운영체제 REPORT

**-Project 1-**

**<Scheduling >**

**모바일 시스템 공학과**

**32151181 김진화**

**32155161 Tan Chaiwee**

**CONTENTS**

**1. 주제**

**2. 알고리즘**

**3. 소스코드 및 부연설명**

**4. 결과**

1. **주제**
2. **Scheduling**

Project 1의 주제는 Scheduling을 구현하는 것이다. Scheduling은 CPU를 사용하려고 하는 프로세스들 사이의 우선 순위를 관리하는 일을 말하며, 이는 다중 프로그래밍을 가능하게 하는 운영 체제의 동작 기법이다. 스케줄링 알고리즘은 여러가지가 있으며 스케줄링 적용 시점에 따라 비선점형(Non-preemptive Scheduling) 과 선점형(Preemptive Scheduling) 의 2가지로 구분할 수 있다. 비선점형 스케줄링은 어떤 프로세스가 CPU를 할당 받으면 그 프로세스가 종료되거나 입출력 요구가 발생하여 자발적으로 중지될 때까지 계속 실행되도록 보장한다. (ex. FIFO, SJF) 반면, 선점형 스케줄링은 어떤 프로세스가 CPU를 할당 받아 실행 중에 있어도 다른 프로세스가 실행 중인 프로세스를 중지하고 CPU를 강제로 점유할 수 있는 것이다. (ex. Round Robin)

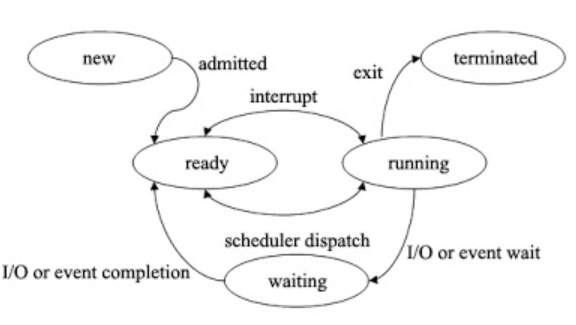


Figure 1. Process state Diagram

Process life cycle

1.     Creation -> Ready – Started by a system call, a process control block is created and it initializes the process.

2.     Ready -> Running – When a processor is free to execute the process and the operating system has scheduled one of the ready state process to run next

3.     Running -> Ready – When there is an external interrupt from a timer, its time quantum has finished.

4.     Running -> Wait – When a process makes an operating system request(syscall), a running process goes into wait state to wait for the interrupt(eg. IO burst) to complete execution.

5.     Wait -> Running – After the interrupt(eg. IO burst) has completed its execution, the waiting process can now be pass to ready state and ready to be processed again.

6. Running -> Termination – After a process complete its execution, it is terminated.

1. **FIFO(First Input First Output)란?**

First-in first-out scheduling algorithm is one simplest scheduling method among all the other scheduling algorithm. It arranges processes in a queue according to the arrival of a process. The first process will be executed first and the next process will start its execution once the first process is done.

• Waiting time:

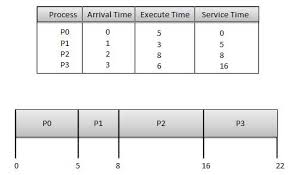
First process will be executed first and though have no waiting time. The next process waiting time will be the execution time of the first process and the next process have the sum of 1st process and 2nd process waiting time and so on.

• Scheduling overhead:

Since scheduling only happen when one process ends, the scheduling overhead is low.

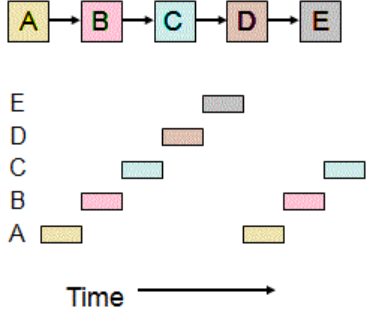
• Effectiveness:

This algorithm leads no starvation because eventually every process gets to be executed, however the throughput is low because the CPU might get hold for a process with long execution time, while some other short processes have to wait for very long time.

