Operating Systems

Assignment #3

담당교수 : 김태석

강의 시간 : 수2

학부 : 컴퓨터정보공학부

학번 : 2017202088

이름 : 신해담

1. Introduction

fork와 thread를 사용하여 동일한 작업을 수행하는 프로그램을 작성하고 수행시간 및 결과값을 비교한다. 프로세스의 스케줄링 정책과 우선순위에 따른 수행시간을 비교하고 분석한다. pid를 입력받아서 해당 pid의 task_struct에 접근하여 각종 프로세스 정보들을 출력하는 모듈을 작성한다.

2. 실행결과 및 분석

3-1. fork, thread

- MAX_PROCESSES 8에서의 실행결과는 다음과 같다.

```
os2017202088@ubuntu:~/working/Assignment3/3-1$ ./fork
value of fork : 136
0.011002
os2017202088@ubuntu:~/working/Assignment3/3-1$ ./thread
value of fork : 136
0.009252
```

- MAX_PROCESSES 64에서의 실행결과는 다음과 같다.

```
os2017202088@ubuntu:~/working/Assignment3/3-1$ ./fork
value of fork : 64
0.019739
os2017202088@ubuntu:~/working/Assignment3/3-1$ ./thread
value of thread : 8256
0.017038
```

- 두 코드를 구현하면서 thread는 별도 설정 없이 read가 잘 작동하지만, fork는 file을 read하지 못하고 0을 읽는 경우가 발생했다. 두 프로세스가 같은 FILE stream에 접근하면서 문제가 발생한 것으로 추정되며, 이를 해결하기 위해 fork는 process마다 file을 open해서 fseek 함수로 위치를 찾도록 구현했다.

- 수행시간의 경우 fork가 fopen 및 fseek를 더 수행하므로 그만큼 더 느린 것으로 추정된다.

- MAX_PROCESSES 64에서 두 결과가 다른데, fork로 생성한 자식 프로세스의 반환값에 한계가 있기 때문이다. Thread는 정확한 결과를 출력한다.

- 자식 프로세스의 반환값 한계
  자식 프로세스의 반환값 크기는 int값을 가진다. int 자료형은 과거 2byte였으며, 그 시절에 구현한 시스템이므로 자식 프로세스는 2byte의 크기를 반환하게 된다. 그 중 1byte는 종료 상태를 가지므로, 나머지 1byte인 2^8 이상의 값을 전달할

수 없다.

- Exit status
  프로세스가 종료되어 반환하는 값의 하위 8bits는 프로세스 종료 상태를 나타낸다. 이 종료상태는 0~255(0000 0000 ~ 1111 1111) 사이의 값이다. 이 종료상태가 0이면 성공적으로 명령을 수행했음을 의미하며, 어떤 사유로 수행에 실패하면 그 사유를 8bits에 실어서 전달하게 된다.

3-2. Scheduling policy

- Standard round-robin time scheduling policy 을 사용하는 경우
  highest nice 에서의 실행결과는 다음과 같다.

```
os2017202088@ubuntu:~/working/Assignment3/3-2$ ./schedtest
value of fork : 50166
0.102017
```

  default nice 에서의 실행결과는 다음과 같다.

```
os2017202088@ubuntu:~/working/Assignment3/3-2$ ./schedtest
value of fork : 50166
0.105176
```

  lowest nice 에서의 실행결과는 다음과 같다.

```
os2017202088@ubuntu:~/working/Assignment3/3-2$ ./schedtest
value of fork : 50166
0.246123
```

- First-in, first-out policy 를 사용하는 경우
  highest priority 에서의 실행결과는 다음과 같다.

```
os2017202088@ubuntu:~/working/Assignment3/3-2$ ./schedtest
value of fork : 50166
0.073775
```

  middle priority 에서의 실행결과는 다음과 같다.

```
os2017202088@ubuntu:~/working/Assignment3/3-2$ ./schedtest
value of fork : 50166
0.106794
```

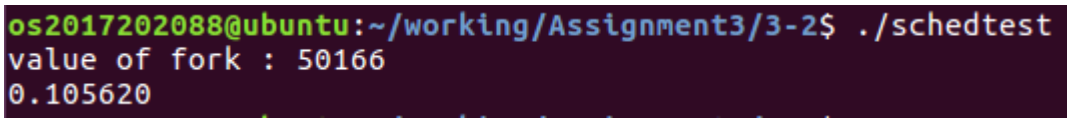
  lowest priority 에서의 실행결과는 다음과 같다.

```
os2017202088@ubuntu:~/working/Assignment3/3-2$ ./schedtest
value of fork : 50166
0.164136
```

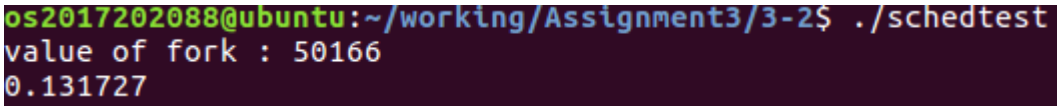
- Round-robin policy 를 사용하는 경우
  highest priority 에서의 실행결과는 다음과 같다.

```
os2017202088@ubuntu:~/working/Assignment3/3-2$ ./schedtest
value of fork : 50166
0.096201
```

middle priority 에서의 실행결과는 다음과 같다



lowest priority 에서의 실행결과는 다음과 같다.



