# Operating System Design & Implementation

# Lab3: Minimal Kernel

TA:陳勇旗

## Objective:

In this lab you can learn

- Understand x86 context-switch mechanism
- Understand how to create and add a new task to schedule routine.
- Implement simple system call services.
- Implement a task scheduler.

### 1. Before you doing this lab

Please download lab4 code from SVN, read lab4_reference.pdf and see what the difference between lab3 and lab4 kernel code.

### 2. Implement system call

System call is a software interrupt that use for provide OS system services. In this lab we define some system call for you, but you need to complete it.

| System call name | Service description | TODO |
|---|---|---|
| SYS_getc | Get character from keyboard. | Already |
| SYS_puts | Put string to video screen. | Already |
| SYS_fork | Create a task job. | You need implement |
| SYS_getpid | Get task id. | You need implement |
| SYS_sleep | Suspend a task by N ticks. | You need implement |
| SYS_kill | Recycle a task space. | You need implement |

Similar to lab3, you need setup a trap handler (in kernel/syscall.c) for system call service before enable the interrupt.

Trace the kernel/syscall.c, you need implement /* Lab4 TODO: xxx */ part.

### 2.1. Implement fork() system call

In our lab, fork system call is use for create a task job and inherit the memory stack status from parent. So trace kernel/task.c and complete the sys_fork() function.

### 2.2. Implement sleep() system call

When a task call this system call will let task into SLEEP state, please implement at syscall.c

### 2.3. Implement kill_self() system call

After a task finish you need recycle the task space, there you need implement a simple kill system service to recycle task space. (Trace task.c and implement sys_kill(int pid) function)

## 3. Implement task scheduler

At this part you need implement a simple task scheduler when timer interrupt occur. Trace kernel/timer.c and implement sched_yield() function in sched.c.

Please see the task state diagram in lab4_reference.pdf p.18, your scheduler need follow that state transient flow.

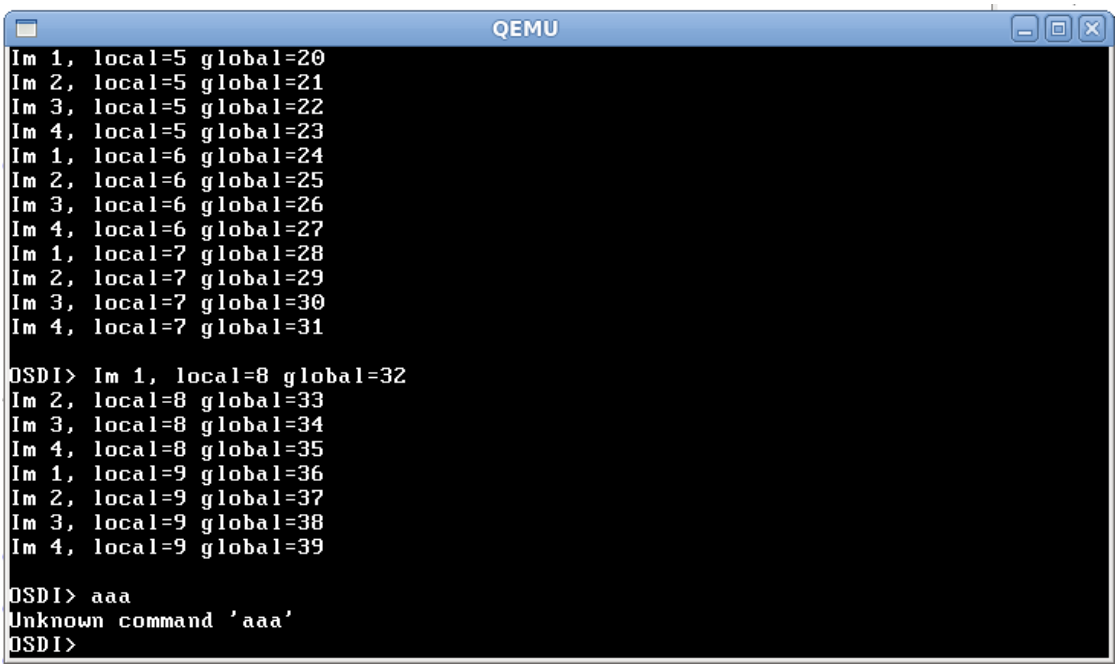
## 4. Demo

In kernel/main.c we fork 4 tasks (user.c) and a shell command program after switch to user mode, each user task will print some message then sleep 100 ticks. And your shell should be first task and always run in background.

```
/* Below code is running on user mode */
if (fork())
{

        /* Child */
        if (fork()) task_job();
        else{
        if (fork()) task_job(); else
        if (fork()) task_job(); else task_job();}

}
else
{
        shell();
}
/* task recycle */
kill self();
```


QEMU
Im 1, local=5 global=20
Im 2, local=5 global=21
Im 3, local=5 global=22
Im 4, local=5 global=23
Im 1, local=6 global=24
Im 2, local=6 global=25
Im 3, local=6 global=26
Im 4, local=6 global=27
Im 1, local=7 global=28
Im 2, local=7 global=29
Im 3, local=7 global=30
Im 4, local=7 global=31
OSDI> Im 1, local=8 global=32
Im 2, local=8 global=33
Im 3, local=8 global=34
Im 4, local=8 global=35
Im 1, local=9 global=36
Im 2, local=9 global=37
Im 3, local=9 global=38
Im 4, local=9 global=39
OSDI> aaa
Unknown command 'aaa'
OSDI>