

# 운영체제

Assignment3

최상호 교수님

컴퓨터정보공학부

2018202076

이연걸

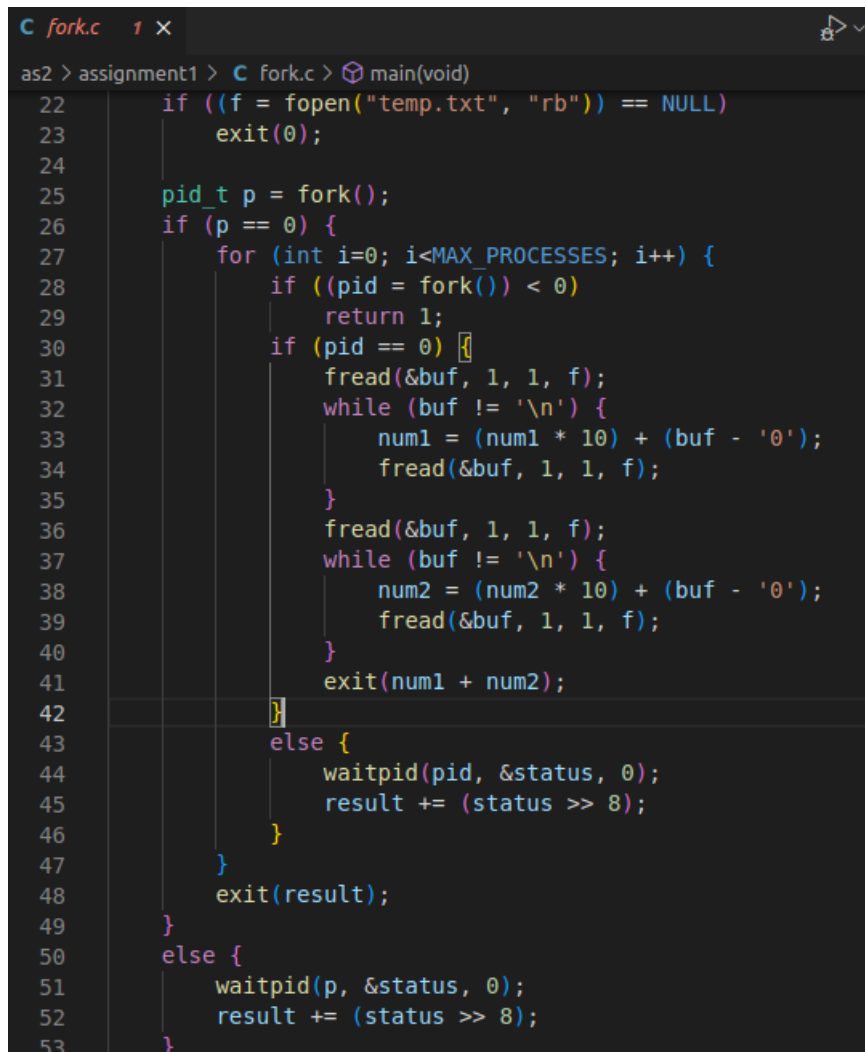
## ● Introduction

본 과제는 프로세스, 스레드에 대한 이해가 필수적인 과제다. 3-1에서는 멀티 프로세스, 멀티 스레드를 활용해 파일 읽기, 및 연산 작업을 수행한다. 모든 작업에 대해 프로세스를 생성하는 것과 스레드를 생성하는 것의 속도차이를 비교해야 한다. 3-2에서는 앞서 실행한 작업과 유사한 작업을 CPU 스케줄링 정책을 변경해가며 실행한다. CPU 스케줄링 간의 속도 차이를 비교하고 동일 스케줄링 정책에서도 Priority, Nice 값에 따른 속도 차이도 측정한다. 3-3에서는 현재 프로세스의 이름, 상태, 그룹 정보, context switching 횟수, fork 횟수 등 현재 프로세스와 관련된 정보, 즉 `task_struct` 구조체에 있는 정보를 알아내기 위한 작업을 하며 `task_struct` 구조체에 없는 정보는 추가해 주는 작업을 실행한다. 본 과제는 리눅스에서 프로세스와 스레드를 깊이 이해하기 위해 초점을 맞춘다.

