UiT The Arctic University of Norway

Project 4: Inter-Process Communication

INF-2201 Staff

UiT

Spring 2024

Project 4

- Project Overview
- Project Tasks
- Administrative Details

Project 4: Administrative Info (mostly like the others)

- Mandatory assignment
- Groups of two
- Design review
- Hand-in
  - Hand-in via Canvas
  - Code: whole repository (working tree + `.git/` dir)
  - Report: place in a `report/` directory in your repo
  - Zip up and upload to Canvas
  - **New!** Also upload the report PDF separately

Project 4: Inter-Process Communication (IPC)

- In Project 3:
  - We had separation between processes and kernel
  - But we didn't enforce it
- In Project 4:
  - We will enforce the separation
  - But we will build an Inter-Process Communication system
- You will:
  - Implement IPC message passing
  - Complete a keyboard driver
  - Reimplement sync abstractions to reduce interrupt disable time
  - Load processes dynamically

Project 4: Environment / Precode

Precode includes our solution to Project 3,
plus many additions to support Project 4.

Project 3	Project 4
All kernel priv (0)	Priv enforced: Kernel (0) vs Process (3)
One address space	Each process has own address space
Atomic via disabling interrupts	Atomic via C <atomic.h>
No interaction	Keyboard input + interactive shell
No disk I/O after boot	USB storage drivers
Flat disk image	Image has simple directory of processes

Be warned: Later projects have received less attention

- You will encounter more old, un-refactored code
- There will be bugs

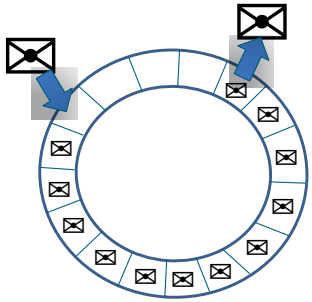
Project 4

Tasks Overview

You will:

- Implement IPC message passing
- Complete keyboard driver
- Reduce interrupt disable time (by reimplementing sync abstractions)
- Load processes dynamically

Task: Implement IPC Message Passing



A ring-buffer as a mailbox

Ring buffer

- Fixed-size: *bounded buffer*
- Variable-sized messages
- First In, First Out (FIFO)

Producer-consumer problem

- Multiple producers
- Multiple consumers
- Blocking operations

Implementation

- `mbox.[ch]`
- Struct already defined
- Locks + condition variables
- You will write the API functions

Mailbox API

- API functions

```
void mbox_init(void);
int mbox_open(int key);
int mbox_close(int q);
int mbox_stat(int q, int *count, int *space);
int mbox_recv(int q, msg_t *m);
int mbox_send(int q, msg_t *m);
```

- Mailboxes identified by integer id (like file descriptors)
- No other addressing
- Any process can put a message in a box
- Any process can remove a message from a box

Task: Complete Keyboard Driver

Chain

```
IRQ 1 -> keyboard_isr_entry -> keyboard_interrupt -> key handlers -> putchar
(hw)     (interrupt_asm.S)     (keyboard.c)          (keyboard.c)    (keyboard.c)
\_____________________________________________________________/    \_________/
                  Provided by precode                                 Your task
```

Hardware

- Hardware IRQ 1
- Read *scan code* from port 0x60

Partial driver

- `keyboard_interrupt`: look up scan code in table
- table has pointers to handler fns
- handler fns:
  - test for CTRL / SHIFT / ALT
  - map scan code to ASCII
  - call `putchar`

