Precept 3: Process Management

COS 318: Fall 2017

Project 3 Schedule

- Precept: Monday 10/16, 7:30pm
  - (You are here)
- Design Review: Monday 10/23
- Due: Sunday, ~~10/29~~ 11/05, 11:55pm

Precept Overview

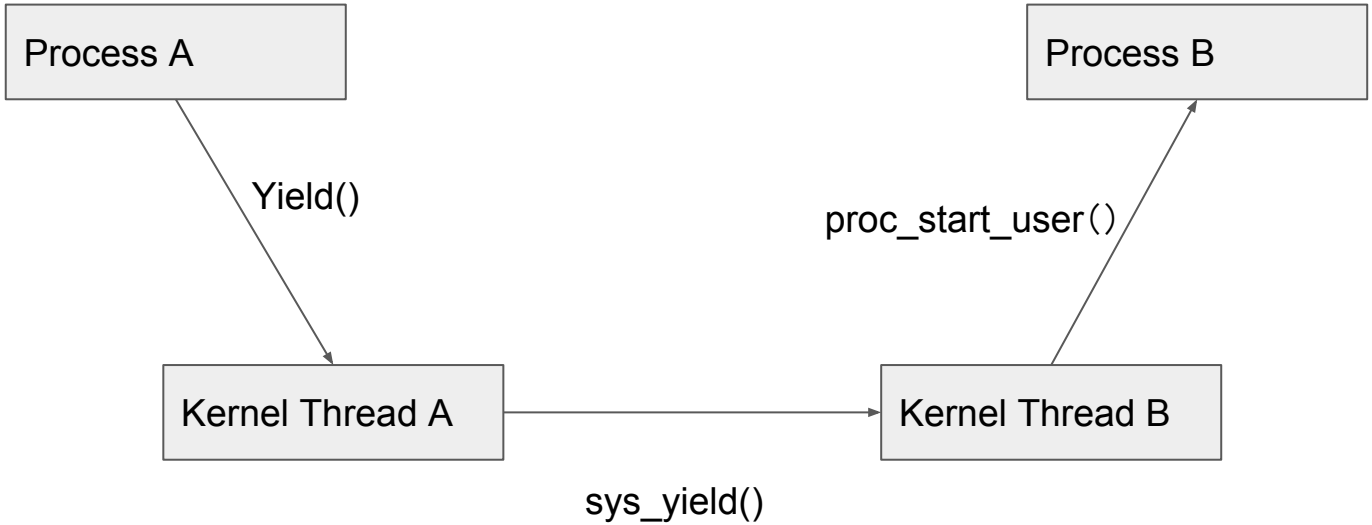
- User Processes and Threads
- Trap and Interrupt Handling
- Copy-on-write and Fork
- Project Specific Topics

