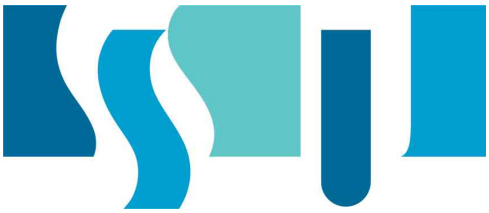


운영체제 과제 4

강의명	운영체제
교수명	김철홍 교수님
학 과	컴퓨터학부
학 번	20172655
이 름	이강산
제출일	2021.11.07.

< 목차 >

1. 구현 사항

1. CFS 동작 확인
   - nice value 조정에 따른 process 수행 순서의 차이를 확인하는 프로그램 작성
   - 구현 파일 :
      - 20172655-1.c : 21개 자식 프로세스 동일한 우선순위, 동일한 작업 수행
      - 20172655-2.c : 그룹 별 연산 횟수 및 nice value값 조정 후 작업 수행
   - nice value를 조정하여 우선순위를 높이기 위해선 root 권한이 필요. 따라서 모든 프로그램은 root 권한으로 수행하였다.

2. FIFO scheduler 동작 확인
   - 커널 수정을 통해 scheduling policy를 CFS에서 Realtime FIFO로 변경
   - 수정 파일 및 경로 : core.c / 해당커널폴더/kernel/sched/core.c 에 정의된 두 함수(shed_fork(), _shed_setscheduler()) 수정
   - 20172655-1.c로 동작을 확인하였다.

2. 구현 상세

2-1. CFS 동작 확인

- 20172655-1.c : 21개 자식 프로세스가 동일한 우선순위로 동일한 작업을 수행한다.

20172655-1.c

```
/* 20172655 LEE KANG SAN */
/* CASE 1 : CFS scheduler - No priority changed */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sched.h>
int main() {
        int sched = sched_getscheduler(getpid());
        switch (sched) {
                case 0: printf("scheduling policy : [%d] SCHED_OTHER\n", sched);
break;
                case 1: printf("scheduling policy : [%d] SCHED_FIFO\n", sched);
break;
                case 2: printf("scheduling policy : [%d] SCHED_RR\n", sched);
break;
```

```
	}
	printf("**************************************************\n");
	pid_t pid_s[21], pid_e[21];
	for(int i=0; i<21; i++) { //21 child processes, same task.
		pid_s[i]=fork();
		if(pid_s[i]>0)
			printf("%d process begins.\n", pid_s[i]);
		else if(pid_s[i]==0) { //task = calculate matrix
			double A[500][500], B[500][500], C[500][500];
			for(int i=0; i<500; i++) { //initialize matrix A, B
				for(int j=0; j<500; j++) {
					A[i][j]=(double)i+j;
					B[i][j]=1000.0-A[i][j];
				}
			}
			for(int i=0; i<500; i++) { //calculate 500*500*500 times
				for(int j=0; j<500; j++) {
					for(int k=0; k<500; k++) {
						C[i][j]+=A[i][k]*B[k][j];
					}
				}
			}
			printf("%d process ends.\n", getpid());
			exit(0);
		}
	}
	for(int i=0; i<21; i++)
		pid_e[i]=wait(NULL);
	printf("**************************************************\n");
	printf("BEGINNING order\n");
	for(int i=0; i<21; i++) {
		printf("[%d] ", pid_s[i]);
		if(i%7==6) printf("\n");}
	printf("ENDING order\n");
	for(int i=0; i<21; i++) {
		printf("[%d] ", pid_e[i]);
		if(i%7==6) printf("\n");}
	printf("**************************************************\n");
	printf("FINISHED\n");
	return 0;}
```