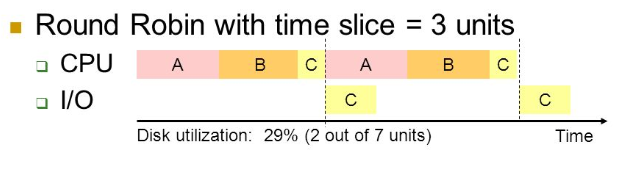


1. **Round Robin이란?**

Round Robin Scheduling은 시분할 시스템을 위해 설계된 선점형 스케줄링의 하나로서, 프로세스들 사이에 우선순위를 두지 않고, 순서대로 시간단위(Time Quantum)로 CPU를 할당하는 방식의 CPU 스케줄링 알고리즘이다. 시간 단위동안 수행한 프로세스는 준비 큐의 끝으로 밀려나게 된다. 문맥 전환의 오버헤드가 큰 반면, 응답시간이 짧아지는 장점이 있어 실시간 시스템에 유리하다.



1. **Round Robin with I/O**



1. **알고리즘**
2. **FIFO**
3. By creating a queue, with front and rear pointer pointing at the same index at first, indicating empty queue, arrange processes into the queue upon their arrivals.
4. Adding process with rear pointer increasing and deleting process upon completion with front pointer increasing.
5. A process is scheduled to start when the process before it ends.
6. **Round Robin**
7. Timer를 설정하여 매 초마다 parent(kernel)가 신호를 받게 된다.

🡪setitimer 로 타이머 설정, sigaction 설정으로 signal 처리

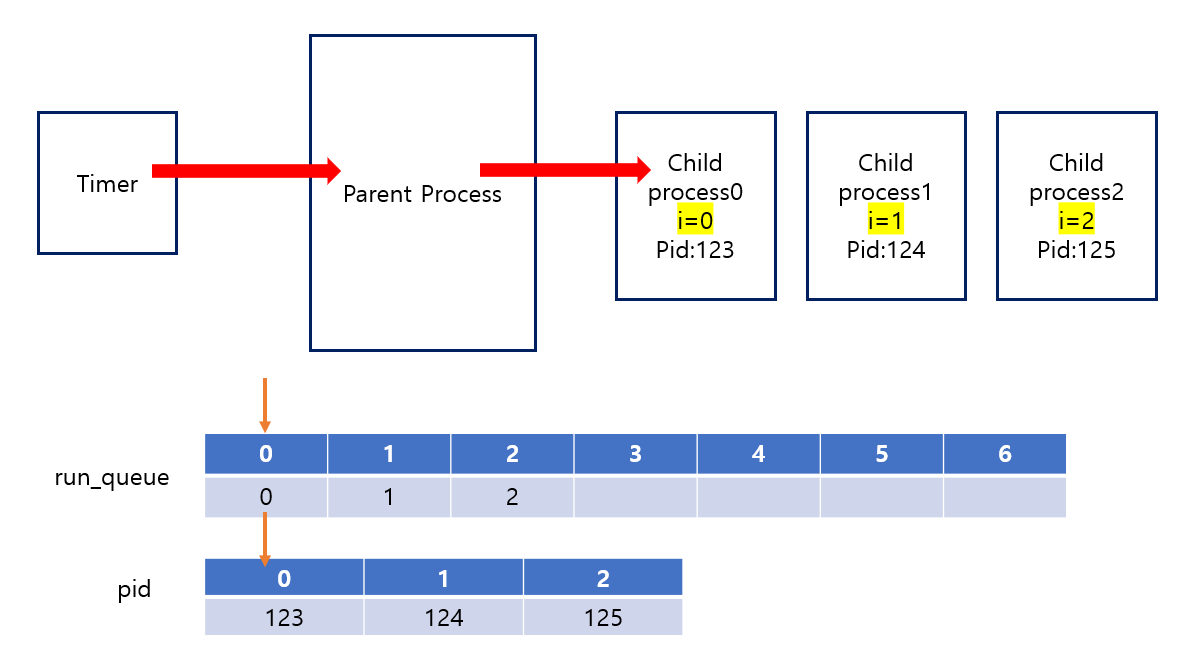
1. Parent 가 신호를 받으면 Parent 의 Handler에서 현재 실행되고 있는 프로세스에 kill을 통해 child process에 신호를 보내고, 현재 실행되고 있는 프로세스(User-process)의 cpu-burst 시간을 하나 줄여준다.
2. Kill을 통해 signal을 받은 자식 프로세스에서도 역시 cpu-burst를 하나 줄여준다.

🡪만약, cpu-burst가 0이 되면 프로세스 종료 시킨다. : exit()

1. Parent에서는 1 quantum(3 sec) 이 지나면, 큐에 있는 다른 프로세스를 실행시킨다. **<SCHEDULING**>

🡪 만약 현재 실행되고 있던 프로세스가 1 quantum이 지나도 cpu-burst가 0이 아니라면 다시 큐에 넣어서 이 후 다시 CPU에서 실행되도록 한다.