User Processes and Threads

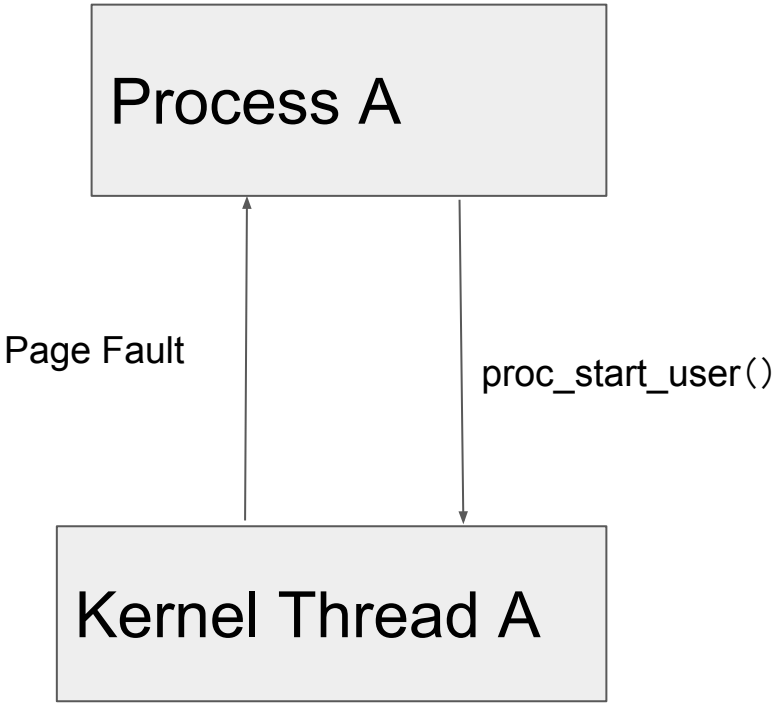
User Process

- Each user process has a corresponding kernel thread.
- When user process tries to enter the kernel mode, it first traps into its kernel thread.

Example : sys_yield



Example : Page Fault



Thread Context Switch

- Change from one kernel thread to another.
- Save the old context and fetch the new.
  - Save the old: EIP, ESP, EDI, ESI, EBX, EBP;
  - Fetch the new; (where should the new EIP be placed?)
  - < 20 lines assembly code.

Kernel Thread

- TCB: thread control block;
  - State: Running, Ready, Dead;
  - Prev: the previous TCB;
  - Next: the next TCB;
- Sleeping Queue: contains all TCBs that can be woken up by current threads;

Tip

- Ready Queue: there is one ready queue storing all ready TCBs. *TQueuePool[NUM_IDS]*
- Please read files in `kern/proc/PProc/PProc.c`
  - proc_start_user()
  - proc_create ()
- Read the assembly file: `kern/dev/idt.S`

Traps and Interrupts

Terminology Dump

- Interrupt: caused by hardware event / `int` instruction
- Exception: caused by currently running code
  - Trap: software defined exception
  - Fault: an “error” that is typically recoverable
  - Abort: an “error” that is typically non-recoverable

Handling Interrupts / Exceptions

- Can’t let user code enter kernel arbitrarily
  - Can only enter kernel through interrupts / exceptions
- CPU looks up appropriate handler in the Interrupt Descriptor Table (IDT)
- Need to save / restore previous state

Interrupt Descriptor Table

- Table of entry points to exception handlers
  - Contains the EIP and CS values to load
- CPU uses interrupt vector as index into the table
- Location of IDT: Determined by IDTR

Switching to Kernel mode

- Need to save state before handling the exception
  - Must also be independent of user level code
- Solution: define a kernel-only stack and save / restore state from that
- Location of stack: in Task State Segment (TSS)

System Calls

- Asks the kernel to perform some task
- Invoked with `int 0x30` in our system
- Number: `%eax`, arguments: other registers
- Error No.: `%eax`, return values: other registers

Copy-on-write and Fork

Fork System Call

- Create Child Process;
  - Duplicate Parent Process;
  - Copy all parent memory state;
- Return 0 in the child process and child's PID in the parent process.
- Example: http://www.csl.mtu.edu/cs4411.ck/www/NOTES/process/fork/create.html

Copy-on-write

- Map the same pages in the two processes;
- Set page read-only and SET COW bit = 1 in PTE;
- When one process tries to write the page,
  - Page Fault Happen
  - Assign a new page and copy the old page content;
  - The new page and the old page becomes writeable.

Project Specific Topics

Implementing Fork

- Need to create your own layer
  - Follow the structure of the other layers!
- Import what you need from MPTIntro
- Import your new layer's functions to write `sys_fork`

Design Review

- When switching from one kernel thread to another, which registers must be stored on the stack, and in what order? Where will the return address be located?
- When crossing the user mode - kernel mode boundary, what state must be saved, and where is it saved?
- How are system calls invoked, and how does the kernel determine which system call handler to use?
- Prepare an outline of how you plan to implement the `sys_fork` system call.

Questions?