동작 결과 및 설명

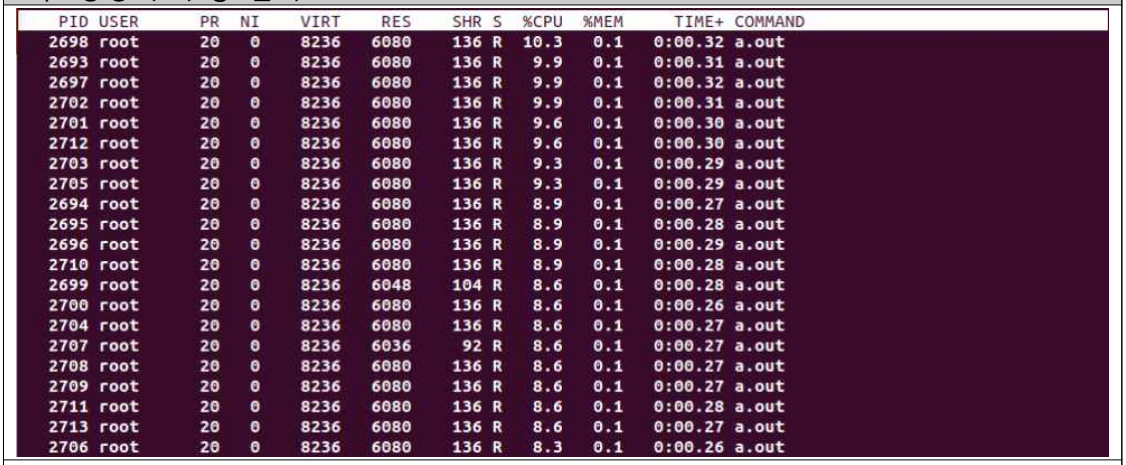
```
root@linux:/home/san/21-2/OS/os-4# ./a.out
scheduling policy : [0] SCHED_OTHER
************************************************
2693 process begins.
2694 process begins.
2695 process begins.
2696 process begins.
2697 process begins.
2698 process begins.
2699 process begins.
2700 process begins.
2701 process begins.
2702 process begins.
2703 process begins.
2704 process begins.
2705 process begins.
2706 process begins.
2707 process begins.
2708 process begins.
2709 process begins.
2710 process begins.
2711 process begins.
2712 process begins.
2713 process begins.
2703 process ends.
2693 process ends.
2699 process ends.
2710 process ends.
2698 process ends.
2708 process ends.
2697 process ends.
2711 process ends.
2705 process ends.
2713 process ends.
2706 process ends.
2702 process ends.
2701 process ends.
2712 process ends.
2709 process ends.
2695 process ends.
2696 process ends.
2704 process ends.
2707 process ends.
2700 process ends.
2694 process ends.
************************************************
BEGINNING order
[2693] [2694] [2695] [2696] [2697] [2698] [2699]
[2700] [2701] [2702] [2703] [2704] [2705] [2706]
[2707] [2708] [2709] [2710] [2711] [2712] [2713]
ENDING order
[2703] [2693] [2699] [2710] [2698] [2697] [2708]
[2711] [2705] [2713] [2706] [2702] [2701] [2712]
[2709] [2695] [2696] [2704] [2707] [2700] [2694]
************************************************
FINISHED
```

프로그램 시작 시 sched_getscheduler()를 호출하여 현재 커널의 scheduling policy가 SCHED_OTERS(CFS)임을 확인하였다.

PID 2693~2713이 순서대로 fork되어 각자의 작업을 수행한다. 주어진 작업은 한 번의 행렬곱 연산으로 500*500*500번의 실수 연산이 수행된다.

작업 완료 시 자식프로세스들은 자신의 PID를 출력하고 exit한다. 모든 자식프로세스가 동일한 양의 task와 동일한 우선순위를 가지므로 이들의 종료 순서는 프로그램 수행 시마다 달라진다.

top 명령어 수행 결과



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2698	root	20	0	8236	6080	136	R	10.3	0.1	0:00.32	a.out
2693	root	20	0	8236	6080	136	R	9.9	0.1	0:00.31	a.out
2697	root	20	0	8236	6080	136	R	9.9	0.1	0:00.32	a.out
2702	root	20	0	8236	6080	136	R	9.9	0.1	0:00.31	a.out
2701	root	20	0	8236	6080	136	R	9.6	0.1	0:00.30	a.out
2712	root	20	0	8236	6080	136	R	9.6	0.1	0:00.30	a.out
2703	root	20	0	8236	6080	136	R	9.3	0.1	0:00.29	a.out
2705	root	20	0	8236	6080	136	R	9.3	0.1	0:00.29	a.out
2694	root	20	0	8236	6080	136	R	8.9	0.1	0:00.27	a.out
2695	root	20	0	8236	6080	136	R	8.9	0.1	0:00.28	a.out
2696	root	20	0	8236	6080	136	R	8.9	0.1	0:00.29	a.out
2710	root	20	0	8236	6080	136	R	8.9	0.1	0:00.28	a.out
2699	root	20	0	8236	6048	104	R	8.6	0.1	0:00.28	a.out
2700	root	20	0	8236	6080	136	R	8.6	0.1	0:00.26	a.out
2704	root	20	0	8236	6080	136	R	8.6	0.1	0:00.27	a.out
2707	root	20	0	8236	6036	92	R	8.6	0.1	0:00.27	a.out
2708	root	20	0	8236	6080	136	R	8.6	0.1	0:00.27	a.out
2709	root	20	0	8236	6080	136	R	8.6	0.1	0:00.27	a.out
2711	root	20	0	8236	6080	136	R	8.6	0.1	0:00.28	a.out
2713	root	20	0	8236	6080	136	R	8.6	0.1	0:00.27	a.out
2706	root	20	0	8236	6080	136	R	8.3	0.1	0:00.26	a.out