🡪만약 1 quantum이 다 지나기 전에, 한 프로세스 실행이 끝난다면 count를 초기화해 다음 프로세스가 다시 3초간(1 quantum)동안 실행되도록 한다.



🡪각 child process 에서는 i 값이 고유값이고, pid 값 보다는 index로 저장, 접근하는 것이 더 간편하여 run\_queue에는 pid array의 index를 저장해두었다.

1. **Round Robin + I/O**
2. Timer를 설정하여 매 초마다 parent(kernel)가 신호를 받게 된다.

🡪 setitimer 로 타이머 설정, sigaction 설정으로 signal 처리

1. Parent 가 신호를 받으면

🡪I/O가 실행되고 있는지 wait\_queue를 확인하여, 있으면 I/O- burst를 하나씩 줄여준다.

🡪Parent 의 Handler에서 현재 실행되고 있는 프로세스에 kill을 통해 child process에 신호를 보내고, 현재 실행되고 있는 프로세스(User-process)의 cpu-burst 시간을 하나 줄여준다.

1. Kill을 통해 signal을 받은 자식 프로세스에서도 역시 cpu-burst를 하나 줄여준다.

🡪만약, cpu-burst가 메시지큐로 메시지를 보내 I/O를 할 것을 parent 가 알도록 해준다. : msgsnd()

1. Parent에서는 1 quantum(3 sec) 이 지나면, 큐에 있는 다른 프로세스를 실행시킨다. **<SCHEDULING**>

🡪 만약 현재 실행되고 있던 프로세스가 1 quantum이 지나도 cpu-burst가 0이 아니라면 다시 큐에 넣어서 이 후 다시 CPU에서 실행되도록 한다.

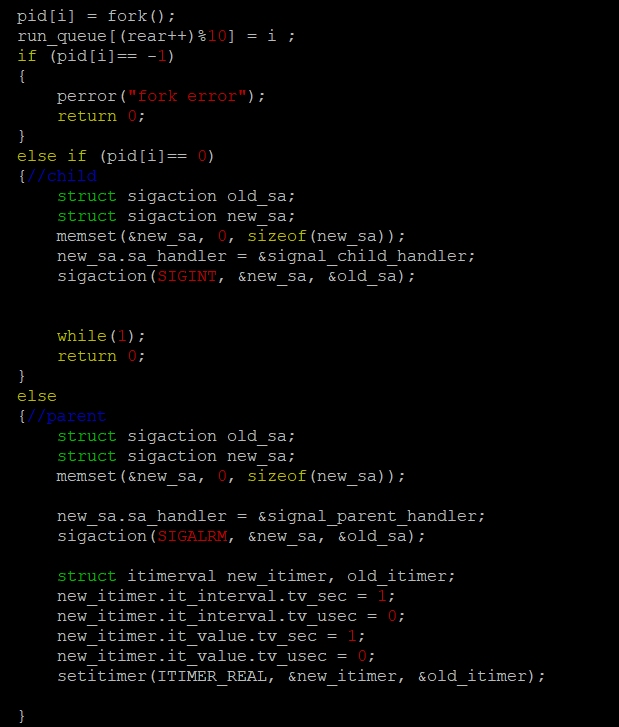
🡪만약 1 quantum이 다 지나기 전에, 한 프로세스 실행이 끝난다면 count를 초기화해 다음 프로세스가 다시 3초간(1 quantum)동안 실행되도록 한다.

🡪만약 현재 실행되던 프로세스가 cpu-burst가 0이 된다면 다시 시간을 채워준다. (I/O는 프로세스가 끝나면 안되기 때문에)

1. Parent가 child로부터 메시지를 받게 되면, child가 보낸 메시지의 정보 (i/o time 등)를 변수에 저장해준다. :msgrcv()

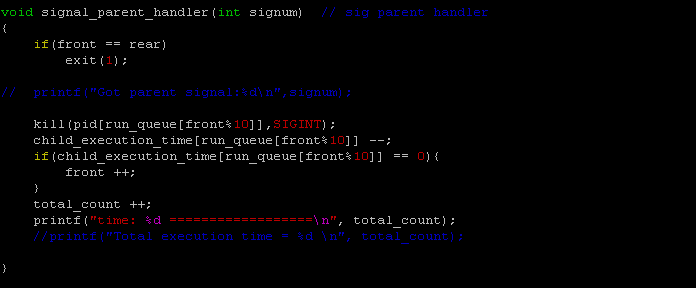
\*Index로 저장하는 구조는 동일함.

