운영체제 실습

[Assignment#3]

Class : 목34

Professor : 최상호 교수님

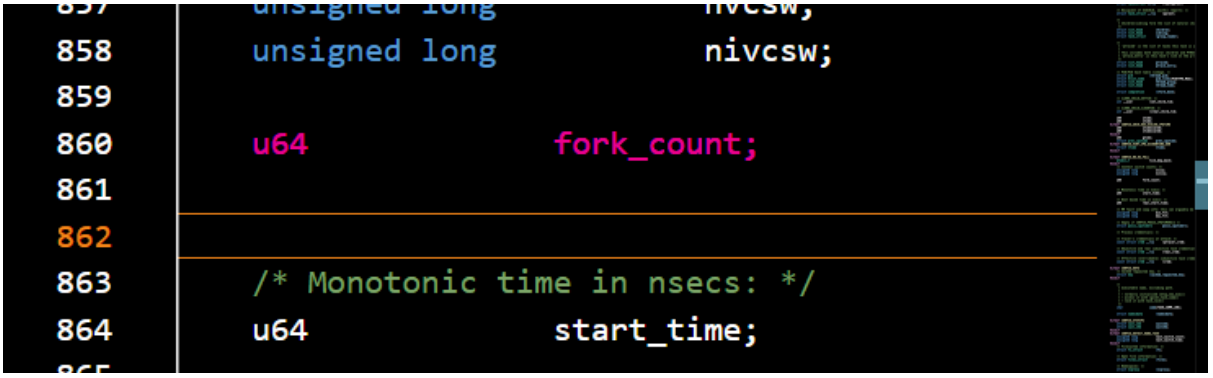
Student ID : 2021202003

Name : 강준우

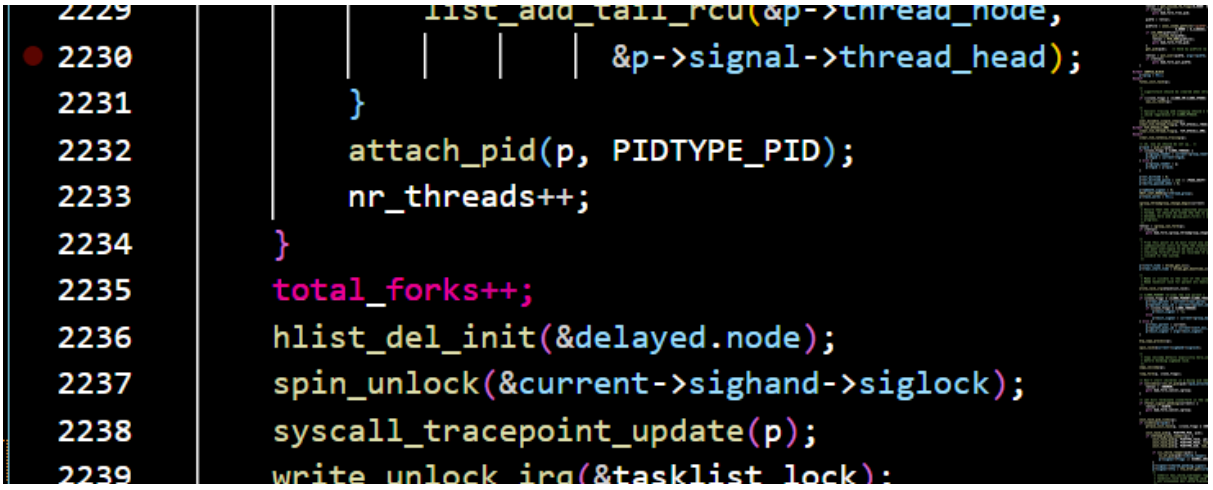
Introduction

3주차는 운영체제의 커널의 단계적 수행 처리와 fork를 기반으로한 병렬 처리의 수행 결과분석 그리고 스케줄링의 결과를 나타낸다. 3-1과제에서는 fork() 호출에 대한 카운트를 기반으로 프로세스 상태에 대해 나타낸다. 3-2에서는 프로세스와 스레드를 다중으로 수행하며 결과와 트레이드오프를 분석한다 3-3번째 과제에서는 FCFS, SJF, SRTF, RR 네 가지 스케줄링 알고리즘을 수행 결과를 기반으로 분석하며 장단점에 대해서 분석한다.

