# OSLab 4 Report

November 22, 2024

## 1 Overview

Lab 4 is about the implementation of preemptive multitasking among multiple simultaneouslt active user-mode enviroments. The lab is divided into three parts:

1. Part A: Multiprocessor Support and Round-Robin Scheduling
2. Part B: Copy-on-Write Fork
3. Part C: Inter-Process Communication

The most important thing is that JOS does not offer a strict test, so most of the permission check might be wrong. So most of the bugs are not permission check, but a wrong variable name or a wrong address. Moreover, We need to modify *syscall.c*! I have wasted a lot of time on this for many times.

## 2 Part A: Multiprocessor Support and Round-Robin Scheduling

### 2.1 Exercise 1

Just follow the instructions, align the size and call *boot_map_region* to map the memory.

### 2.2 Exercise 2

Just follow the instructions.

#### 2.2.1 Question 1

In *boot/boot.S*, we haven not begin the paging mode, so we can directly use the physical address. However, in *kern/mpentry.S*, we have already begin the paging mode, so we need to compute the virtual address. The macro $MPBOOTPHYS$ is used to convert the physical address to virtual address.

### 2.3 Exercise 3

Just follow the instructions.

### 2.4 Exercise 4

Just follow the instructions,we first get the running CPU id by *cpunum*() and then compute the corresponding address that stores the information of the current CPU. Then we cam correctly modify them.

### 2.5 Exercise 5& Challenge 1

Just follow the instructions.

#### 2.5.1 Question 2

The kernel will lock after the trap, so some registers has already been saved in the stack. So if two CPUs are trapped at the same time, the registers will be overwritten. The first Challenge is too difficult for me, so I just skip it.

### 2.6 Exercise 6&Challenge 2&Challenge 3

Just follow the instructions.

#### 2.6.1 Question 3

The address mapping in the kernel address space is the same for all env mappings.

#### 2.6.2 Question 4

Beacuse the process does not have it's own kernel stack, so we need to store the registers in a $PCB$ structure when the process is trapped. The code is in *trap.c*. I implemented a less trivial version of scheduling policy in Challenge 2. I put a priority variable for each *env*, and after each time slice, I will increase the priority of the current running *env* by 1. The timeslice is also determined by the priority. When a env voluntarily give up the CPU, it's priority will be decreased by 1.

### 2.7 Exercise 7

1. $sys\_exofork()$: Just follow the instructions.
2. $sys\_env\_set\_status()$: Just follow the instructions.
3. $sys\_page\_alloc()$: Just follow the instructions.It's strange to me that we must check the address is page-aligned.
4. $sys\_page\_map()$: Just follow the instructions.The important thing is to check the permission. In fact, the test is not strict enough, so I have to add all the permission check I can think of.
5. $sys\_page\_unmap()$: Just follow the instructions.

## 3 Part B: Copy-on-Write Fork

### 3.1 Exercise 8&9

Just follow the instructions.

### 3.2 Exercise 10

It's a quite difficult exercise, I spent 1 hour try to solve it but failed. Finally, I learned how the struct $UTrapframe$ is defined, and solved it.

### 3.3 Exercise 11

Just follow the instructions.

### 3.4 Exercise 12&Challenge 4

The *duppage* function must go through all pages below $UTOP$,not only below $USTACKTOP$.

As for the Challenge 4, I implemented $sfork()$. First,we need to add a new auxiliary function *sduppage* based on *duppage*. In *sduppage*, we simply copy all permissions, which is used to map the shared region. Then we implemented $sfork()$ based on $fork()$. We step from the stack end, when we go across the stack top, we will see a unprotected page, which can be used as a indicator of whether to use *duppage* or *sduppage*. By this way, we can pass the $froktree$ test. However, the *pingpongs* test will fail, since the shared region leads to a wrong *thisenv* reference. A better way is to use $sys_getenvid()$ to avoid the wrong reference. However, this needs to modify all the usage of *thisenv* in the kernel, which is too complex for me. The best solution needs a additional *Env* pointer array to store all the *env* pointers, and change *thisenv* to a macro definition. In this way, we can solve this problem easily.

## 4 Part C: Inter-Process Communication

### 4.1 Exercise 13&14

Just follow the instructions. Remember to uncomment the *sti* in *sched_halt*().

### 4.2 Exercise 15

We just follow the instructions. Remember to modify the *syscall.c*! I thought I have a wrong implementation at first, but finally I found that the bug lies in the *syscall.c*.