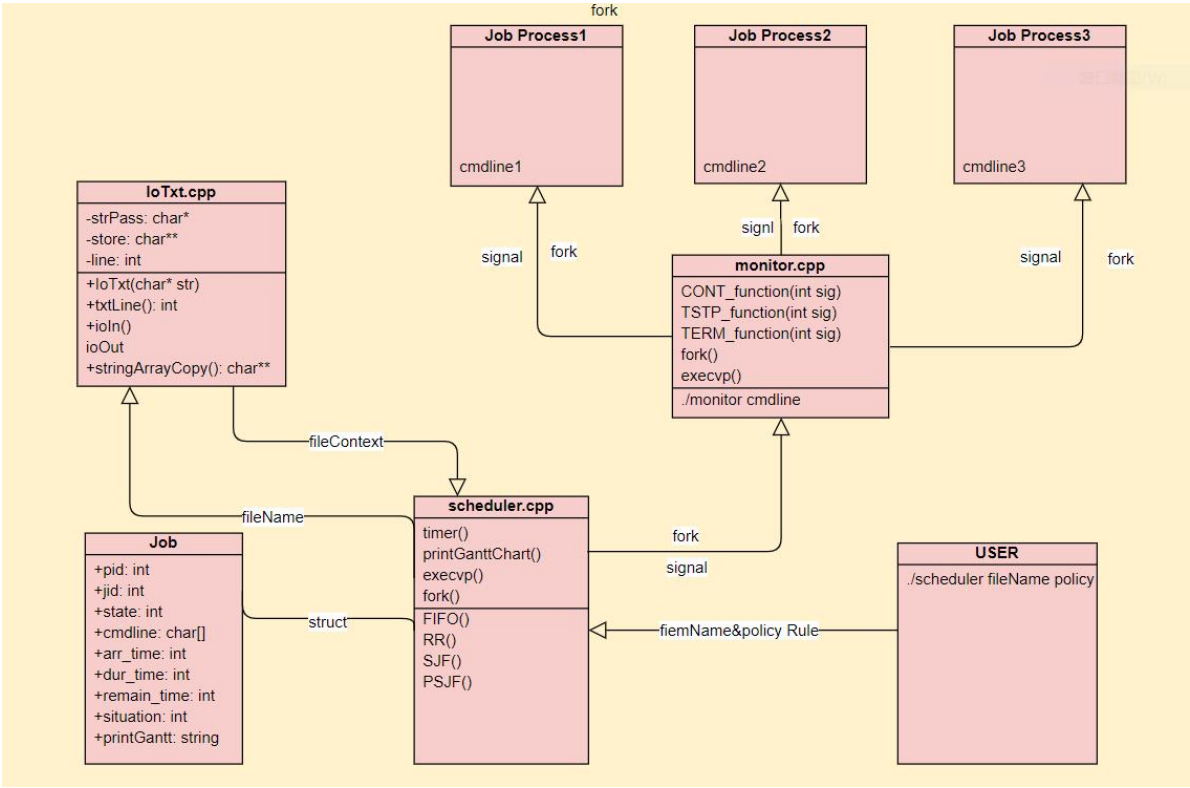
Report of Project --- Process Scheduler

> Group Member:
>
> > Li Yichu 18098533-I011-0103
> >
> > Wang Hongbo 18098532-I011-0018
>
> Subject: Projects on Operating Systems
>
> Teacher: Henry Dai

> Date: 2021/05/09

System Architecture



Descriptions

Monitor

monitor.cpp

1. Name: monitor
2. Features: through this file, we can complete the monitor program, including invoking the correct job process, achieving signal relaying mechanism and correct measurement of the execution time of the job process.
3. Functions:
    - CONT_function(int sig): signal handler to SIGCONT signal, use to control the child process.

      Returned data type: void
    - TSTP_function(int sig): signal handler to SIGTSTP signal, use to control the child process.

      Returned data type: void
    - TERM_function(int sig): signal handler to SIGTERM signal, use to control the child process.

      Returned data type: void
    - main(int argc, char* argv[]): complete the monitor program. It forks a now process to execute the job, changes the signal handlers, and records the execution time of the job process.

