OS Project 1

Design

Processes communicate with each other by shared memory. The contents of shared memory are the total running time, the pid of next running process, the pid of running process, the time of its remaining exection time, and the value that represents whether the process has finished.

- $FIFO$:
    - First, sort the process by its ready time. After sorting, we can use sched_setscheduler() to set the correct priority to the process by the order at the right time. Below is the compare function I used.

    ```
    int cmp(const void* a, const void *b){
      return ((struct process *)a)->s_time > ((struct process *)b)->s_time;
    }
    ```

- RR :
    - First, sort the process by its ready time like I mentioned above. After that, we can schedule the processes by its ready time. When the running process has run 500 unit time or finished the task. It should lower the priority of itself and increase the priority of scheduler. When scheduler has the control of CPU, it can select the next running process, and increase the priority of target process.
- SJF :
    - First, sort the process by its execution time and ready time respectively. After sorting, we can use the same algorithm of $FIFO$ to help scheduling those process. Below is the compare function I used in SJF.

    ```
    int cmp_for_SJF(const void *a, const void *b){
        struct process* _a = (struct process*)a;
        struct process* _b = (struct process*)b;
        if(_a->s_time == _b->s_time){
            return _a->e_time > _b->e_time;
        }
        return _a->s_time > _b->s_time;
    }
    ```

- $PSJF$:
    - The difference betwwen $PSJF$ and SJF is the preemption. Hence, we need to sort the scheduling order each time when a new process is ready. After sorting, it's the same as $FIFO$. We can use the same algorithm of $FIFO$ to schedule it. Below is the compare function I used in SJF.

```c
int cmp_for_PSJF(const void *a, const void *b){
    struct process* _a = (struct process*) a;
    struct process* _b = (struct process*) b;
    if(_a->e_time == 0)
        return _a->e_time < _b->e_time;
    if(_b->e_time == 0)
        return _b->e_time > _a->e_time;
    return _a->e_time > _b->e_time;
}
```

The kernel version: Linux ubuntu 4.14.25

Comaprison

- Because I use only one CPU to simulate the scheduler and processes. Besides, my computer will also context switch, which means sometimes the virtual machine can't get the resource of CPU. Hence, the main difference between the reality and the theory is the time of **context switch**. Because **context switch** has more overhead, which means it will idle CPU some time.