## ● Conclusion & Analysis

### ■ 3-1

#### CODE



```
as2 > assignment1 > C fork.c > main(void)
22     if ((f = fopen("temp.txt", "rb")) == NULL)
23         exit(0);
24
25     pid_t p = fork();
26     if (p == 0) {
27         for (int i=0; i<MAX_PROCESSES; i++) {
28             if ((pid = fork()) < 0)
29                 return 1;
30             if (pid == 0) {
31                 fread(&buf, 1, 1, f);
32                 while (buf != '\n') {
33                     num1 = (num1 * 10) + (buf - '0');
34                     fread(&buf, 1, 1, f);
35                 }
36                 fread(&buf, 1, 1, f);
37                 while (buf != '\n') {
38                     num2 = (num2 * 10) + (buf - '0');
39                     fread(&buf, 1, 1, f);
40                 }
41                 exit(num1 + num2);
42             }
43             else {
44                 waitpid(pid, &status, 0);
45                 result += (status >> 8);
46             }
47         }
48         exit(result);
49     }
50     else {
51         waitpid(p, &status, 0);
52         result += (status >> 8);
53     }
```

fork.c 파일의 일부분을 가져온 것이다. Child process 에서 exit status 를 waitpid 에서 가져오고 8bit 를 shift 해주는 것을 확인할 수 있다. Child process 가 exit()로 값을 반환하며 정상 종료되었다면 waitpid 내 &status 의 상위 8 비트에 exit status value 가 저장된다. 하위 8 비트에는 0 이 저장되며 실제 값을 찾기 위해서는 8 비트를 right shift 해주어야 한다.

**결과 화면 MAX\_PROCESSES 8:**

```

os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $rm -rf tmp*
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $sync
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $ make
gcc -o numgen numgen.c
gcc -o fork fork.c
gcc -o thread thread.c -pthread
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $ ./numgen
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $ ./fork
value of fork : 136
0.015857
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $ ./thread
value of fork : 136
0.020610
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
○ $ █

```

결과 화면 MAX\_PROCESSES 64:

```

os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
○ $
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $ rm -rf tmp*
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $ sync
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $ make
gcc -o numgen numgen.c
gcc -o fork fork.c
gcc -o thread thread.c -pthread
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $ ./numgen
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $ ./fork
value of fork : 64
0.044303
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
● $ ./thread
value of fork : 8256
0.040853
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment1
○ $ █

```

결과를 살펴보자 캐시 와 버퍼를 먼저 비워서 실험에 영향을 주는 요소를 제거했다. 그 후 Makefile 로 세 파일을 컴파일 했고 numgen 으로 파일 탐색에 사용할 파일을 만들고 fork 프로그램으로 멀티 프로세스 작업, thread 프로그램으로 멀티 스레드 작업을 실행했다.

반면 최대 프로세스/스레드의 개수가 64 개 일 때는 프로세스보다 스레드의 속도가 더 빨랐다. 동일한 작업을 반복하는데 반복 횟수가 많아지면서 code, data 영역을 공유하는 스레드가 자원을 더 효율적으로 사용해 반복 횟수를 줄인 것으로 생각된다.

```

26
27 // OTHER : can only be used at status priority 0
28 param.sched_priority = 0;
29
30 pid_t p = fork();
31 if (p == 0) {
32     for (int i=0; i<MAX_PROCESSES; i++) {
33         char file_name[10000] = {'t','e','m','p','/',' ',
34             sprintf(num, "%d", i);
35             FILE *f = fopen(strcat(file_name, num), "rb");
36             if ((pid = fork()) < 0)
37                 return 1;
38             sched_setscheduler(0, SCHED_OTHER, &param);
39             nice(-20); // max nice -20
40             if (pid == 0) {
41                 fread(&buf, 1, 1, f);
42                 exit(0);
43             }
44             else {
45                 waitpid(pid, &status, 0);
46             }
47             fclose(f);
48     }
49     exit(0);
50 }

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL bash - assignment2 + v [ ] [ ] [ ]

```

os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$ rm -rf tmp*
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$ sync
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$ ./schedtest
0.706356
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$

```

```
as2 > assignment2 > C schedtest.c > main(void)
26
27 // OTHER : can only be used at status priority 0
28 param.sched_priority = 0;
29
30 pid_t p = fork();
31 if (p == 0) {
32     for (int i=0; i<MAX_PROCESSES; i++) {
33         char file_name[10000] = {'t','e','m','p','/','\0'};
34         sprintf(num, "%d", i);
35         FILE *f = fopen(strcat(file_name, num), "rb");
36         if ((pid = fork()) < 0)
37             return 1;
38         sched_setscheduler(0, SCHED_OTHER, &param);
39         // default nice
40         if (pid == 0) {
41             fread(&buf, 1, 1, f);
42             exit(0);
43         }
44         else {
45             waitpid(pid, &status, 0);
46         }
47         fclose(f);
48     }
49     exit(0);
50 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL bash - assignment2 + - X

```
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ rm -rf tmp*
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ sync
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ ./schedtest
0.632312
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
○ $
```

```
as2 > assignment2 > C schedtest.c > main(void)
26
27 // OTHER : can only be used at status priority 0
28 param.sched_priority = 0;
29
30 pid_t p = fork();
31 if (p == 0) {
32     for (int i=0; i<MAX_PROCESSES; i++) {
33         char file_name[10000] = {'t','e','m','p','/','/','\0'};
34         sprintf(num, "%d", i);
35         FILE *f = fopen(strcat(file_name, num), "rb");
36         if ((pid = fork()) < 0)
37             return 1;
38         sched_setscheduler(0, SCHED_OTHER, &param);
39         nice(19); // min nice 19
40         if (pid == 0) {
41             fread(&buf, 1, 1, f);
42             exit(0);
43         }
44         else {
45             waitpid(pid, &status, 0);
46         }
47         fclose(f);
48     }
49     exit(0);
50 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL bash - assignment2

