2020 OS Project 1 – Report

B06202018 數學三 張芷榕

1. 設計

- **kernel files**

```
asmlinkage int sys_get_time(unsigned long *sec, unsigned long *nsec);
```

用 `getnstimeofday` 來獲得時間。

```
asmlinkage int sys_show_info(char *output);
```

用 `printk` 來 output information。

- **process.c**

自定義結構型態 process 來存取 process 的 information：

```c
typedef struct {
    pid_t pid; // pid of the process
    char name[32]; // name of the process
    int exec_t; // the amount of time the process still need to be executed
    int ready_t; // the time the process is ready
} process;
```

同時使用結構型態 process_node, process_list 來進行 process queue 的實作：

```c
typedef struct node {
    process *p;
    struct node *next;
} process_node;

typedef struct{
    process_node *head;
    process_node *tail;
} process_list;
```

```
void proc_assign_cpu(pid_t pid, int cpu);
```

利用 `sched_setaffinity` 將所有 process assign 到同一 CPU 來模擬排

程。

```
pid_t proc_exec(process *p);
```

根據 process 的 information 來執行此 process。

在 fork 出的 child process 用 system call 獲得開始執行的時間，然後依照要求 run 數個 unit time，結束後同樣地用 system call 獲得結束時間，並 output 到 dmesg。

parent process 則會回傳 child process 的 pid。

```
void proc_blck(pid_t pid);
```

利用 `sched_setscheduler(pid, SCHED_IDLE, &param)`的方式來 block 指定的 process。

```
void proc_wake(pid_t pid);
```

利用 `sched_setscheduler(pid, SCHED_OTHER, &param)`來提高指定 process 的 priority，使得此 process 能被執行。

- **schedule.c**

用來記錄排程 information 的一些變數：

```
1. // initialization
2. int cur_t = 0; // current time
3. int num_fin = 0; // number of finish process
4. int ready_idx = 0; // index of process list to check if the pro-
   cess is ready
5. process *last_p = NULL; // last executed process
6. int last_t; // the last time at which context switch hap-
   pens, used for RR policy
```

```
void insert_to_list(process_list *queue, process_node *new, int
policy);
```

根據 scheduling policy 將 process insert 到 queue 之中。

1. FIFO, RR
   將 process 插入至 queue 的尾端。
2. SJF, PSJF
   根據 process 的 `exec_t` 長短，依照順序插入 queue。但要注意的是，SJF 是 non-preemptive，所以儘管要插入的 process，`exec_t` 比現在 queue 的 head process 還要短，也只能插在 head 的後面。

```
void remove_from_list(process_list *queue);
```

將 process queue 的第一個 node 移除。

```
process *sched_next(process_list *queue, int policy);
```

根據 scheduling policy 找出下一秒要執行的 process，return 此 process 的 pointer，若是當前沒有 process 樣執行，則 return NULL。

基本上就是 return process queue 的 head node，只有 RR 要注意，若是距離上次 context switch 已經過了 500 個 unit time，要 return 的是 queue 中的下一個 process。

```
void scheduler(process *P, int num_p, int policy);
```

將 process list 中的 process 依照 policy 模擬排程。

首先用 `proc_assign_cpu(getpid(), SCHED_CPU)`及 `proc_wake(getpid())`，使得 scheduler 能獨自使用一顆 CPU，不與排程的 process 混在一起執行。

將 process list 依照每個 process 的 ready time 進行 sorting，以便後續的操作。

每一秒檢查是否有 process 在這一秒 ready（`P[ready_idx].ready_t == cur_t`），若是 ready 了就建立一個 process node（ `process_node *new`）依照 policy 加入 queue 之中（`insert_to_list(ready_queue, new, policy)`），並呼叫 `proc_exec(&P[ready_idx])`來執行並取得此 process 的 pid，隨後使用 `proc_blck(P[ready_idx].pid)` 先將 process block 起來，等待排程來決定執行的時間。

再來呼叫 `sched_next(ready_queue, policy)`來決定要執行的 process，並用 `proc_blck(last_p->pid)`和 `proc_wake(cur_p->pid)`進行 context switch。要注意的是，若是 policy 為 RR 且發生 context switch 的話（`last_p != cur_p`），要將目前 queue 的 head node 移到 queue 的尾端

執行一個 unit time 後，將目前執行的 process 的待執行時間（`cur_p->exec_t`）減一，若時間剩下為零表示 process 執行結束，將此 process 從 queue 中移除（`remove_from_list(ready_queue)`）。依照要求將需要的 information output 到 stdout，並將記錄完成 process 數的 `num_fin` 加一。

等到所有 process 結束執行（`num_fin == num_p`），便結束排程。

- **main.c**

根據 stdin 的 input 創造一個 process list P，再呼叫 `scheduler(P, num_p, policy)`進行排程。

2. 核心版本

4.14.25

3. 理論與實際結果的比較

比較實際結果與理論結果，process 完成的順序是正確的。因為將 scheduler 與 process 分開在不同 CPU 執行，所以能有較為準確的結束時間計算，但每個 process 的結束時間仍然多少會和理論有所出入，這可能是因為在執行 context switch 時的 overhead 所導致。