# OS Report 2020

劉恩婷 b05902090 資工四

## 設計

The main process will controll all the process scheduling. Since the machine has multiple cores, the main process is set to run on core 0 while all child process run on core 1. This is done by the code below:

```
cpu_set_t set;
CPU_ZERO(&set);
CPU_SET(core, &set);
if (sched_setaffinity(0, sizeof(set), &set) == -1){
        fprintf(stderr, "set core error");
        exit(1);
}
```

Scheduler will continue to run until all the process finish execution. In the main process, each unit of time is defined as execution time of empty loop of one million iterations. For each round, if a process's arrival time equals to main process time, main process will fork a process but set its priority to lowest until it is schedule to run.

```
struct process{
        char Ni[32];
        int Ri;
        int Ti;
        int pid;
        int ran;
        unsigned long start_s, start_ns, end_s, end_ns;
};
```

structure of a process

```
sched_setscheduler(pid, SCHED_IDLE, &param)
//set process to run at very low priority background jobs

sched_setscheduler(pid, SCHED_OTHER, &param)
//the standard round-robin time-sharing policy, since no other process, it will
continue to run until time quantum expires
```

Main process record the start of child process when the process first set to run by the scheduler. The process end time will be recorded after waitpid().

## FIFO

The process are sorted according to their arrival time. Then, $P_{i+1}$ will only run until $P_i$ has finish executed.

## RR

Processes will be fork and store in queue upon arrival. If CPU is idle, the first process in queue will be selected to run. If the process $T_i$ is more than time quantum, it will be terminated in this quantum, else the next round will be time + $T_i$.

If time quantum expires and process is still running, it will be set to idle, another process from the queue will later be selected to run.

## SJF

If the cpu is idle and there are processes ready to run, it will select the process with the shortest $T_i$(CPU burst). $P_i$ will continue to run until finish execution.

The main process will then select the shortest job from the rest of the processes.

## PSJF

If the cpu is idle and there are processes ready to run, it will select the process with the shortest $T_i$(CPU burst). $P_i$ will continue to run until finish execution. Then, the main process will select the shortest job from the rest of the processes.

However, while running $P_i$, if there exists another job $P_j$ for which $T_j < T_i$, then main process will set $P_i$ to idle state and schedule $P_j$ to run.

# 核心版本

Ubuntu Linux 4.14.25

## get time function

```
asmlinkage int sys_my_time(unsigned long *time_s, unsigned long *time_ns){
        struct timespec ts;
        getnstimeofday(&ts);
        *time_s = ts.tv_sec;
        *time_ns = ts.tv_nsec;
        return 0;
}
```

## print relevant information in dmesg

```
asmlinkage int sys_my_print(int pid, unsigned long start_s, unsigned long start_ns,
unsigned long end_s, unsigned long end_ns){
        printk("[Project1] %d %ld.%ld %ld.%ld\n", pid, start_s, start_ns, end_s,
end_ns);
        return 0;
}
```

# 比較實際結果與理論結果，並解釋造成差異的原因

Generally, the sequence of the result is as expected. In FIFO, we expect that the start time of $P_{i+1}$ is more than $P_i$, however, it didn't start immediately after $P_i$ ended. This might be due to CPU handling other tasks during context switching.

For SJF, we expect that it behaves like FIFO with selecting the job that has shortest time. However, process that has < 50 execution time ($R_i$ > 100 ) finish before process ($R_i$ == 0). This situation is probably due to the child process finish executed the unit time function before parent process set its priority to run in background.

Besides, preemptive process tend to differ more than non-preemptive process in terms of running time.