결과화면



/usr/src/linux-5.4.282/include/linux/sched.h 에서 fork count를 정의

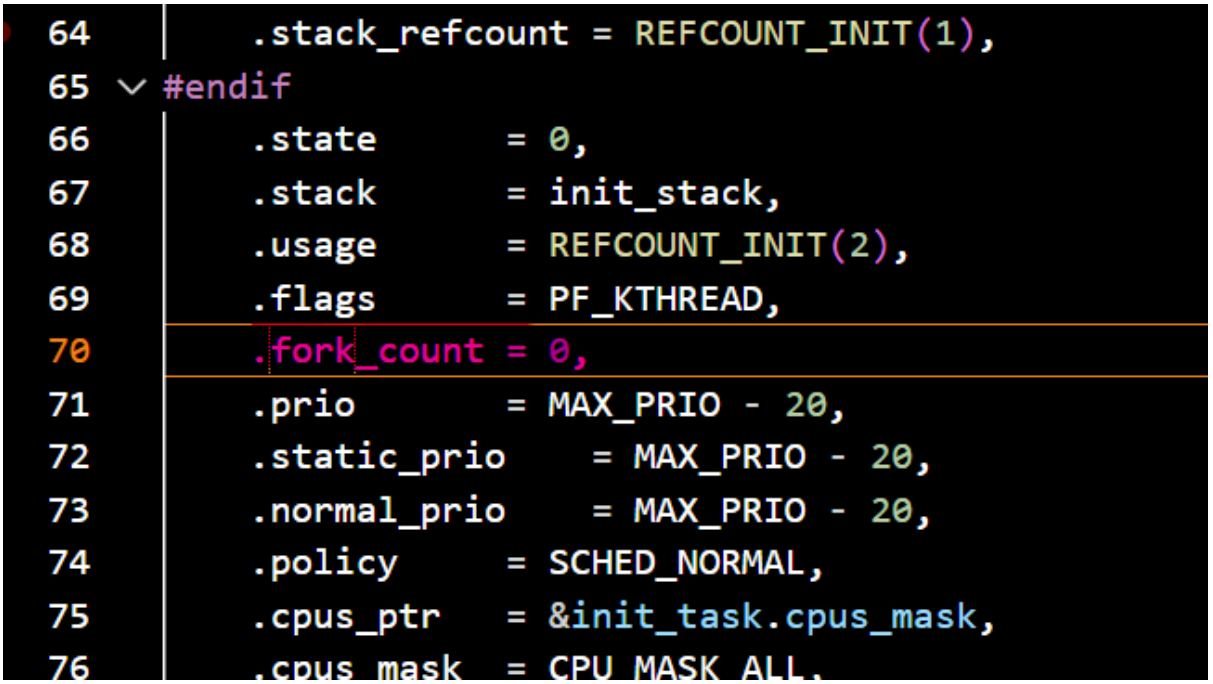


```
1955    task_io_accounting_init(&p->ioac);
1956    acct_clear_integrals(p);
1957
1958    posix_cputimers_init(&p->posix_cputimers);
1959    p->fork_count = 0;
1960    p->io_context = NULL;
1961    audit_set_context(p, NULL);
1962    cgroup_fork(p);
1963 #ifdef CONFIG_NUMA
```

/usr/src/linux-5.4.282/kernel/fork.c에서 해당 새 프로세스는 0부터 시작하게 구성

```
2382    add_latent_entropy();
2383    Click to add a breakpoint
2384    if (IS_ERR(p))
2385        return PTR_ERR(p);
2386
2387    current->fork_count++;
2388
2389    /*
2390     * Do this prior waking up the new thread - t
2391     * might get invalid after that point, if the
2392     */
```