```
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ rm -rf tmp*
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ sync
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ ./schedtest
0.480952
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
○ $
```

결과 화면 FIFO -> max priority, default priority, min priority:

A first-in first-out policy



```
as2 > assignment2 > C schedtest.c > main(void)
```

```

27 // FIFO : max Priority (99)
28 param.sched_priority = sched_get_priority_max(SCHED_FIFO);
29
30 pid_t p = fork();
31 if (p == 0) {
32     for (int i=0; i<MAX_PROCESSES; i++) {
33         char file_name[10000] = {'t','e','m','p',' ','/','\0'};
34         sprintf(num, "%d", i);
35         FILE *f = fopen(strcat(file_name, num), "rb");
36         if ((pid = fork()) < 0)
37             return 1;
38         sched_setscheduler(0, SCHED_FIFO, &param);
39         if (pid == 0) {
40             fread(&buf, 1, 1, f);
41             exit(0);
42         }
43         else {
44             waitpid(pid, &status, 0);
45         }
46         fclose(f);
47     }
48     exit(0);

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL bash - assignment2 + v [ ] [X] ^ >

```
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$ rm -rf tmp*
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$ sync
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$ ./schedtest
0.223345
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$
```



```
as2 > assignment2 > C schedtest.c > main(void)
27
28 // FIFO : default Priority
29
30 pid_t p = fork();
31 if (p == 0) {
32     for (int i=0; i<MAX_PROCESSES; i++) {
33         char file_name[10000] = {'t','e','m','p','/','\0'};
34         sprintf(num, "%d", i);
35         FILE *f = fopen(strcat(file_name, num), "rb");
36         if ((pid = fork()) < 0)
37             return 1;
38         sched_setscheduler(0, SCHED_FIFO, &param);
39         if (pid == 0) {
40             fread(&buf, 1, 1, f);
41             exit(0);
42         }
43         else {
44             waitpid(pid, &status, 0);
45         }
46         fclose(f);
47     }
48     exit(0);
49 }
50 else {
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL bash - assignment2 + -

```
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영 체제/as2/assignment2
• $ rm -rf tmp*
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영 체제/as2/assignment2
• $ sync
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영 체제/as2/assignment2
• $ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영 체제/as2/assignment2
• $ ./schedtest
0.181709
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영 체제/as2/assignment2
○ $
```

```
as2 > assignment2 > C schedtest.c > main(void)
26
27 // FIFO : min Priority (1)
28 param.sched_priority = sched_get_priority_min(SCHED_FIFO)
29
30 pid_t p = fork();
31 if (p == 0) {
32     for (int i=0; i<MAX_PROCESSES; i++) {
33         char file_name[10000] = {'t','e','m','p','/','\0'}
34         sprintf(num, "%d", i);
35         FILE *f = fopen(strcat(file_name, num), "rb");
36         if ((pid = fork()) < 0)
37             return 1;
38         sched_setscheduler(0, SCHED_FIFO, &param);
39         if (pid == 0) {
40             fread(&buf, 1, 1, f);
41             exit(0);
42         }
43         else {
44             waitpid(pid, &status, 0);
45         }
46         fclose(f);
47     }
48     exit(0);
49 }
```

PROBLEMS 1 OUTPUT TERMINAL ... bash - assignment2

```
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ rm -rf tmp*
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ sync
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ ./schedtest
0.278625
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
○ $
```

결과 화면 RR -> max priority, default priority, min priority:

A round-robin policy

```
as2 > assignment2 > C schedtest.c > main(void)
```

```

26
27 // RR : max Priority (99)
28 param.sched_priority = sched_get_priority_max(SCHED_RR);
29
30 pid_t p = fork();
31 if (p == 0) {
32     for (int i=0; i<MAX_PROCESSES; i++) {
33         char file_name[10000] = {'t','e','m','p','/', '\0'};
34         sprintf(num, "%d", i);
35         FILE *f = fopen(strcat(file_name, num), "rb");
36         if ((pid = fork()) < 0)
37             return 1;
38         sched_setscheduler(0, SCHED_RR, &param);
39         if (pid == 0) {
40             fread(&buf, 1, 1, f);
41             exit(0);
42         }
43         else {
44             waitpid(pid, &status, 0);
45         }
46         fclose(f);
47     }
48     exit(0);

```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL bash - assignment2 + v [ ] [ ] [ ] [ ] [ ]