Scheduler

iotxt.h and iotxt.cpp

1. Name: iotxt
2. Features: through these file, We can complete the reading of the file under the relative path.
3. Data structure:
    1. class IoTxt
        1. Functions:
            - IoTxt(char* str): construct function, be used as a get method to get the file path entered by user. Provide an interface to input the absolute path of txt to the interface in the program.

              Returned data type: None
            - txtLine(): get the total number of lines in a txt file.

              Returned data type: int
            - ioOut(): obtain the input file and save the file in an array to facilitate future processing and use.

              Returned data type: void
            - stringArrayCopy(): read from ioOut() and store the file information in the char** array.

              Returned data type: char**

scheduler.cpp

1. Name: scheduler
2. Features: through this file, we can complete the FIFO, SJF, RR and preemptive SJF scheduler.
3. Data structure:
    1. Class Job
        1. Variables:
            - pid: record the sequential number of this job in the execution file.
            - jid: job process number created by monitor.
            - cmdline[MAX_CMD]: where MAX_CMD is the maximum number of bytes of Command in the file, cmdline[] is the command that the job needs to run in the execution file.
            - arr_time: the arrival time of the job in the execution file.
            - dur_time: in the execution file.
            - remain_time: the remaining running time of the job.
            - situation: the status of the job (there are four states, namely unreached state, reached state, and executed state, execution completed status).
            - printGantt: record job processing information.
4. Functions:
    1. strtok_counter_function(char cmdline[]): According to cmdline[], calculate how many parts it is divided into by spaces, and then return the length of the command string.

       Returned data type: int
    2. strtok_function(char cmdline[], int tokenArrayLength, char* tokenArray[]): save all the command that scheduler needs to execute to our tokenArray.

Returned data type: void

3. setJobProcess(Job process[], int line, char** store): store the arrival time, execution time, cmdline and other information in the file in the corresponding structure Job array.

   Returned data type: void

4. beReadyProcess(Job process[], int line): by calling the fork() function and execvp() function, the corresponding process is generated.

   Returned data type: void

5. printGanttChart(Job process[], int line): Print the Gantt chart of the corresponding job according to the system scheduling record.

   Returned data type: void

6. timer(): Make the parent process sleep for one second by calling the function sleep(1), while make the child process execute for a complete second.

   Returned data type: int

7. record(Job process[], int line): record the status of the current scheduler every second.

   Returned data type: void

8. FIFO(Job process[], int line): complete the function of the First-In-First-Out scheduler.

   Returned data type: void

9. SJF(Job process[], int line): complete the function of the Shortest-Job-First scheduler.

   Returned data type: void

10. RR(Job process[], int line): complete the function of the Round-Robin scheduler.

    Returned data type: void

11. PSJF(Job process[], int line): complete the function of the preemptive Shortest-Job-First scheduler.

    Returned data type: void

12. policyRule(char* policy, Job process[], int line): According to the scheduling method input by the user, run the corresponding scheduling function.

