

41347903S 鄭詠澤

1 Further study on process switching

With direct execution is achieving a switch between processes, if a process is running on the CPU, this by definition means the OS is not running. Thus, OS need a method to regain the control of the CPU.

There are 2 approaches, Cooperative and Non-Cooperative Approach. Timer interrupt was introduced from the second approach.

A timer device can be programmed to raise an interrupt every so many milliseconds; when the interrupt is raised, the currently running process is halted, and a pre-configured interrupt handler in the OS runs.

At this point, the OS has regained control of the CPU, and thus can decide to stop the current process or start a different one.

2 Basic concepts of terminal and shell

2.1 Screenshot

```
mes@MesDesktop:~/NTNU-OS/ostep-code/intro$ setarch $(uname --machine) --addr-no-randomize /bin/bash
mes@MesDesktop:~/NTNU-OS/ostep-code/intro$ ./mem 10 & ./mem 555
[1] 3305
(3306) addr pointed to by p: 0x4052a0
(3305) addr pointed to by p: 0x4052a0
(3306) value of p: 556
(3305) value of p: 11
(3306) value of p: 557
(3305) value of p: 12
(3306) value of p: 558
(3305) value of p: 13
(3306) value of p: 559
(3305) value of p: 14
(3306) value of p: 560
(3305) value of p: 15
(3306) value of p: 561
(3305) value of p: 16
(3306) value of p: 562
(3305) value of p: 17
(3306) value of p: 563
(3305) value of p: 18
[1]+  Terminated                  ./mem 10
Terminated
mes@MesDesktop:~/NTNU-OS/ostep-code/intro$

mes@MesDesktop:~/NTNU-OS/ostep-code/intro$ setarch $(uname --machine) --addr-no-randomize /bin/bash
mes@MesDesktop:~/NTNU-OS/ostep-code/intro$ killall mem
mes@MesDesktop:~/NTNU-OS/ostep-code/intro$
```

2.2 Explanation

In a Unix-like system, each process is identified by a unique process ID (PID) and managed by the OS itself, rather than being tied to a particular terminal.

When we issue a kill command from any terminal, it sends a signal to the target process's PID. As long as the user running the kill command has sufficient privileges (for example, the same user that started the process or the superuser), the OS will deliver the signal to the process, causing it to terminate.

Because the OS abstracts away the terminal association of processes—meaning, it only cares about the PID and not which terminal launched it—processes can be killed from any terminal session with the appropriate permissions.

3 Learning from the Linux man pages

The `us` (user), `sy` (system), and `id` (idle) fields refer to how the CPU is spending its time:

- **us** : time running un-niced user processes
- **sy** : time running kernel processes
- **ni** : time running niced user processes
- **id** : time spent in the kernel idle handler
- **wa** : time waiting for I/O completion
- **hi** : time spent servicing hardware interrupts
- **si** : time spent servicing software interrupts
- **st** : time stolen from this vm by the hypervisor

The **us** value shows the percentage of CPU time spent running processes in user space (i.e., non-kernel applications), the **sy** value shows the percentage of CPU time spent running kernel processes or system calls, and the **id** value represents the percentage of CPU time that is idle (i.e., not doing any work).

4 I/O and system calls

The man page said

EEXIST pathname already exists and `O_CREAT` and `O_EXCL` were used.

and for `O_CREAT` and `O_EXCL`:

O_CREAT: If pathname does not exist, create it as a regular file.

O_EXCL: Ensure that this call creates the file: if this flag is specified in conjunction with `O_CREAT`, and pathname already exists, then `open()` fails with the error `EEXIST`.

Thus, to trigger an `EEXIST` error, we need to open an already-existing file with both `O_CREAT` and `O_EXCL`:

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
```

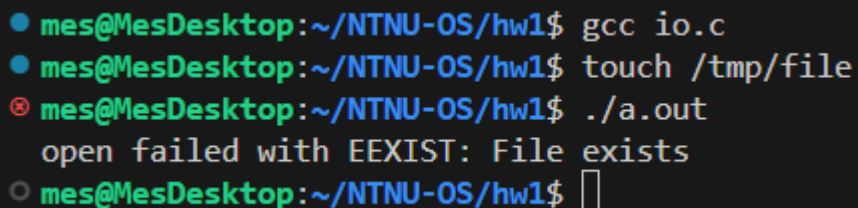
```
#include <errno.h>
#include <string.h>

int main(int argc, char *argv[]) {
    int fd = open("/tmp/file", O_WRONLY | O_CREAT | O_EXCL, S_IRUSR |
S_IWUSR);
    if (fd < 0) {
        if (errno == EEXIST) {
            perror("open failed with EEXIST");
        } else {
            perror("open failed");
        }
        return 1;
    }
    close(fd);
    return 0;
}
```

To test it, we have to ensure that `/tmp/file` already exists, we can do it by running `touch /tmp/file` command.

Then because the file is already there, `open()` will fail with the error `EEXIST`. The reason it happens is that `O_EXCL` explicitly tells `open()` to fail if the file is already present, thus preventing accidental overwrites.

Test output:



```
mes@MesDesktop:~/NTNU-OS/hw1$ gcc io.c
mes@MesDesktop:~/NTNU-OS/hw1$ touch /tmp/file
mes@MesDesktop:~/NTNU-OS/hw1$ ./a.out
open failed with EEXIST: File exists
mes@MesDesktop:~/NTNU-OS/hw1$
```

5 Setting up xv6

```
mes@MesDesktop: ~/NTNU-C × + v
mes@MesDesktop:~/NTNU-OS/hw1/xv6-riscv$ make qemu
qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 3 -nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ ls
.          1 1 1024
..         1 1 1024
README    2 2 2292
cat       2 3 34264
echo      2 4 33192
forktest  2 5 16184
grep      2 6 37528
init      2 7 33656
kill      2 8 33104
ln        2 9 32928
ls        2 10 36296
mkdir     2 11 33168
rm        2 12 33152
sh        2 13 54728
stressfs  2 14 34048
usertests 2 15 179352
grind     2 16 49400
wc        2 17 35224
zombie    2 18 32528
console   3 19 0
```