```
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
● $ rm -rf tmp*
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
● $ sync
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
● $ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
● $ ./schedtest
0.944977
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
○ $
```

```
as2 > assignment2 > C schedtest.c > main(void)
27
28 // RR : default Priority
29
30 pid_t p = fork();
31 if (p == 0) {
32     for (int i=0; i<MAX_PROCESSES; i++) {
33         char file_name[10000] = {'t','e','m','p','/',' '};
34         sprintf(num, "%d", i);
35         FILE *f = fopen(strcat(file_name, num), "rb");
36         if ((pid = fork()) < 0)
37             return 1;
38         sched_setscheduler(0, SCHED_RR, &param);
39         if (pid == 0) {
40             fread(&buf, 1, 1, f);
41             exit(0);
42         }
43         else {
44             waitpid(pid, &status, 0);
45         }
46         fclose(f);
47     }
48     exit(0);
49 }
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL bash - assignment2 + - [ ] [ ] ^

```
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ rm -rf tmp*
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ sync
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
• $ ./schedtest
0.624926
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
○ $
```



```
as2 > assignment2 > C schedtest.c > main(void)
26
27 // RR : min Priority (1)
28 param.sched_priority = sched_get_priority_min(SCHED_RR);
29
30 pid_t p = fork();
31 if (p == 0) {
32     for (int i=0; i<MAX_PROCESSES; i++) {
33         char file_name[10000] = {'t','e','m','p','/','\0'}
34         sprintf(num, "%d", i);
35         FILE *f = fopen(strcat(file_name, num), "rb");
36         if ((pid = fork()) < 0)
37             return 1;
38         sched_setscheduler(0, SCHED_RR, &param);
39         if (pid == 0) {
40             fread(&buf, 1, 1, f);
41             exit(0);
42         }
43         else {
44             waitpid(pid, &status, 0);
45         }
46         fclose(f);
47     }
48     exit(0);
}

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL bash - assignment2 + - -
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$ rm -rf tmp*
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$ sync
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$ echo 3 | sudo tee /proc/sys/vm/drop_caches
3
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$ ./schedtest
0.909014
os2018202076@ubuntu:~/Desktop/KW-Univ/3학년 2학기/운영체제/as2/assignment2
$
```

작업의 개수, 즉 프로세스의 개수를 30000 으로 설정하고 실험을 진행하였다. sched\_setscheduler()를 사용하면 각 프로세스에서 CPU 스케줄링 정책을 변경할 수 있다. 스케줄링 정책 별 평균 소요 시간을 비교하면 다음과 같다.

The standard round-robin time-sharing policy: 0.606540

A first-in first-out policy: 0.227893

A round-robin policy: 0.826305

빠른 순서대로 FIFO -> Standard RR -> RR 순이다. 동일한 작업을 반복하는 것이기 때문에 작업 당 실행시간이 거의 동일하다. 그렇기 때문에 context switching 을 최대한 적게 하는 FIFO 방식이 가장 빠른 것으로 생각된다. 그 다음은 기본적으로 여러 프로세스 간 Time-sharing 방식(CFS Scheduler) 또는 우선순위 기반 Scheduling 을 사용해 작업을 수행하는 The standard round-robin time-sharing policy 였는데 FIFO 보다는 더 많은

context switching 을 하기 때문에 2 번째로 빠른 속도를 가진 것으로 생각된다. 마지막으로 Round Robin 이 있다. RR 은 Context-switching 이 많지만 response time 이 짧아지는 장점을 갖고 있어 실시간 시스템에 적합하다. 하지만 본 과제에서 수행하는 작업은 실시간 시스템이 아니기 때문에 잦은 context-switching 으로 얻을 수 있는 장점보다. Overhead 가 매우 커서 세 정책 중 가장 비효율 적인 것으로 생각된다.

이 밖에도 FIFO, RR 은 priority 를 다르게 해가면서 실행시간을 측정했다. 이들은 1 에서 99 까지의 priority value 를 갖지만 sched\_get\_priority\_max 혹은 sched\_get\_priority\_min 을 사용해 그 값을 다르게 한다고 해도 프로그램 실행마다 값의 변화가 크지 않아 특별한 차이를 찾기 어려웠다.

OTHER 의 경우는 nice 값을 다르게 해가며 실행시간을 측정했다. nice 는 priority 와는 반대로 높은 nice value 를 갖으면 낮은 priority 를 갖게 된다. nice 는 -19 ~ 20 까지의 값을 갖을 수 있지만 값에 따라 변화되는 시간의 양이 크지 않거나 완전히 무관한 값이 나오는 등 nice 값이 프로그램 실행시간에 변화를 준다고 보기 어려웠다.