Keyboard Interface with OS: `putchar`

Precode

- Maps keyboard scan code to ASCII
- NB: US keyboard layout

`putchar`

- Put character into mailbox
- How exactly? Up to you

Classic producer/consumer problem

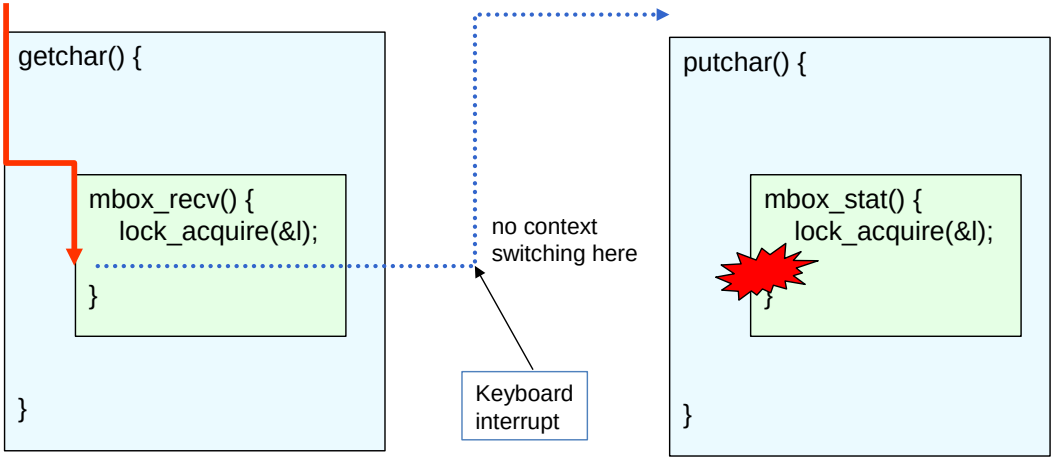
- Single producer. Or is it multiple?
- One source of input: keyboard
- But interrupt occurs in context of different threads/processes

- Multiple consumers. Or is it single?
- Any process can pull ASCII messages off of the mailbox
- But in practice it will usually just be the shell

Keyboard: Subtle Points

- Producer must not block
  - This is important for a keyboard handler
  - Why is this?
  - Solution: drop key if buffer is full
  - Check before send: `mbox_stat` before `mbox_send`
  - Need check + send to be atomic. Why?
- What if `getchar` is interrupted by a keyboard interrupt?
  - Why is this a problem?
  - Solution: disable interrupt. But where?
- ↑ Design review questions ↑

Keyboard: Deadlock Danger



`getchar` interrupted by keyboard interrupt

Task: Reduce Interrupt Disable Time

- Interrupts should be disabled as little as possible
  - Potential to lose hardware events
  - Interrupt controller will wait for CPU, but…
  - What if another key comes in before you handle the first?
  - What if another timer event fires before you handle the last?
- Where can we reduce?
  - Scheduler? Too difficult to rewrite this time
  - Sync abstractions? …

Rewriting Sync Abstractions

- In Project 3, we achieved atomicity by disabling interrupts
- Alternative: atomic test-and-set
  - C standard <atomic.h> provides `atomic_flag` type
  - `atomic_flag_test_and_set()`
  - `atomic_flag_clear()`
  - Compile atomic CPU instructions
- Use atomic-test-and-set to build a spinlock

```
spinlock_acquire(atomic_flag *flag) {
    while(atomic_test_and_set(flag)) yield();
}

spinlock_release(atomic_flag *flag) {
    atomic_flag_clear(flag);
}
```

- Use spinlocks to implement locks, condvars, etc.

Using Spinlocks: An Example

Via interrupt disable

```
static void lock_acquire(lock_t *l)
{
    nointerrupt_enter();
    while (l->locked) {
        block(&l->wait_queue);
    }
    l->locked = true;
    nointerrupt_leave();
}
```

Via spinlock

```
static void lock_acquire(lock_t *l)
{
    spinlock_acquire(&l->spinlock);
    while (l->locked) {
        block(&l->wait_queue, &l->spinlock);
    }
    l->locked = true;
    spinlock_release(&l->spinlock);
}
```

- `block` takes lock param
  - Releases before blocking
  - Re-aquires before returning
- This is no longer in the precode
- Should it be? Possible regression?

Task: Load Processes Dynamically

- In Project 3, all programs loaded at startup
- In Project 4, we will load them dynamically
- For dynamic loading, we need:
  1. Separate address space for each process
  2. Memory manager
  3. Disk format that lists processes
  - These are provided in precode

You will implement…

`readdir()`

1. Load process-directory sector
2. Return data to shell for parsing

`loadproc()`

1. Allocate physical memory
2. Read code/data from disk
3. Pass mem info to `create_process()`

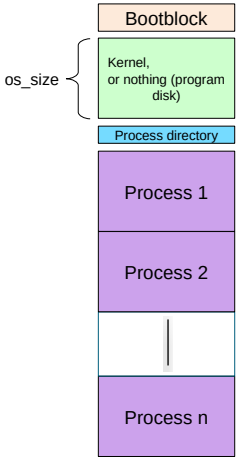
Loading Need 1: Separate Address Spaces

- Privilege enforcement via Global Descriptor Table
  - Descriptors for kernel-level code/data (ring 0)
  - Descriptors for user-level code/data (ring 3)
  - Precode `pcb` struct now has CS/DS fields
  - In-kernel threads: CS/DS point to kernel-level descriptors
  - User processes: CS/DS point to user-level descriptors
- Separate address spaces for user processes
  - Process loaded to virtual address 0x100 0000
  - GDT Task State Segment to help with task switching
  - Precode will handle the details
  - You just need to implement physical page allocation
  - In Project 5 you will get into the details

