Project 4
Inter-Process Communication and Process Management

COS 318

Fall 2016

Project 4: IPC and Process Management

- Goal: Add new IPC mechanism and process management to the kernel.
- Read the project spec for the details.
- Get a fresh copy of the start code from the lab machines. (/u/318/code/project4/)
- Start as early as you can and get as much done as possible by the design review.

Project 4: Schedule

- Design Review:
  - Thursday 11/17
  - Sign up on the project page;
  - Please, draw pictures and write your idea down (1 piece of paper).
- Due date: Tuesday, 11/22, 11:55pm.

Project 4: Overview

- Implement a spawn system call.
- Implement inter-process communication using message boxes.
- Implement a handler for the keyboard interrupt.
- Implement a kill system call.
- Implement a wait system call.

Design Review

- Design Review:
  - Answer the questions:
    - ✓ **Process Management:**
      - ✧ How will your spawn, wait, and kill work?
      - ✧ How will you satisfy the requirement that “if a process is killed while blocked on a lock, semaphore, condition variable or barrier, the other processes which interact with that synchronization primitive [will] be unaffected?
    - ✓ **Mailboxes:**
      - ✧ What fields will the structs need?
      - ✧ Which synchronization primitives will you use?

Implementation Checklist

- `do_spawn`: creates a new process
- `do_mbox_*`: mbox functions to enable IPC
  - `open, close, send, recv, is_full`.
- Handle keyboard input:
  - `putchar();`
  - `do_getchar();`
- `do_kill()`: kills a process.
- `do_wait()`: waits on a process.

Spawn

- Kernel has a fixed array of PCBs.
- What info do you need to initialize a process?
  - PID
  - New stacks (user/stack)
  - Entry point (ramdisk_find)
  - total_ready_priority (lottery scheduling)

- Scheduler uses lottery scheduling: make sure you keep the sum of the priorities updated.

Message Boxes

- Bounded buffer:
  - Has fixed size;
  - FIFO;
  - Variable size message.
- Multiple producers:
  - Put data into the buffer.
- Multiple consumers:
  - Remove data from the buffer.
- Blocking operations:
  - Sender blocks if not enough space;
  - Receiver blocks if no message.
- Review Lecture 11 on Message Passing.
- Read MOS 2.3.7 and 2.3.8.

Mailbox – Implementation

- Buffer management:
  - Circular buffer: head and tail pointers.
- Bounded buffer problem:
  - Use locks and condition variables to solve this problem as shown in class;
  - Two condition variables: moreData and moreSpace (or any other names you prefer).

Keyboard – Overview

- How does the keyboard interact with the OS?
  - A hardware interrupt (IRQ1) is generated when a key is pressed or released;
  - Interrupt handler talks to the hardware and gets the scan code of the pressed/released key;
  - If it is SHIFT/CTRL/ALT/…, some internal states are changed;
  - Otherwise the handler converts the scan code into an ASCII character depending on the states of SHIFT/NUM LOCK/…

Keyboard – Overview

- How does the keyboard interact with the OS?
  - `init_idt()` in kernel.c sets handler to `irq1_entry` in entry.S;
  - `irq1_entry` calls `keyboard_interrupt` in keyboard.c;
  - `keyboard_interrupt` talks to the hardware and gets the scan code back (key = inb(0x60)) and calls the key specific handler;

Keyboard – Overview

- If key is SHIFT/CTRL/ALT/…, some internal states are changed.
- Otherwise `normal_handler` converts the scan code into an ASCII character.
- `normal_handler` calls `putchar()` to add character to the keyboard buffer.
- You need to implement `putchar()`.
- You also need to implement `do_getchar()`, which is called by the shell via syscall (get_char).

Keyboard – Implementation

- It is a bounded buffer problem:
  - Use mailbox.
- But, there are some variations:
  - Single producer (IRQ1 handler);
  - Multiple consumers (more than one process could use keyboard);
  - Producer cannot block – discard character if buffer is full.

Keyboard – Implementation

- Producer should not be blocked:
  - Solution: check and send message only if mailbox is not full, otherwise discard it.
  - Use the function do_mbox_full().
- Is that all?
  - What if a process being interrupted by IRQ1 is currently calling get_char()?
  - Address how to fix this issue in the design review.

Kill

- A process should be killed immediately.
  - Which queue it is in (ready, blocked, sleeping, etc.) doesn’t matter – kill it!
- Do not reclaim locks (this is extra credit).
- Reclaim memory:
  - PCB;
  - Stacks;
  - Look at robinhood test case to figure out what else needs to be reclaimed.
- Update total_ready_priority.

Wait

- Waits for a process to terminate:
  - Blocks until the process is killed or exits normally.
- What do you need to add to the PCB to implement this behavior?
- Return -1 on failure, 0 on success.

Hints/Tips

- List of functions to implement is straightforward. But, realizing the implementation is tricky!
- Look at util.h and check out any of the header files in the project folder for a helper function you might want.
- Use the settest script (two tests provided).
- You will need to change data structures and functions that are not annotated with TODO in the source code.