### ■ 3-3

3-3 과제는 PID 를 바탕으로 프로세스의 정보를 출력하는 Module 을 작성하는 것이다. 리눅스에서 프로세스의 정보는 task\_struct 구조체에 있는데 이 구조체는 sched.h 헤더파일에 선언되어 있어 헤더 파일 코드를 분석해 필요한 정보를 뽑아내는 과제였다.

Pid 와 process 이름, 부모 프로세스 정보, context switching 횟수는 각각 pid, comm, parent, nvcsw 와 nivcsw 에 담겨 있다. 하지만 task\_struct 에 fork 횟수를 count 해주는 필드는 존재하지 않는데 이것을 추가해 주어야 한다.

이를 위해 sched.h 을 수정해 fork 횟수를 세어주는 fork\_coount 필드를 추가했다.

```

C sched.h > task_struct > fork_count
1193     /* A live task holds one reference: */
1194     atomic_t      stack_refcount;
1195 #endif
1196 #ifdef CONFIG_LIVEPATCH
1197     int patch_state;
1198 #endif
1199 #ifdef CONFIG_SECURITY
1200     /* Used by LSM modules for access restriction: */
1201     void          *security;
1202 #endif
1203
1204     /*
1205      * New fields for task_struct should be added above here, so that
1206      * they are included in the randomized portion of task_struct.
1207      */
1208     int fork_count;
1209     randomized_struct_fields_end
1210
1211     /* CPU-specific state of this task: */
1212     struct thread_struct      thread;
1213

```

헤더 파일에 task\_struct 에 새로운 필드를 추가하고 싶다면 이곳에서 추가해주면 된다. 기존의 헤더파일에 이미 선언 위치를 명시해 두었다.

이제 과제에서 제시한 조건대로 fork()로 자식 프로세스를 생성할 때 해당 변수를 초기화 하고 fork 가 호출될 때 해당 변수 즉, 부모 프로세스의 fork\_count 값을 1 씩 증가시키면 된다. 이 작업이 실행되는 부분 또한 기존의 fork.c 파일에 명시되어 있다.

```

/*
 * Ok, this is the main fork-routine.
 *
 * It copies the process, and if successful kick-starts
 * it and waits for it to finish using the VM if required.
 */
long _do_fork(unsigned long clone_flags,
              unsigned long stack_start,
              unsigned long stack_size,
              int __user *parent_tidptr,
              int __user *child_tidptr,
              unsigned long tls)
{
    struct completion vfork;
    struct pid *pid;
    struct task_struct *p;
    int trace = 0;
    long nr;

    /*
     * Determine whether and which event to report to ptracer. When
     * called from kernel_thread or CLONE_UNTRACED is explicitly
     * requested, no event is reported; otherwise, report if the event
     * for the type of forking is enabled.
     */
    if (!(clone_flags & CLONE_UNTRACED)) {
        if (clone_flags & CLONE_VFORK)
            trace = PTRACE_EVENT_VFORK;
        else if ((clone_flags & CSIGNAL) != SIGCHLD)
            trace = PTRACE_EVENT_CLONE;
        else

```

이곳에 this is the main fork-routine 이 적혀 있고



```

2190     if (!(clone_flags & CLONE_UNTRACED)) {
2191         if (clone_flags & CLONE_VFORK)
2192             trace = PTRACE_EVENT_VFORK;
2193         else if ((clone_flags & CSIGNAL) != SIGCHLD)
2194             trace = PTRACE_EVENT_CLONE;
2195         else
2196             trace = PTRACE_EVENT_FORK;
2197
2198         if (likely(!ptrace_event_enabled(current, trace)))
2199             trace = 0;
2200     }
2201
2202     p = copy_process(clone_flags, stack_start, stack_size,
2203                     child_tidptr, NULL, trace, tls, NUMA_NO_NODE);
2204     add_latent_entropy();
2205
2206     if (IS_ERR(p))
2207         return PTR_ERR(p);
2208
2209     p->fork_count = 0;
2210     p->parent->fork_count++;
2211

```

자식(현재) 프로세스의 fork\_count 초기화와 fork 를 호출한 프로세스의 fork\_count 의 개수 증가까지 추가해 완성하였다. 이제 이 코드를 실제 커널에 최신화를 시켜주면 된다. 해당 c 파일과 헤더파일이 선언된 위치로 가서 파일을 최신화 해주고 커널 컴파일을 시작하면

```

os2018202076@ubuntu:~/Downloads/linux-4.19.67$ sudo make -j12
DESCEND  objtool
CC       arch/x86/kernel/asm-offsets.s
GEN      scripts/gdb/linux/constants.py
CALL     scripts/checksyscalls.sh
CC       arch/x86/events/core.o
CC       arch/x86/ia32/sys_ia32.o
CC       certs/system_keyring.o
CC       arch/x86/ia32/ia32_signal.o
CC       arch/x86/hyperv/hv_init.o
CC       init/main.o
CC       arch/x86/entry/syscall_64.o
CC [M]   arch/x86/kvm/../../../../virt/kvm/kvm_main.o
CC       arch/x86/crypto/crc32c-intel_glue.o
CC       kernel/fork.o
CC       arch/x86/entry/common.o
CC       mm/filemap.o
CC       arch/x86/kernel/process_64.o