/usr/src/linux-5.4.282/kernel/fork.c에서 부모는 1추가되게 구현



/usr/src/linux-5.4.282/init/init_task.c 에서 fork_count를 0으로 초기화 선언함

Assignment 3-1

```
root@ubuntu:~/Assignment3/3-1_process_tracer# dmesg
[ 2702.944251] [OSLab.] ##### TASK INFORMATION of '[1] systemd' #####
[ 2702.944295] [OSLab.] - task state : Wait
[ 2702.944337] [OSLab.] - Process Group Leader : [1] systemd
[ 2702.944377] [OSLab.] - # of context-switch(es) : 6358
[ 2702.944417] [OSLab.] - Number of calling fork() : 193
[ 2702.944457] [OSLab.] - its parent process : [0] swapper/0
[ 2702.944496] [OSLab.] - its sibling process(es):
[ 2702.944536] [OSLab.]   > [2] kthreadd
[ 2702.944575] [OSLab.]   > This process has 1 sibling process(es)
[ 2702.944614] [OSLab.] - its child process(es):
[ 2702.944654] [OSLab.]   > [332] systemd-journal
[ 2702.944705] [OSLab.]   > [371] systemd-udevd
[ 2702.944747] [OSLab.]   > [383] vmware-vmblock-
[ 2702.944788] [OSLab.]   > [730] systemd-resolve
[ 2702.944828] [OSLab.]   > [732] systemd-timesyn
[ 2702.944869] [OSLab.]   > [742] VGAuthService
[ 2702.944909] [OSLab.]   > [744] vmtoolsd
[ 2702.944950] [OSLab.]   > [811] accounts-daemon
[ 2702.945004] [OSLab.]   > [812] acpid
[ 2702.945369] [OSLab.]   > [815] avahi-daemon
[ 2702.945410] [OSLab.]   > [816] bluetoothd
[ 2702.945450] [OSLab.]   > [819] cron
[ 2702.945490] [OSLab.]   > [821] cupsd
[ 2702.945529] [OSLab.]   > [822] dbus-daemon
[ 2702.945570] [OSLab.]   > [824] NetworkManager
[ 2702.945609] [OSLab.]   > [832] irqbalance
[ 2702.945649] [OSLab.]   > [833] networkd-dispat
[ 2702.945699] [OSLab.]   > [836] polkitd
[ 2702.945738] [OSLab.]   > [842] rsyslogd
[ 2702.945779] [OSLab.]   > [859] snapd
[ 2702.945818] [OSLab.]   > [868] switcheroo-cont
[ 2702.945863] [OSLab.]   > [872] systemd-logind
[ 2702.945904] [OSLab.]   > [874] udisksd
[ 2702.945943] [OSLab.]   > [875] wpa_supplicant
[ 2702.945987] [OSLab.]   > [907] ModemManager
[ 2702.946028] [OSLab.]   > [910] cups-browsed
[ 2702.946095] [OSLab.]   > [941] unattended-upgr
[ 2702.946136] [OSLab.]   > [954] sshd
[ 2702.946177] [OSLab.]   > [966] gdm3
[ 2702.946217] [OSLab.]   > [991] whoopsie
[ 2702.946257] [OSLab.]   > [994] kerneloops
[ 2702.946297] [OSLab.]   > [1002] kerneloops
[ 2702.946337] [OSLab.]   > [1029] rtkit-daemon
[ 2702.946377] [OSLab.]   > [1148] upowerd
[ 2702.946418] [OSLab.]   > [1463] colord
[ 2702.946458] [OSLab.]   > [1525] systemd
[ 2702.946498] [OSLab.]   > [1539] gnome-keyring-d
[ 2702.946537] [OSLab.]   > [2089] systemd
[ 2702.946576] [OSLab.] ##### END OF INFORMATION #####
[ 2702.946617] [OSLab.]   > This process has 38 child process(es)
```

fork() 호출 횟수 카운터를 추가하고, 이를 사용자 지정 시스템 콜을 통해 조회하여 프로세스 정보를 출력하는 것을 목표로 하였다. 커널 패치 적용 후, dmesg 로그를 통해 systemd를 대상으로 추적을 실행하여 구현의 성공을 검증하였다.