Loading Need 2: Memory Manager

- Pool of physical pages
  - 4 KiB per page
  - 1 MiB pool
- You implement: `alloc_memory()`
  - Get pages from the pool
- Extra challenge: implement `free_memory()`

Loading Need 3: Disk Format that Lists Processes



Updated disk format with process directory

- Add a process directory after the kernel
- Very simple process directory format:
  - List of (location, size) pairs, both `int`
  - No names. Processes identified by index
  - Process 0: shell
  - Process 1: process1 (plane)
  - …
- Kernel will load processes dynamically
  - Initiated by shell
  - `load 1`: load process1 (plane)
  - `load 2`: load process2 (math)
  - …
- In diagram: “Program disk”?
  - Old feature to hot-swap floppies
  - No longer working

Implementing Loading

USB storage support (precode)

- Precode provides a basic USB storage driver (`kernel/usb/**`)
- Key header for you: `scsi.h`
  - `scsi_read()`: reads one 512-byte sector
  - `scsi_write()`: writes one 512-byte sector
- For now, only reading

Loading a program: `loadproc()` (you)

- Read process directory: `readdir()` (you)
- Allocate pages for code + data: `alloc_memory()` (you)
- Also allocate pages for stack
- Read process sectors: `scsi_read()` (precode)
- Initialize a `pcb` struct: `create_process()` (precode)

Implementation Help: “Given” Files

- This project has many pieces that lean on each other
  - Hard to develop and test one without the others
- “Given” files: Compiled binaries of working implementations:
  - `mbox.given.o`: IPC mailboxes
  - `keyboard.given.o`: keyboard driver
  - `sync.given.o`: sync abstractions
  - `pcb.given.o`: dynamic loading
- Lines in `Makefile.common`: Uncomment to use given files

```
#KERNEL_OBJS := $(KERNEL_OBJS:kernel/mbox.o=kernel/mbox.given.o)
#KERNEL_OBJS := $(KERNEL_OBJS:kernel/keyboard.o=kernel/keyboard.given.o)
#KERNEL_OBJS := $(KERNEL_OBJS:kernel/sync.o=kernel/sync.given.o)
#KERNEL_OBJS := $(KERNEL_OBJS:kernel/pcb.o=kernel/pcb.given.o)
```

- Careful with headers: Some changes may break given files

Extra Challenges

- Free memory when a process exits
- Use a more interesting allocation algorithm
  - Make it more like a proper `malloc`
- Implement new commands in the shell:
  - `ps`: list running processes
  - `kill`: stop a running process
  - Suspend / resume a process
- To really go above and beyond, get creative:
  - You have input now. You could write a simple game
  - Try switching to VGA graphics mode
  - Write a better shell

Procedure is (Mostly) the Same as Other Projects

- Design reviews start on Monday (!!)
- Code
- Report
- Hand in via Canvas: zip up entire repository plus report
- **New!** Also upload the report PDF separately

Report

- Structure: scientific paper
- Length: around 4 pages
- Citations: required
  - You must cite sources you use
  - At the very least, you should have the textbook as a source
- File format: PDF
- AI tools: discouraged but not banned. Use must be declared
- **New!** See Howto doc and template in repository
  - `doc/how-to-write-a-report.md`: gives more guidelines and advice
  - `report/latex-src/template.tex`: LaTeX tutorial / template
  - If you use the template, be sure to remove all existing content

Hand In via Canvas

- Put your report in your repository, under a `report/` dir
  - Report shoud be a PDF format
  - If you write in Word or other WYSIWYG word processor, export to PDF
- Zip up your entire repository (code tree + report + `.git/` dir)
- Submit via Canvas
  - Upload zip
  - **New!** Also upload report PDF
  - Having both makes it easier for us to grade