```

```

Found linux image: /boot/vmlinuz-4.19.67-2018202076
Found initrd image: /boot/initrd.img-4.19.67-2018202076
Found linux image: /boot/vmlinuz-4.19.67-2018202076.old
Found initrd image: /boot/initrd.img-4.19.67-2018202076
Found linux image: /boot/vmlinuz-4.15.0-142-generic
Found initrd image: /boot/initrd.img-4.15.0-142-generic
Found linux image: /boot/vmlinuz-4.15.0-29-generic
Found initrd image: /boot/initrd.img-4.15.0-29-generic
Found mentest86+ image: /boot/mentest86+.elf
Found mentest86+ image: /boot/mentest86+.bin
done
os2018202076@ubuntu:~/Downloads/linux-4.19.67$

```

아래의 사진과 동일한 결과를 얻을 수 있다.

```

[ 335.986071] ##### TASK INFORMATION OF '[1] systemd' #####
[ 335.986074] - task state : Wait with ignoring all signals
[ 335.986075] - Process Group Leader: [1] systemd
[ 335.986075] - Number of context switches : 6343
[ 335.986076] - Number of calling fork() : 220
[ 335.986077] - it's parent process : [0] swapper/0
[ 335.986077] - it's sibling process(es) :
[ 335.986078] > [381] systemd-journal
[ 335.986079] > [405] systemd-udev
[ 335.986080] > [416] vmware-vmblock-
[ 335.986081] > [422] vntoolsd
[ 335.986082] > [505] systemd-timesyn
[ 335.986083] > [889] systemd-logind
[ 335.986084] > [890] cron
[ 335.986085] > [891] avahi-daemon
[ 335.986086] > [893] rsyslogd
[ 335.986087] > [896] dbus-daemon
[ 335.986088] > [923] NetworkManager
[ 335.986089] > [928] accounts-daemon
[ 335.986090] > [930] cupsd
[ 335.986091] > [935] VGAuthService
[ 335.986091] > [936] cups-browsed
[ 335.986092] > [937] runsvdir
[ 335.986093] > [956] acpid
[ 335.986094] > [975] agetty
[ 335.986095] > [979] lightdm
[ 335.986095] > [997] irqbalance
[ 335.986097] > [1003] polkitd
[ 335.986098] > [1286] rtkit-daemon
[ 335.986098] > [1307] upowerd
[ 335.986099] > [1321] colord
[ 335.986100] > [1405] systemd
[ 335.986101] > [1443] gnome-keyring-d
[ 335.986102] > [1580] whoopsie
[ 335.986103] > [1715] apache2
[ 335.986103] > [2028] udiskd
[ 335.986104] > [2039] fwupd
[ 335.986105] > [2701] aptd
[ 335.986106] > This process has 31 sibling process(es)
[ 335.986106] - it's child process(es) :
[ 335.986107] > [381] systemd-journal
[ 335.986107] > [405] systemd-udev
[ 335.986107] > [416] vmware-vmblock-
[ 335.986108] > [422] vntoolsd
[ 335.986108] > [505] systemd-timesyn
[ 335.986109] > [889] systemd-logind
[ 335.986109] > [890] cron
[ 335.986110] > [891] avahi-daemon
[ 335.986110] > [893] rsyslogd
[ 335.986111] > [896] dbus-daemon
[ 335.986111] > [923] NetworkManager
[ 335.986112] > [928] accounts-daemon

```

```

335.986106] - it's child process(es) :
335.986107] > [381] systemd-journal
335.986107] > [405] systemd-udevd
335.986107] > [416] vmware-vmtoolsd
335.986108] > [422] vmtoolsd
335.986108] > [505] systemd-timesyn
335.986109] > [889] systemd-logind
335.986109] > [890] cron
335.986110] > [891] avahi-daemon
335.986110] > [893] rsyslogd
335.986111] > [896] dbus-daemon
335.986111] > [923] NetworkManager
335.986112] > [928] accounts-daemon
335.986112] > [930] cupsd
335.986113] > [935] VGAuthService
335.986113] > [936] cups-browsed
335.986114] > [937] runsvdir
335.986114] > [956] acpid
335.986114] > [975] agetty
335.986115] > [979] lightdm
335.986115] > [997] irqbalance
335.986116] > [1003] polkitd
335.986116] > [1286] rtkit-daemon
335.986117] > [1307] upowerd
335.986117] > [1321] colord
335.986118] > [1405] systemd
335.986118] > [1443] gnome-keyring-d
335.986119] > [1580] whoopsie
335.986119] > [1715] apache2
335.986120] > [2028] udisksd
335.986120] > [2039] fwupd
335.986120] > [2701] aptd
335.986121] > This process has 31 child process(es)
335.986121] ##### END OF INFORMATION #####
376.948389] perf: interrupt took too long (6453 > 5290), lowering kernel.perf_event_m
_rate to 30750
os2018202076@ubuntu:~$