1. **소스코드 및 부연설명**
2. **FIFO (fifo\_sched.c)**
3. Main.c



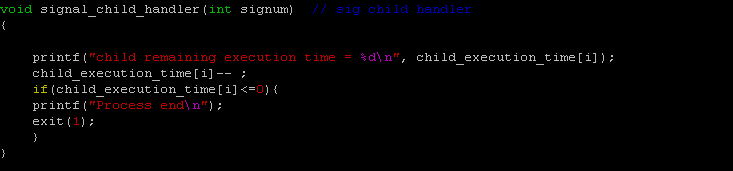
1. Parent signal handler:

To get signal from child processes once its execution time reaches 0 and schedule the next child process.

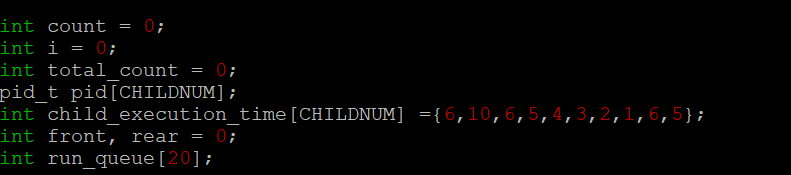


1. Child signal handler:

Get signal from parent to start execution, its execution time keep decreasing until it reaches 0.



1. **Round Robin(scheduling.c)**
   * + 1. 변수설정



-count는 1 quantum을 check 하기 위한 변수임

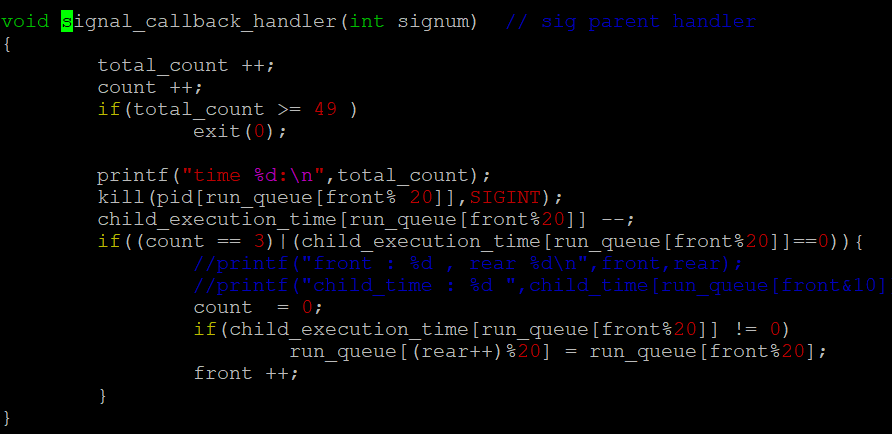
-i는 child 가 가질 고유의 값

-CHILDNUM은 10으로 정의 되어 있음

* + - 1. main함수



1. 먼저 fork 를 통해 10 개의 프로세스들을 만들어주고, run\_queue에 각각의 i 값(index)을 넣어 줌
2. Child는 signal이 들어오면 signal\_user\_handler 함수를 실행하게 됨
3. Parent는 timer을 설정하고 signal이 들어오면 signal\_callback\_handler를 실행하게 됨.
   * + 1. Signal\_callback\_handler 함수(Parent signal handler)



1. Signal을 받을 때 마다 count를 증가시켜 줌

-total count가 전체 시간을 넘어가면 종료시켜 줌

-count는 1 time quantum을 재기 위한 것임

1. 현재 실행중인 프로세스에다가 kill 함수를 통해 signal 보내줌

-run queue에 저장되어 있는 것이 pid index임

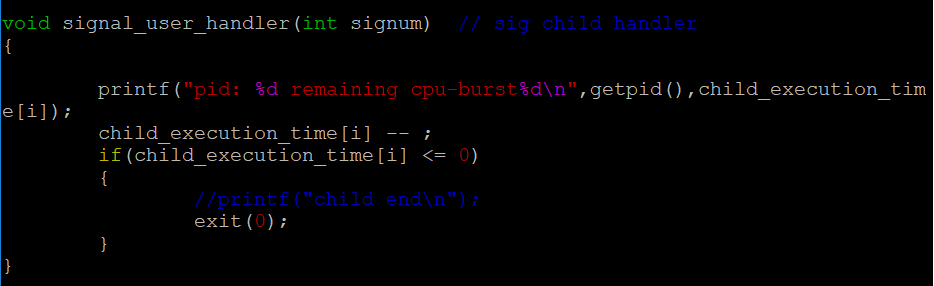
1. 만약 1 quantum이 지났거나, 현재 프로세스의 cpu-burst가 0이라면