프로그램 동작 시 top 명령어의 모습이다. 자식프로세스 생성 시 별도의 값을 설정하지 않았으므로 모든 자식 프로세스들이 동일한 NI값을 갖고 있다.

- 20172655-2.c : 21개 자식 프로세스를 세 그룹으로 나누어 각각 다른 task, 다른 priority를 부여한다.

20172655-2.c

```
/* 20172655 LEE KANG SAN */
/* CASE 2 : CFS scheduler - Priorty changed */
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <sys/resource.h>
#include <sched.h>
int main() {
        int sched = sched_getscheduler(getpid());
        switch (sched) {
                case 0: printf("scheduling policy : [%d] SCHED_OTHER\n", sched);
break;
                case 1: printf("scheduling policy : [%d] SCHED_FIFO\n", sched);
break;
                case 2: printf("scheduling policy : [%d] SCHED_RR\n", sched);
break;
        }
        printf("**************************************************\n");
        pid_t pid_s[21], pid_e[21];
        for(int i=0; i<7; i++) { //GROUP1 : little task, nice = 15
                pid_s[i]=fork();
                if(pid_s[i]>0)
                        printf("%d process bigins.\n", pid_s[i]);
                else if(pid_s[i]==0) { //task = calculate matrix 1 time
                        setpriority(PRIO_PROCESS, getpid(), 15);
                        double A[500][500], B[500][500], C[500][500];
                        for(int i=0; i<500; i++) { //initialize matrix A, B
                                for(int j=0; j<500; j++) {
                                        A[i][j]=(double)i+j;
                                        B[i][j]=1000.0-A[i][j];
                                }
                        }
                        for(int i=0; i<500; i++) { //calculate 500*500*500 times
                                for(int j=0; j<500; j++) {
                                        for(int k=0; k<500; k++) {
                                                C[i][j]+=A[i][k]*B[k][j];
```

```
                                        }
                                }
                        }
                        printf("%d process ends.\n", getpid());
                        exit(0);
                }
        }
        for(int i=7; i<14; i++) { //GROUP2, same task, nice = 0
                pid_s[i]=fork();
                if(pid_s[i]>0)
                        printf("%d process bigins.\n", pid_s[i]);
                else if(pid_s[i]==0) { //children task = calculate matrix 3 times
                        setpriority(PRIO_PROCESS, getpid(), 0);
                        double A[500][500], B[500][500], C[500][500];
                        for(int i=0; i<500; i++) { //initialize matrix A, B
                                for(int j=0; j<500; j++) {
                                        A[i][j]=(double)i+j;
                                        B[i][j]=1000.0-A[i][j];
                                }
                        }
                        for(int c=0; c<3; c++) {//calculate 3*500*500*500 times
                                for(int i=0; i<500; i++) {
                                        for(int j=0; j<500; j++) {
                                                for(int k=0; k<500; k++) {
                                                        C[i][j]+=A[i][k]*B[k][j];
                                                }
                                        }
                                }
                        }
                        printf("%d process ends.\n", getpid());
                        exit(0);
                }
        }
        for(int i=14; i<21; i++) { //GROUP3, more task, nice = -15
                pid_s[i]=fork();
                if(pid_s[i]>0)
                        printf("%d process bigins.\n", pid_s[i]);
                else if(pid_s[i]==0) { //task = calculate matrix 5 times
                        setpriority(PRIO_PROCESS, getpid(), -15);
                        double A[500][500], B[500][500], C[500][500];
```

```
                        for(int i=0; i<500; i++) { //initialize matrix A, B
                                for(int j=0; j<500; j++) {
                                        A[i][j]=(double)i+j;
                                        B[i][j]=1000.0-A[i][j];
                                }
                        }
                        for(int c=0; c<5; c++) {//calculate 5*500*500*500 times
                                for(int i=0; i<500; i++) {
                                        for(int j=0; j<500; j++) {
                                                for(int k=0; k<500; k++) {
                                                        C[i][j]+=A[i][k]*B[k][j];
                                                }
                                        }
                                }
                        }
                        printf("%d process ends.\n", getpid());
                        exit(0);
                }
        }

        for(int i=0; i<21; i++)
                pid_e[i]=wait(NULL);
        printf("**************************************************\n");
        printf("BEGINNING order\n");
        for(int i=0; i<21; i++) {
                printf("[%d] ", pid_s[i]);
                if(i%7==6) printf("\n");
        }
        printf("ENDING order\n");
        for(int i=0; i<21; i++) {
                printf("[%d] ", pid_e[i]);
                if(i%7==6) printf("\n");
        }
        printf("**************************************************\n");
        printf("FINISHED\n");
        return 0;
}
```