```

테스트 함수:

```

C main.c > main(void)
1  #include <linux/unistd.h>
2  #include <unistd.h>
3
4  #define __NR_ftrace 336
5
6  int main(void) {
7      syscall(__NR_ftrace, 1);
8      return 0;
9  }

```

Process tracing 할 process 를 1 로 설정했기 때문에 위에서 보이는 것처럼 프로세스 1 에 해당하는 정보를 가져오는 것을 확인할 수 있다.

## ● 고찰

이번 과제를 수행하면서 여러 어려운 점이 있었다.

첫번째는 exit()의 반환 값이 8 비트로 표현할 수 있는 범위를 넘을 때 하위 8 비트만 가져와 해당 8 비트에 0 으로 8 비트를 채워 부모 프로세스로 넘겨주는데 이것을 제대로 캐치하지 못한 것이다. 과제에서 부모 프로세스와 자식 프로세스는 다른 작업을 한다고 명시해 주었기 때문에 부모 프로세스에서 파일에 적힌 값을 더하는 것이 아닌, 자식 프로세스에서 값을 더해 부모 프로세스로 넘겨주는 방식으로 해결할 수 있었다. 처음 코드는 다음과 같았다.

```

if ((f = fopen("temp.txt", "rb")) == NULL)
    exit(0);

for (int i=0; i<MAX_PROCESSES; i++) {
    if ((pid = fork()) < 0)
        return 1;
    if (pid == 0) {
        fread(&buf, 1, 1, f);
        while (buf != '\n') {
            num1 = (num1 * 10) + (buf - '0');
            fread(&buf, 1, 1, f);
        }
        fread(&buf, 1, 1, f);
        while (buf != '\n') {
            num2 = (num2 * 10) + (buf - '0');
            fread(&buf, 1, 1, f);
        }
        exit(num1 + num2);
    }
    else {
        waitpid(pid, &status, 0);
        result += (status >> 8);
    }
}
clock_gettime(CLOCK_REALTIME, &stop);
printf("value of fork : %d\n", result);
printf("%lf\n", (double)((stop.tv_sec - start.tv_sec) +

```

이 코드를 수정해 다음과 같이 고칠 수 있었다.

```

if ((f = fopen("temp.txt", "rb")) == NULL)
    exit(0);

pid_t p = fork();
if (p == 0) {
    for (int i=0; i<MAX_PROCESSES; i++) {
        if ((pid = fork()) < 0)
            return 1;
        if (pid == 0) {
            fread(&buf, 1, 1, f);
            while (buf != '\n') {
                num1 = (num1 * 10) + (buf - '0');
                fread(&buf, 1, 1, f);
            }
            fread(&buf, 1, 1, f);
            while (buf != '\n') {
                num2 = (num2 * 10) + (buf - '0');
                fread(&buf, 1, 1, f);
            }
            exit(num1 + num2);
        }
        else {
            waitpid(pid, &status, 0);
            result += (status >> 8);
        }
    }
    exit(result);
}
else {
    waitpid(p, &status, 0);
    result += (status >> 8);
}
clock_gettime(CLOCK_REALTIME, &stop);
printf("value of fork : %d\n", result);
printf("%lf\n", (double)((stop.tv_sec - start.tv_sec) + (

```

두번째 문제는 assignment2 에서 발생했다.

filegen 에서 시작됐다. for 문에 인덱스로 쓰인 i 값을 문자로 바꿔 파일 이름으로 쓰려고 시도하는 곳에서 문제가 있었다. 처음에는 i 값을 뒤에서부터 하나씩 10 으로 나뉘가면서 배열에 저장하는 방식을 사용했다.

```
tmp[a--] = i % 10
```

```
i /= 10
```

하지만 이 방식은 i 의 자릿수가 올라갈수록 a 의 값을 계속 바꿔주어야 하고 바뀐 값마다 NULL 을 넣어야 하는 위치를 바꿔주어야 하는 등 코드의 길이가 점점 복잡해지고 어디선가 실수를 해서 segmentation fault 가 발생했다. itoa 를 사용하는 방법도 있었지만 vscode 가 자체적으로 지원하지 때문에 vim 에서 컴파일 시 오류가 생겼다. 결국 sprintf 로 정수를 통째로 문자배열로 바꾸었고 strcat 을 사용해 디렉토리 이름에 붙여 경로와 파일 이름을 같이 정의해 주어서 문제를 해결할 수 있었다. 최종 코드는 다음과 같다.

```
#define MAX_PROCESS 30000

int main(void) {
    char num[100];
    mkdir("temp", 0777);
    DIR *dir = opendir("temp");
    for (int i=0; i<MAX_PROCESS; i++) {
        char file_name[10000] = {'t','e','m','p','/','\0'};
        sprintf(num, "%d", i);
        FILE *f_write = fopen(strcat(file_name, num), "w+");
        fprintf(f_write, "%d", 1+rand()%9);
        fclose(f_write);
    }
    closedir(dir);
}
```