-count를 초기화 함

-만약 현재 프로세스의 cpu-burst가 0 이 아니라면 다시 run queue에 넣어 줌

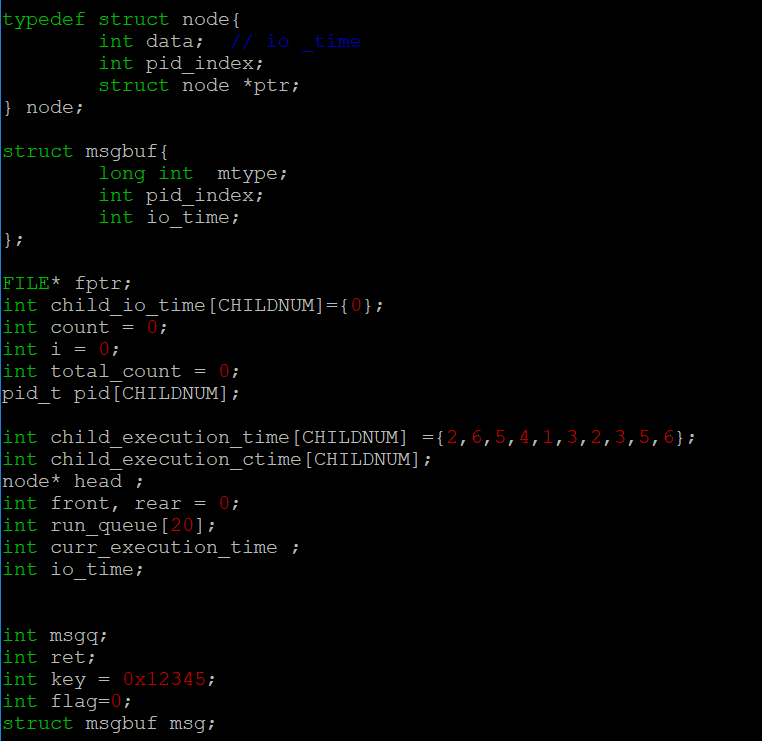
-다음 프로세스를 실행시킴

* + - 1. signal\_user\_handler 함수(Child signal handler)



만약 현재 프로세스의 cpu-burst가 0이라면 종료 시킴

1. **Round Robin + io (sched\_io.c)**
2. 변수설정

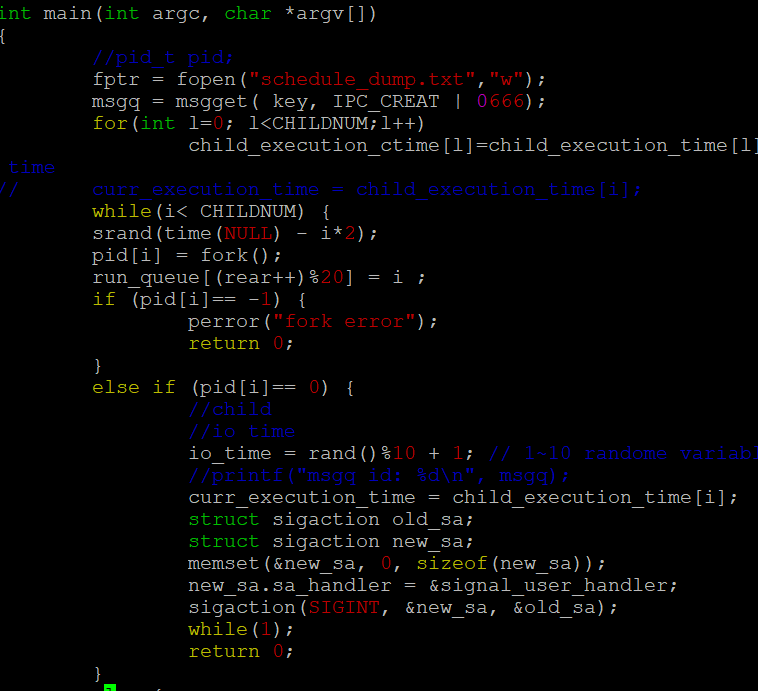


1. i/o는 큐로 관리 할 수 없기 때문에 sorted linked list로 구현해야 함.