핵심 요구사항인 fork() 호출 횟수 항목에 193 이라는 유효한 숫자가 정확히 출력되었다. 이 결과는 struct task_struct 필드 추가, kernel/fork.c에서의 초기화 및 증가 로직, 그리고 init/init_task.c에서의 최초 초기화 단계들을 확인할 수 있다.

또한, systemd의 Context Switch 횟수는 6358로 정확히 계수되었으며, task state는 Wait로 올바르게 출력되었다. 모듈은 프로세스 계층 구조 분석 요구사항에 따라 부모([0] swapper/0) 정보와 함께 38개의 자식 프로세스 목록 및 총 개수를 완벽하게 출력하였다. 최종적으로 모든 출력 항목이 [OSLab.] 접두사를 포함하는 등 과제에서 요구하는 출력 형식을 충족하며 기능 구현을 완료하였다.

Assignment 3-2

Assignment 3-2는 다중 프로세스와 다중 스레드 환경에서 트리 기반의 알고리즘을 통해서 합산 작업을 실행한다. 이후 성능과 통신 제약을 비교 분석한다.

```
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# rm -rf tmp*
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# sync
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# echo 3 | sudo tee /proc/sys/vm/drop_caches
3
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# gcc -O2 -Wall -Wextra numgen.c -o numgen
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# ./numgen
Generated 16 numbers into ./temp.txt
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# gcc -O2 -Wall -Wextra fork.c -o fork_ver
fork.c: In function ‘main’:
fork.c:61:11: warning: variable ‘pids’ set but not used [-Wunused-but-set-variable]
   61 |     pid_t pids[MAX_PROCESSES];
      |           ^~~~
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# ./fork_ver
value of fork : 136
0.002449
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# gcc -O2 -Wall -Wextra -pthread thread.c -o thread_ver
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# ./thread_ver
value of thread : 136
0.001437
```

```
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# gcc -O2 -Wall -Wextra numgen.c -o numgen
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# ./numgen
Generated 128 numbers into ./temp.txt
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# gcc -O2 -Wall -Wextra fork.c -o fork_ver
fork.c: In function ‘main’:
fork.c:61:11: warning: variable ‘pids’ set but not used [-Wunused-but-set-variable]
   61 |     pid_t pids[MAX_PROCESSES];
      |           ^~~~
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# ./fork_ver
value of fork : 8256
0.014381
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# gcc -O2 -Wall -Wextra -pthread thread.c -o thread_ver
root@ubuntu:~/Assignment3/3-2_proc_thread_sum# ./thread_ver
value of thread : 8256
0.017539
```

.

MAX_PROCESSES=8에서 스레드 기반 구현이 41% 더 빠르게 나타났으나,
MAX_PROCESSES=64에서는 반대로 fork 기반 구현이 18% 더 빠르게 나타났다

M=8: 스레드는 주소공간 공유라 생성·자원준비 비용이 작고 캐시 친화적이라 스레드 쪽이 유리하다.

M=64: 작업이 너무 미세해서, 64개의 스레드가 동시에 스케줄될 때 스케줄링/컨텍스트 스위치, join 대기, 런큐 경쟁 등의 스레드 관리 오버헤드가 상대적으로 커진다.

반면 이번 시나리오에서의 fork()는자식이 즉시 종료 하므로 비용을 낮게 유지될 수 있다. 그 결과 프로세스 쪽이 더 빠르게 나온 것으로 해석할 수 있다. 연산량이 너무 작으며며 동시성이 커질수록 스레드 관리 비용이 상대적으로 커져 역전될 수 있다.

Assignment 3-3

본 과제는 커널 패치, 병렬 컴퓨팅 비교, CPU 스케줄링 시뮬레이션의 세 가지 영역을 수행하며 운영체제의 핵심 메커니즘을 분석하였다.
Input은 강의 자료의 자료를 사용하였다.