동작 결과 및 설명

```
root@linux:/home/san/21-2/OS/os-4# ./a.out
scheduling policy : [0] SCHED_OTHER
************************************************
2781 process bigins.
2782 process bigins.
2783 process bigins.
2784 process bigins.
2785 process bigins.
2786 process bigins.
2787 process bigins.
2788 process bigins.
2789 process bigins.
2790 process bigins.
2791 process bigins.
2792 process bigins.
2793 process bigins.
2794 process bigins.
2795 process bigins.
2796 process bigins.
2797 process bigins.
2798 process bigins.
2799 process bigins.
2800 process bigins.
2801 process bigins.
2799 process ends.
2797 process ends.
2800 process ends.
2795 process ends.
2798 process ends.
2801 process ends.
2796 process ends.
2793 process ends.
2794 process ends.
2788 process ends.
2791 process ends.
2792 process ends.
2789 process ends.
2790 process ends.
2785 process ends.
2782 process ends.
2784 process ends.
2783 process ends.
2787 process ends.
2786 process ends.
2781 process ends.
************************************************
BEGINNING order
[2781] [2782] [2783] [2784] [2785] [2786] [2787]
[2788] [2789] [2790] [2791] [2792] [2793] [2794]
[2795] [2796] [2797] [2798] [2799] [2800] [2801]
ENDING order
[2799] [2797] [2800] [2795] [2798] [2801] [2796]
[2793] [2794] [2788] [2791] [2792] [2789] [2790]
[2785] [2782] [2784] [2783] [2787] [2786] [2781]
************************************************
FINISHED
```

프로그램 시작 시 sched_getscheduler()를 호출하여 현재 커널의 scheduling policy가 SCHED_OTERS(CFS)임을 확인하였다.

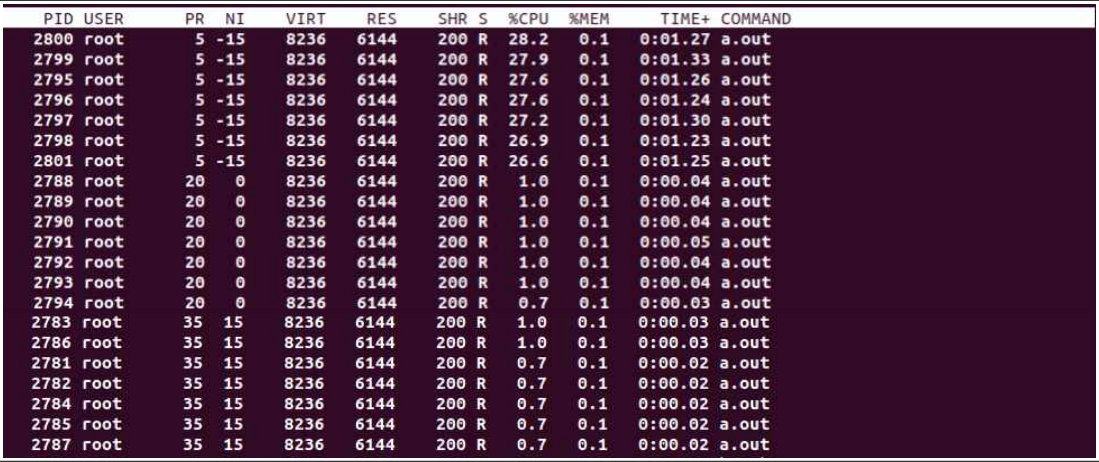
PID 2781~2801이 순서대로 fork되어 각자의 작업을 수행한다. 주어진 작업은 행렬곱 연산으로 한 번의 행렬곱 연산은 500*500*500번의 실수 연산이 수행된다.