왜냐하면, io-burst가 작은 것부터 나가야 하기 때문임

Linked list구현 하기 위한 node임

1. parent와 child 간의 message 전달이 필요하며, 이를 위해 message queue 구현이 필요함. Msg를 다루기 위한 변수임.
2. main함수

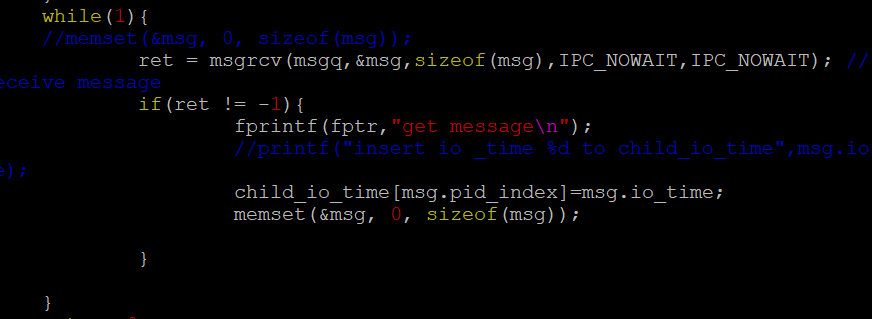


-scheduling.c 와 같은 구조임

- msgq를 사용하기 위한 초기화 코드가 추가 됨

- io 에 random한 값을 넣어주기 위해 srand, rand()를 사용하여 각 process가 다른 random 값을 갖도록 해줌 🡪 만약 srand(time(NULL))만 사용하면 모든 child process가 같은 random 값을 갖게 됨

\*parent process while(1); 에서 다음과 같이 변동됨

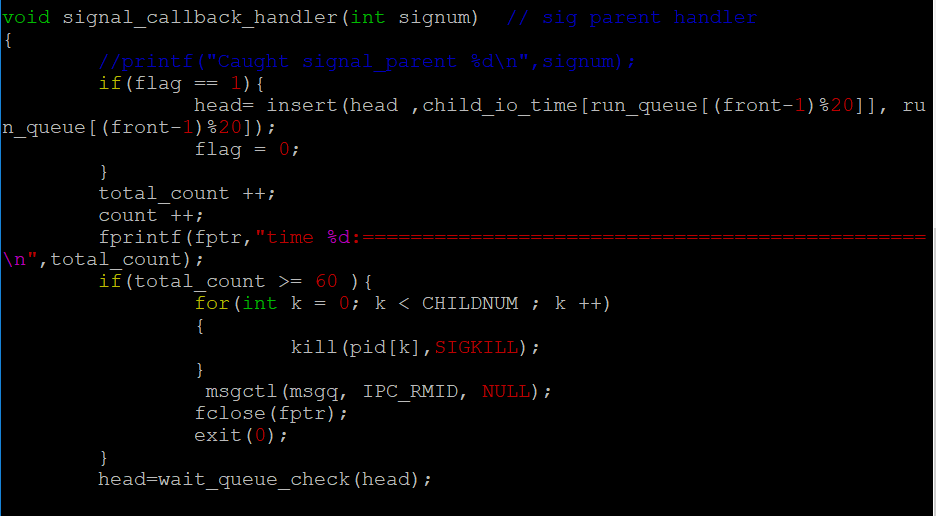


-Parent 가 message 가 있나 없나 계속해서 확인해야 하므로 while 안에 msgrcv함수를 사용하여 message를 받도록 함.

-IPC\_NOWAIT를 사용하여 없으면 바로 ret 에 -1을 return 하도록 함.

-만약 메시지가 있다면 child\_io\_time에 io\_time을 설정 해 줌.