1 0 10

2 0 9

3 3 5

4 7 4

5 10 6

6 10 7

```
root@ubuntu:~/Assignment3/3-3_cpu_scheduler# ./cpu_scheduler input.1 FCFS
Gantt Chart:
| P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P2 | P2 | P2 | P2 | P2
| P2 | P2 | P2 | P2 | P3 | P3 | P3 | P3 | P3 | P4 | P4 | P4 | P4 | P5 | P5
| P5 | P5 | P5 | P5 | P6 | P6 | P6 | P6 | P6 | P6 |
Average Waiting Time = 14.17
Average Turnaround Time = 21.00
Average Response Time = 14.17
CPU Utilization = 98.56%
root@ubuntu:~/Assignment3/3-3_cpu_scheduler# ./cpu_scheduler input.1 SJF
Gantt Chart:
| P2 | P2 | P2 | P2 | P2 | P2 | P2 | P2 | P2 | P4 | P4 | P4 | P4 | P3 | P3
| P3 | P3 | P3 | P5 | P5 | P5 | P5 | P5 | P5 | P6 | P6 | P6 | P6 | P6 | P6
| P6 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 |
Average Waiting Time = 10.83
Average Turnaround Time = 17.67
Average Response Time = 10.83
CPU Utilization = 98.56%
root@ubuntu:~/Assignment3/3-3_cpu_scheduler# ./cpu_scheduler input.1 RR 2
Gantt Chart:
| P1 | P1 | P2 | P2 | P1 | P1 | P1 | P1 | P2 | P2 | P3 | P3 | P2 | P2 | P1
| P1 | P1 | P1 | P4 | P4 | P2 | P2 | P5 | P5 | P6 | P6 | P2 | P3 | P3 | P3
|
Average Waiting Time = 1.00
Average Turnaround Time = 7.83
Average Response Time = 7.67
CPU Utilization = 94.04%
root@ubuntu:~/Assignment3/3-3_cpu_scheduler# ./cpu_scheduler input.1 SRTF
Gantt Chart:
| P2 | P2 | P2 | P3 | P3 | P3 | P3 | P3 | P4 | P4 | P4 | P4 | P2 | P2 | P2
| P2 | P2 | P2 | P5 | P5 | P5 | P5 | P5 | P5 | P6 | P6 | P6 | P6 | P6 | P6
| P6 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 | P1 |
Average Waiting Time = 10.50
Average Turnaround Time = 17.33
Average Response Time = 9.00
CPU Utilization = 98.32%
```

스케줄링 알고리즘의 성능을 0.1ms 컨텍스트 스위치 오버헤드를 반영하여 비교 분석하였다. SJF는 FCFS의 컨보이 효과를 개선하여 비선점 방식 중 우수한 평균 지표를 기록했으며, 두 비선점형 알고리즘 모두 98.56%의 높은 CPU 활용률을 유지했다. SRTF는 남은 시간이 짧은 작업을 우선하여 평균 반환 시간을 17.33ms로 평균 지표 최적화면에서 강점을 보였다. 반면, RR2은 공정한 할당을 통해 평균 응답 시간 7.67ms, 평균 대기 시간 1.00ms로 최저 수준의 응답성을 달성했으나, 스위치 오버헤드의 영향으로 CPU 활용률이 94.04%로 나오며 문제점을 보였다. SRTF와 RR 사이에 트레이드오프를 확인할 수 있었다.

고찰

3-1 과제를 수행하기 전에 fork 카운터에 대해서 운영체제에서 내장하여 호출할때 활용하는 기능을 기반으로 이전에매번 구현해야하는 과정들을 내장시켜 도이상 반복해서 구현하지 않아도 되는점이 좋아보였습니다. 3-3 과제를 수행하면서 나의 계산이 아닌 결과 분석 툴을 구현함으로 써 시각적인 비교가 가능하여 좋았습니다.

Reference

실습자료 이용했습니다. 감사합니다.