    Returned data type: void

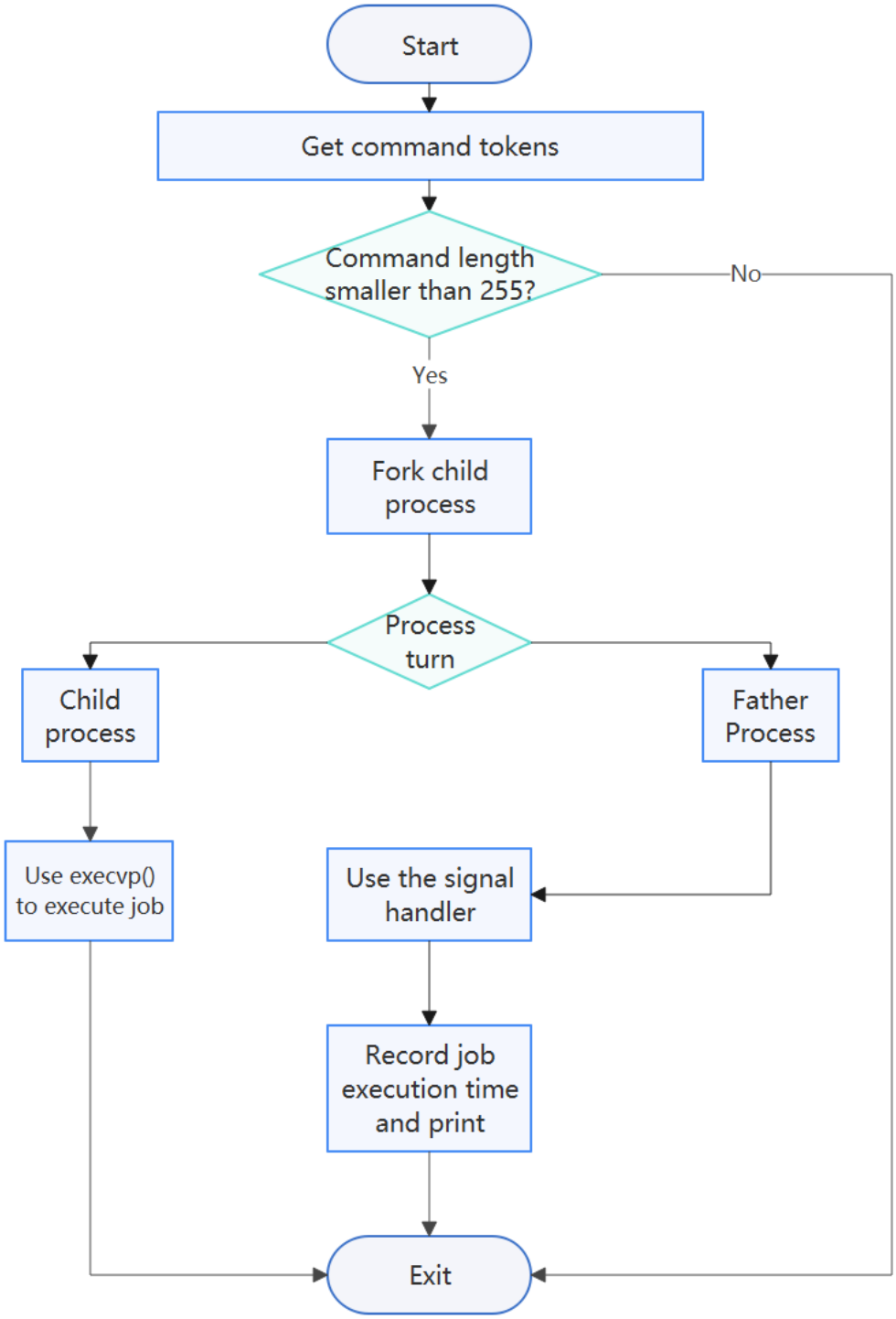
13. main(int argc, char* argv[]): Complete the FIFO, SJF, RR and preemptive SJF scheduler. The main processes are:

    1. Read data of the selected file, record some information about jobs.
    2. Generate processes that will be executed.
    3. Use a type of schedule method that is selected by user to execute the jobs.
    4. Print the Gantt chart of the jobs.