1. Signal\_callback\_handler 함수

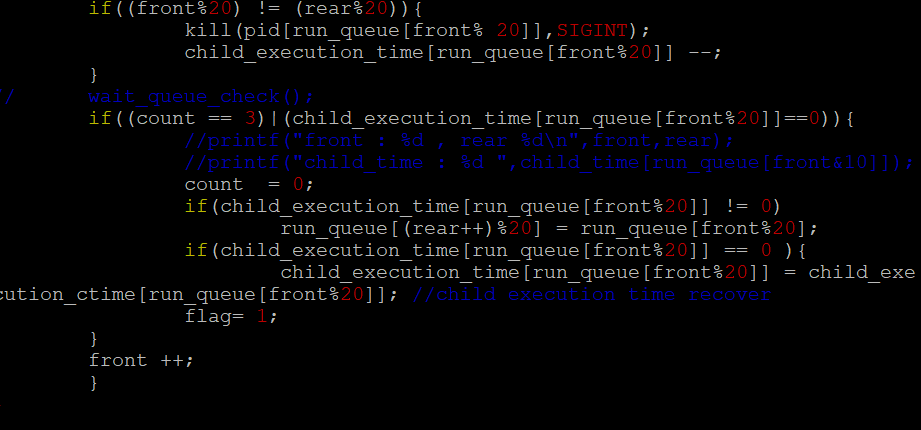


1. flag값을 사용한 이유는 signal에 순서가

Parent 🡪 Child 🡪 message 순이기 때문에, child 에서 message를 보내도 이가 현재 tick에서 반영이 될 수 가 없기 때문에, 다음 턴에서 바로 반영해줌

* insert 코드는 sorted linked list 구현 코드임 (open source 참고)

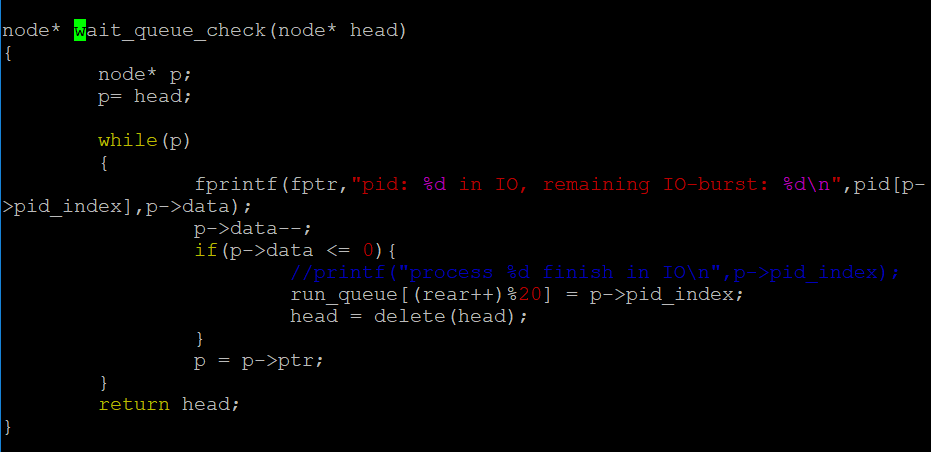
1. 이 코드는 child가 스스로 exit하는 코드가 없으므로 시그널을 통해 모두 종료되도록 함
2. i/o에 처리해야 할 것이 있는 지 확인 하는 코드임



-현재 실행할 process 가 없는 경우를 방지하기 위해 empty일 때는 실행하지 않음

-만약 현재 process cpu-burst가 0일 경우, i/o 들어가지만 그전에 다시 recover 해줘 i/o 끝나면 다시 running queue로 들어가도록 함