자식프로세스들은 세 개의 그룹으로 나뉜다. 그룹1(PID 2781~2787)은 한 번의 행렬곱 연산과 낮은 우선순위(15)를, 그룹2(PID 2788~2794)는 세 번의 행렬곱 연산과 기본 우선순위(0)를, 그룹3(PID 2795~2801)은 다섯 번의 행렬곱 연산과 높은 우선순위(-15)를 갖도록 조정되었다.

작업 완료 시 자식프로세스들은 자신의 PID를 출력하고 exit한다. 그룹1이 그룹3보다 먼저 fork되고 작업량도 적으나, 우선순위가 낮으므로 가장 마지막에 종료되었다.

결과의 ENDING order를 확인 하면, 우선순위가 높은 순인 그룹3 -> 그룹2 -> 그룹1 순으로 자식프로세스가 종료되었음을 알 수 있다.

top 명령어 수행 결과



PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
2800	root	5	-15	8236	6144	200	R	28.2	0.1	0:01.27	a.out
2799	root	5	-15	8236	6144	200	R	27.9	0.1	0:01.33	a.out
2795	root	5	-15	8236	6144	200	R	27.6	0.1	0:01.26	a.out
2796	root	5	-15	8236	6144	200	R	27.6	0.1	0:01.24	a.out
2797	root	5	-15	8236	6144	200	R	27.2	0.1	0:01.30	a.out
2798	root	5	-15	8236	6144	200	R	26.9	0.1	0:01.23	a.out
2801	root	5	-15	8236	6144	200	R	26.6	0.1	0:01.25	a.out
2788	root	20	0	8236	6144	200	R	1.0	0.1	0:00.04	a.out
2789	root	20	0	8236	6144	200	R	1.0	0.1	0:00.04	a.out
2790	root	20	0	8236	6144	200	R	1.0	0.1	0:00.04	a.out
2791	root	20	0	8236	6144	200	R	1.0	0.1	0:00.05	a.out
2792	root	20	0	8236	6144	200	R	1.0	0.1	0:00.04	a.out
2793	root	20	0	8236	6144	200	R	1.0	0.1	0:00.04	a.out
2794	root	20	0	8236	6144	200	R	0.7	0.1	0:00.03	a.out
2783	root	35	15	8236	6144	200	R	1.0	0.1	0:00.03	a.out
2786	root	35	15	8236	6144	200	R	1.0	0.1	0:00.03	a.out
2781	root	35	15	8236	6144	200	R	0.7	0.1	0:00.02	a.out
2782	root	35	15	8236	6144	200	R	0.7	0.1	0:00.02	a.out
2784	root	35	15	8236	6144	200	R	0.7	0.1	0:00.02	a.out
2785	root	35	15	8236	6144	200	R	0.7	0.1	0:00.02	a.out
2787	root	35	15	8236	6144	200	R	0.7	0.1	0:00.02	a.out

프로그램 동작 시 top 명령어의 모습이다. 자식프로세스들이 setpriority()함수를 통해 그룹별로 동일한 NI값을 갖도록 조정되었음을 보여준다.

2-2. FIFO scheduler 동작 확인

/kernel/sched/core.c 수정 부분

1. shed_fork() 함수 수정 (표시된 부분 추가)

```
3724        //20172655-sched_fork
3725        if (p->policy == SCHED_NORMAL) {
3726                p->prio = current->normal_prio - NICE_WIDTH -
3727                        PRIO_TO_NICE(current->static_prio);
3728                p->normal_prio = p->prio;
3729                p->rt_priority = p->prio;
3730                p->policy = SCHED_FIFO; //FIFO
3731                p->static_prio = NICE_TO_PRIO(0);
3732        }
3733        //20172655-sched_fork
3734        uclamp_fork(p);
```

2. _shed_setscheduler() 함수 수정 (표시된 부분 추가)

```
6052        }
6053        //20172655-_sched_setscheduler
6054        if (attr.sched_policy == SCHED_NORMAL) {
6055                attr.sched_priority = param->sched_priority -
6056                        NICE_WIDTH - attr.sched_nice;
6057                attr.sched_policy = SCHED_FIFO; //FIFO
6058        }
6059        //20172655-_sched_setscheduler
6060        return   sched_setscheduler(p, &attr, check, true);
```

ps -c 명령어 수행 결과

scheduler 변경 전(TS)

```
root@linux:/home/san# ps -c
    PID CLS PRI TTY          TIME CMD
   1821 TS   19 pts/0    00:00:00 su
   1823 TS   19 pts/0    00:00:00 bash
   1834 TS   19 pts/0    00:00:00 ps
root@linux:/home/san#
```