세번째 문제는 assignment2 실험 도중 발생했다. 테스트를 할 때마다 실행시간이 극단적으로 변했다. 0.002 초가 나오고 어떤 때는 0.89 초가 나오고 너무 차이가 커서 모든 결과 값이 의미가 없어지는 상황이 생긴 것이다. 수시간의 시도 끝에 과제에서 예시로 든 10000 개의 프로세스가 적다는 것을 인지하고 유의미한 차이를 갖는 3 만 개까지 올려서 스케줄링 정책 간의 속도를 비교할 수 있었다.

네번째, 다섯번째 문제는 assignment3 구현 도중 발생했다. 이전 assignment2 에서는 현재 process 를 가져오기 위해 current 포인터를 사용했다. 하지만 이번에는 for\_each\_process 를 사용해 모든 프로세스 중 내가 지정한 프로세스만 검사해야 하기 때문에 현재 프로세스를

가져오는 것이 아니라 main 함수에서 지정한 프로세스만 가져와야 했다. 때문에 current 를 사용하지 말고 if 문을 사용해 main 함수에서 전달 한 process 정보를 비교하는 방식이 더 적합했다.

```
static asmlinkage pid_t process_tracer(const struct pt_regs *regs)
{
    struct task_struct *cur_task, *par_task;
    struct task_struct *sibling_task, *child_task;
    struct list_head *sibling_list, *child_list;

    for_each_process(cur_task) {
        if (regs->di == cur_task->pid) {
            // cur_task = current;
            par_task = cur_task->real_parent;
            cur_pid = regs->di;
            cur_process = cur_task->comm;
            sibling_count = 0;
            child_count = 0;
            printk(KERN_INFO "##### TASK INFORMATION OF '['%d] %s' #####\n", cur_pid, cur_process);

            switch (cur_task->state)
            {
            case 0:
                // ...
            }
        }
    }
}
```

보이는 것처럼 cur\_task 가 해당 함수를 동작시킨 프로세스를 가리키는 것이 아닌, 전달된 값과 동일한 프로세스를 가리켜야 한다. 위처럼 코드를 바꿔서 해결할 수 있었다.

처음에는 과제에서 제시된 조건대로 pid\_t trace\_task argument 를 hooking 함수로 사용했지만 값이 제대로 전달되지 않았다. 바로 위에서 설명한 것처럼 if 문을 통해서 값을 비교하려 했지만 계속 조건을 만족하지 못했고 trace\_task 값이 쓰레기 값으로 채워진 것을 확인할 수 있었다. Assignment2 에서 진행한 대로 레지스터로 값을 받아와 해결할 수 있었다.

```
static asmlinkage pid_t process_tracer(const struct pt_regs *regs)
{
    struct task_struct *cur_task, *par_task;
    struct task_struct *sibling_task, *child_task;
```

여섯번째 이자 마지막 문제는 실험 도중에 발생했는데 VMware 에 깔린 Ubuntu 가 한글을 제대로 인식하지 못해 Make 를 제대로 실행하지 못한것이다.

```
KW-Univ > 3학년 2학기 > 운영체제 > as2 > assignment3 > M Makefile
1  obj-m := process_tracer.o
2  KDIR := /lib/modules/$(shell uname -r)/build
3  PWD := $(shell pwd)
4
5  all:
6      $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) modules
7  clean:
8      $(MAKE) -C $(KDIR) SUBDIRS=$(PWD) clean
9
```

잠시 해당 파일들만 root 로 가져와 실습을 진행해서 해결할 수 있었다.

3-2 를 수행하면서 약간의 아쉬운 점이 있었다. Priority, nice 별 수행 시간 차이를 측정하지 못한 것이다. 프로세스의 개수를 10 만개 이상으로 늘리면 가능할지 모르나 5 만개부터 VMware 가 제대로 작동하지 않아 중간에 멈추게 되었다. 컴퓨팅 자원 전체를 활용할 수 있는 서버에서 실행하면 해당 실행시간 차이를 확인할 수 있을 것으로 예상된다.

- **Reference**

[https://linux.die.net/man/2/sched\\_setscheduler](https://linux.die.net/man/2/sched_setscheduler)

[https://blog.naver.com/alice\\_k106/221149061940](https://blog.naver.com/alice_k106/221149061940)