intr_handler    /* Call interrupt handler. Context switch happens in there*/
        addl $4, %esp
        /* FALL THROUGH */
intr_exit:                      /* Separate entry for initial user program start */
        /* Restore caller's registers. */
        popal; popl %gs; popl %fs; popl %es; popl %ds
        iret  /* Return to current process, or to new process after context switch. */
```

Context Switching: Summary

- Context switch means to save the current and restore next process's execution context
- Context Switch != Mode Switch
  - Although mode switch often precedes context switch
- Asynchronous context switch happens in interrupt handler
  - Usually last thing before leaving handler
- Have ignored so far when to context switch & why → next

Process States



- Only 1 process (per CPU) can be in RUNNING state

Process Lists

- All ready processes are inserted in a ready list
  - Running process typically not kept on ready list
  - Can have multiple ready lists, e.g., one for each priority class
- All blocked processes are kept on lists
  - List usually associated with event that caused blocking
- A lot of scheduling involves clever ways of manipulating lists

Process Events

- What's an event?
  - External event:
    - disk controller completes sector transfer to memory
    - network controller signals that new packet has been received
    - clock has advanced to a predetermined time
  - Event that arise from process interaction:
    - a resource that was previously held by some process is now available (e.g., lock_release)
    - an explicit signal is sent to a process (e.g., cond_signal)
    - a process has exited or was killed