scheduler 변경 후(FF)

```
root@linux:/home/san/21-2/OS/os-4# ps -c
    PID CLS PRI TTY          TIME CMD
   1813 FF   59 pts/1    00:00:00 su
   1814 FF   59 pts/1    00:00:00 bash
   2380 FF   59 pts/1    00:00:00 top
   2441 FF   59 pts/1    00:00:00 top
   2673 FF   59 pts/1    00:00:00 ps
```

ps명령어의 -c 옵션은 동작 중인 프로세스들의 CLS를 표시한다. CLS는 TS, RR, FF 중 하나로 표시되며 TS는 SCHED_OTHER(CFS), RR은 SCHED_RR(round robin), FF는 SCHED_FIFO(FIFO)를 의미한다.

위는 /kernel/sched/core.c 내의 두 함수 shed_fork()와 _shed_setscheduler()의 수정 전후로 ps -c 명령어를 수행한 결과이다. 동작 중인 프로세스들이 FIFO scheduler에 의해 scheduling 중임을 알 수 있다.

20172655-1.c 수행 결과

```
root@linux:/home/san/21-2/OS/os-4# ./a.out
scheduling policy : [1] SCHED_FIFO
************************************************
2719 process begins.
2720 process begins.
2721 process begins.
2722 process begins.
2723 process begins.
2724 process begins.
2725 process begins.
2726 process begins.
2727 process begins.
2728 process begins.
2729 process begins.
2730 process begins.
2731 process begins.
2732 process begins.
2733 process begins.
2734 process begins.
2735 process begins.
2736 process begins.
2737 process begins.
2738 process begins.
2739 process begins.
2720 process ends.
2719 process ends.
2721 process ends.
2722 process ends.
2723 process ends.
2724 process ends.
2725 process ends.
2726 process ends.
2727 process ends.
2728 process ends.
2729 process ends.
2730 process ends.
2731 process ends.
2732 process ends.
2733 process ends.
2734 process ends.
2735 process ends.
2736 process ends.
2737 process ends.
2738 process ends.
2739 process ends.
************************************************
BEGINNING order
[2719] [2720] [2721] [2722] [2723] [2724] [2725]
[2726] [2727] [2728] [2729] [2730] [2731] [2732]
[2733] [2734] [2735] [2736] [2737] [2738] [2739]
ENDING order
[2719] [2720] [2721] [2722] [2723] [2724] [2725]
[2726] [2727] [2728] [2729] [2730] [2731] [2732]
[2733] [2734] [2735] [2736] [2737] [2738] [2739]
************************************************
FINISHED
```

프로그램 시작 시 sched_getscheduler()를 호출하여 현재 커널의 scheduling policy가 이전과 달리 SCHED_FIFO임을 확인하였다.

PID 2719~2739가 순서대로 fork되어 각자의 작업을 수행한다. 주어진 작업은 한 번의 행렬곱 연산으로 500*500*500번의 실수 연산이 수행된다.

CFS scheduler와 달리 FIFO scheduler는 먼저 fork된 자식프로세스가 먼저 종료됨을 알 수 있다. 가끔 종료 순서가 실행 순서와 약간씩 다른 경우가 있는데, 이는 Linux에 구현된 FIFO가 preemptive realtime FIFO라 우선순위도 고려되기 때문이다.

3. 과제를 통해 배운 점

리눅스 커널은 기본적으로 CFS를 사용하지만, 그 외에도 FIFO, Round Robin 등 여러 스케쥴러들이 /kernel/sched/ 디렉토리 내에 이미 구현되어 있다.

자식프로세스의 scheduling policy는 별도의 언급이 없으면 부모의 scheduling policy와 동일하다.

CFS(Completely Fair Scheduler)는 O(logN) 성능을 가지며, nice value로 계산되는 우선순위를 기준으로 대기중인 프로세들에게 cpu 자원을 할당한다.

CFS의 동작을 확인하는 프로그램에서 NI 값의 조정을 통해, 먼저 fork되고, task도 더 적은 프로세스보다, 후순위로 fork되고 task도 많지만 우선순위가 더 높은 프로세스들이 먼저 종료됨을 확인하였다.

FIFO의 동작을 확인하는 프로그램에서, 동일한 우선순위와 task를 갖는 자식프로세스들이 fork된 순서대로 종료됨을 확인하였다.