Design

Initial

- Use 2 CPU, one for the scheduler, one for fork() process.
- In main.c determine the schedule policy and execute correspond scheduler.
- Initial function
    - SET_CPU() : to set CPU for process
    - Unit_time() : run defined unit time

Quick Sort

- Use to sort process based on their ready time
- Use q sort in C library
- Compare function
    - cmp() : compare the ready time

Queue

- Use in RR policy
- Implement by linked list
- Defined structure to save the info
    - front : pointer to the first node in queue
    - rear : pointer to the last node in queue
    - count : number of element in queue
    - index : the index of the process
    - next : pointer to next node
- Queue function
    - init() : initial a queue
    - enqueue() : add element to queue

Process

- Define a structure to save process info
    - name : the process name
    - ready : the ready time for the process
    - execute : the execution time need for the process
    - pid : the process id of the process
- Process function
    - Proc_Exec() : to fork() the process

- Proc_Wake() : to set high priority to process, the process is now running
- Proc_Block() : to set low priority to process, the process is now prevent from running
- Proc_next_FIFO() : determine next process to run for FIFO policy
- Proc_next_RR() : determine next process to run for RR policy
- Proc_next_SJF() : determine next process to run for SJF policy
- Proc_next_PSJF() : determine next process to run for PSJF policy

Scheduling

- **FIFO**
  - The processes are sorted by the ready time.
  - Execute the processes based on their ready time one by one.
  - Every process will run till complete before next process run.
  - Subtract the execute of the running process by 1 for each unit time.
  - After the execute of the process equal 0, stop by wait() and free the CPU for other processes.
- **RR**
  - The processes are sorted by the ready time.
  - A queue is use to save the queue of the processes to run.
  - When reach the ready time of the process, enqueue the process index to the queue.
  - Each process will run for a unit time, if after a unit time but the process didn't complete, reallocate the node of current process as the rear of the queue.
  - If the queue not empty, we get the index of next process from the front node of the queue.
  - When the process is complete, remove its node from the queue and subtract the count of queue by 1.
  - Subtract the execute of the running process by 1 for each unit time.
  - After the execute of the process equal 0, stop by wait() and free the CPU for other processes.
  -
- **SJF**
  - The processes are sorted by the ready time.
  - Each time we choose the process with the smallest execute to run.
  - If the process pid equal -1 or execute equal 0, this process is either not reach its ready time or complete. Compare the execute for the index of process with smallest execute.

    - Every process will run till complete before next process run.
    - Subtract the execute of the running process by 1 for each unit time.
    - After the execute of the process equal 0, stop by wait() and free the CPU for other processes.
- **PSJF**
    - The processes are sorted by the ready time.
    - Each time we choose the process with the smallest execute to run.
    - If the process pid equal -1 or execute equal 0, this process is either not reach its ready time or complete. Compare the execute for the index of process with smallest execute.
    - If exist process with smaller execute compare to running process, we change to the one with smaller execute.
    - Subtract the execute of the running process by 1 for each unit time.
    - After the execute of the process equal 0, stop by wait() and free the CPU for other processes.

Kernel and Linux version

• Ubuntu	: ubuntu-18.04.4-desktop-amd64
• Linux	: linux 4.14.25

Comparison

- The actual result for the execution period for the process under 4 given schedule policy is actually not similar to the theoretical result.
- Since for the local system, there are background processes that still running when we run the program. This might cause our program experience context switch while the timer for our program didn't stop when our program stopped.
- Also for each execution of our program, the scheduler program will not completely synchronize with the timer in out system thus there will be some disparity.
- There are many function that we call for the scheduling of the process, the calling of those process also take time and this cause the different between the actual and theoretical result.