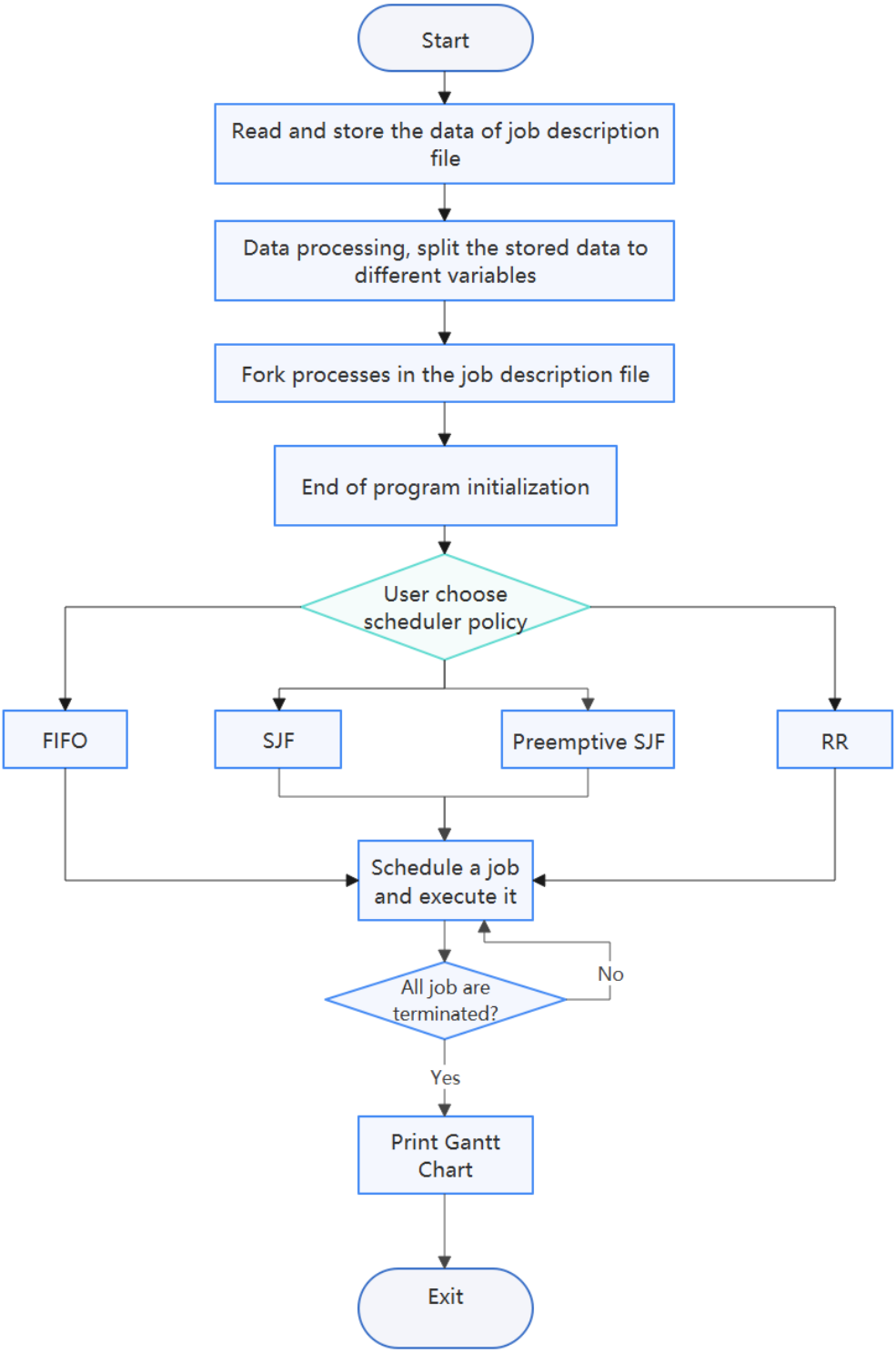
    Returned data type: int

Flow Chart

Monitor


Start
Get command tokens
Command length smaller than 255?
No
Yes
Fork child process
Process turn
Child process
Father Process
Use execvp() to execute job
Use the signal handler
Record job execution time and print
Exit

Scheduler



Division of Labor

Monitor

Wang Hongbo: designed the architecture of the monitor and completed the signal handler part.

Li Yichu: completed the execution function of job process and accurately count the execution time.

Scheduler

Wang Hongbo: designed the architecture of scheduler, I/O function, participated in the implementation process of scheduler policy.

Li Yichu: completed the scheduler policy of FIFO, RR, SJF and preemptive SJF.

Others

Wang Hongbo: wrote some descriptions of this report, draw the system architecture diagram of this report.

Li Yichu: wrote the rest part of this report, wrote a makefile and a user manual(README.md).

Thoughts about This Project

During the completion of this project, we understand better about UNIX system working mechanism and system call. We also reviewed some of the techniques we learned in class, such as how to create child processes and successfully control them by using signal. Also, we have also encountered some problems, which may not be explained in class or explained in detail. In this case, we learned to look Linux help manual or use google for a answer. We realized that knowledge is not only limited to what the teacher said in class and we improved our problem-solving ability during this process. In conclusion, we can say that completing this project helped us a lot.