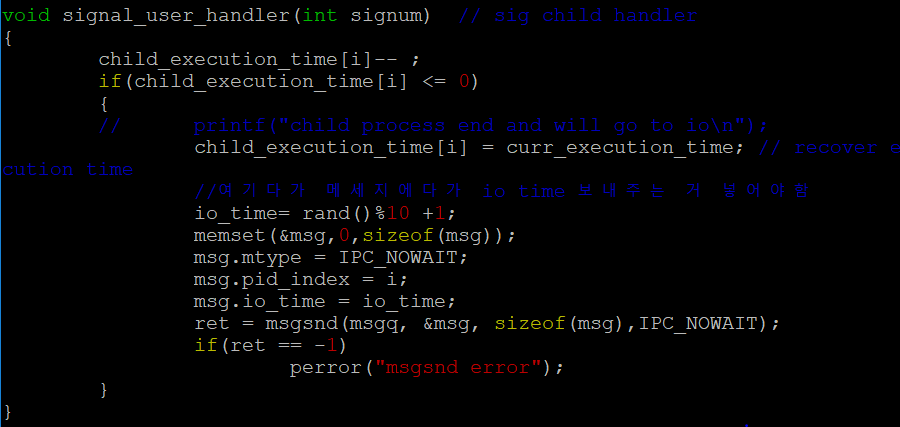
4. wait\_queue\_check함수



-만약 i/o 처리해야 할 것이 있다면 모든 node의 io\_time을 하나씩 줄여 줌

-만약 io burst가 0일경우, delete하고 그 프로세스를 run queue에 올려줌

5. signal\_user\_handler 함수



-만약 현재 프로세스의 cpu-burst가 0일 경우 msgq를 사용하여 message를 보냄

-io burst와, index 값을 전달 함

1. **결과**
2. 결과
3. fifo\_sched.c

Build 환경

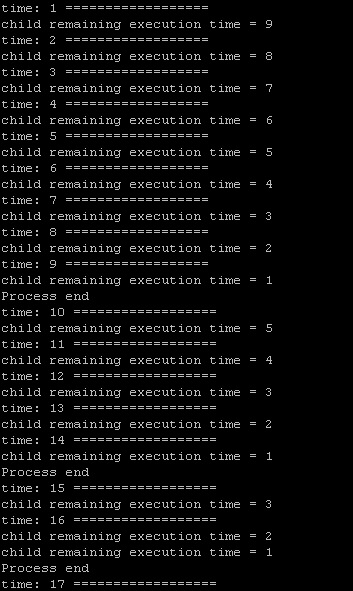
-Compilation: Linux Assam, with GCC

-To compile, please type: gcc fifo\_sched.c

-To run, please type: ./a.out

result

(For example, having three processes)



1. scheduling .c

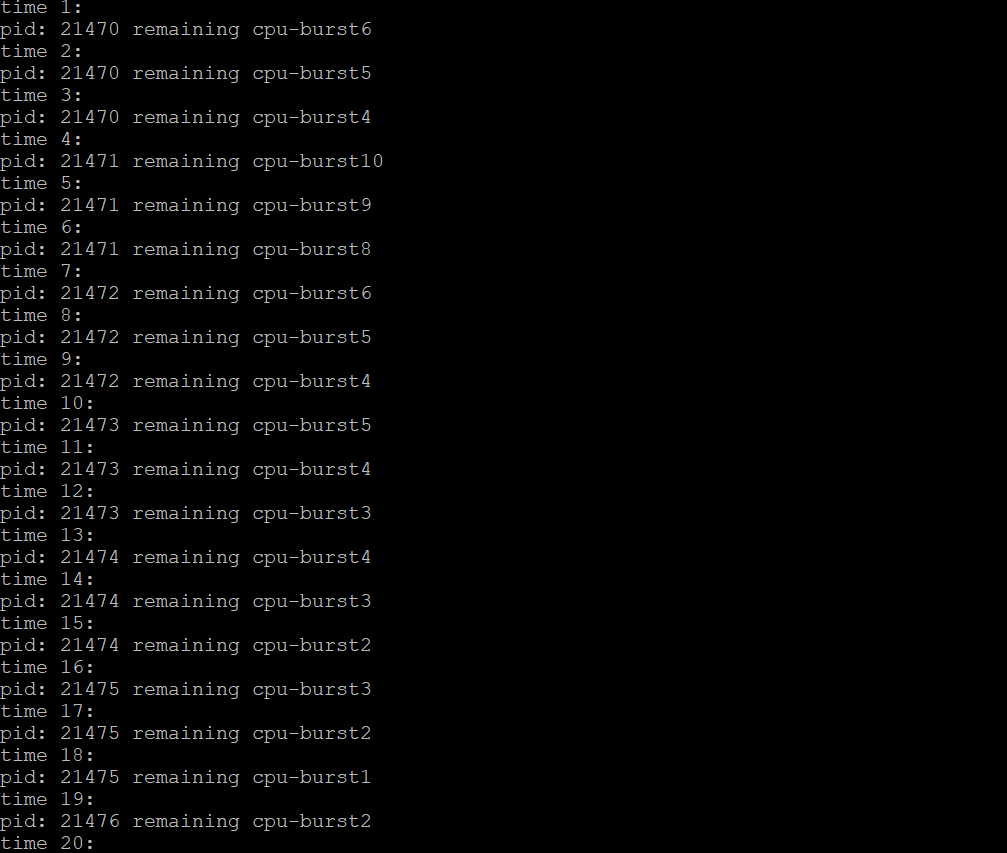
Build 환경

-Compilation: Linux Assam, with GCC

-To compile, please type: gcc scheduling.c -o scheduling

-To run, please type: ./scheduling

result



1. sched\_io.c

Build 환경

-Compilation: Linux Assam, with GCC

-To compile, please type: gcc sched\_io.c -o iocode

-To run, please type: ./iocode

result(schedule\_dump.txt 저장)