- 리눅스의 기본 스케줄링 정책은 SCHED_OTHER 이며, 운영체제에 의해 지속적으로 변경되는 nice 우선순위를 가진다.
  SCHED_FIFO 는 긴급한 실시간 프로세스에 사용되며, 모든 SCHED_OTHER 그룹보다 높은 우선순위를 지닌다. 선입선출이므로 time slice 가 없다.
  SCHED_RR 은 모든 SCHED_OTHER 그룹보다 높은 우선순위를 가진다. time slice 가 존재해서 응답시간이 짧다.

- 실행결과를 통해 모든 정책에서 highest, default, lowest 순으로 시간이 오래 걸림을 확인할 수 있다. 또한 SCHED_OTHER 정책보다 SCHED_FIFO, SCHED_RR 이 프로세스를 더 빠르게 수행한다는 것을 알 수 있다. FIFO 는 우선순위가 높은 프로세스부터 처리하는 경향이 있고, RR 은 time slice 를 분배해서 어느정도 골고루 수행하므로 FIFO 의 결과가 RR 보다 우선순위에 따른 편차가 크게 나왔다고 볼 수 있다.

3-3. task_struct

- 1번 프로세스의 정보를 출력한 결과화면이다.

```
[14908.617280] ##### TASK INFORMATION of ''[1] systemd'' #####
[14908.617283] - task state : Wait
[14908.617285] - Process Group Leader : [1] systemd
[14908.617285] - Number of context switched : 2827
[14908.617286] - Number of calling fork() :
[14908.617286] - it's parrent process : [0] swapper/0
[14908.617287] - it's sibling process :
[14908.617288]   > [2] kthreadd
[14908.617288]   > This process has 1 sibling process(es)
[14908.617289] - it's child process(es) :
[14908.617290]   > [357] systemd-journal
[14908.617290]   > [384] vmware-vmblock-
[14908.617291]   > [391] systemd-udevd
[14908.617292]   > [409] vmtoolsd
[14908.617293]   > [456] systemd-timesyn
[14908.617294]   > [866] avahi-daemon
[14908.617295]   > [868] cron
[14908.617296]   > [873] acpid
[14908.617296]   > [874] systemd-logind
[14908.617297]   > [876] accounts-daemon
[14908.617298]   > [877] dbus-daemon
[14908.617299]   > [892] NetworkManager
[14908.617299]   > [895] rsyslogd
[14908.617300]   > [896] VGAuthService
[14908.617301]   > [901] cupsd
[14908.617302]   > [902] cups-browsed
[14908.617303]   > [952] irqbalance
[14908.617303]   > [955] polkitd
[14908.617304]   > [968] agetty
[14908.617305]   > [969] lightdm
[14908.617306]   > [1018] whoopsie
[14908.617307]   > [1188] rtkit-daemon
[14908.617308]   > [1194] upowerd
[14908.617308]   > [1210] colord
[14908.617309]   > [1228] systemd
[14908.617310]   > [1235] gnome-keyring-d
[14908.617311]   > [1663] fwupd
[14908.617312]   > [1698] udisksd
[14908.617312]   > This process has 28 child process(es)
[14908.617313] ##### END OF INFORMATION #####
```

- 테스트 프로세스 a.out과 생성한 두 자식 프로세스의 출력 결과이다.
  [13524] a.out은 부모 프로세스로 bash shell을 가지고, 두 자식 프로세스 [13525], [13526]을 가진다. 형제 프로세스는 존재하지 않는다.
  [13525] a.out은 [13524]인 부모 프로세스, [13526]인 형제 프로세스를 가진다. 자식 프로세스는 존재하지 않는다.
  [13526] a.out은 [13524]인 부모 프로세스, [13525]인 형제 프로세스를 가진다. 자식 프로세스는 존재하지 않는다.

```
[14908.617470] ##### TASK INFORMATION of ''[13524] a.out'' #####
[14908.617472] - task state : Running or ready
[14908.617473] - Process Group Leader : [13524] a.out
[14908.617473] - Number of context switched : 0
[14908.617474] - Number of calling fork() :
[14908.617475] - it's parrent process : [1970] bash
[14908.617475] - it's sibling process :
[14908.617476]   > It has no sibling.
[14908.617476] - it's child process(es) :
[14908.617476]   > [13525] a.out
[14908.617477]   > [13526] a.out
[14908.617477]   > This process has 2 child process(es)
[14908.617478] ##### END OF INFORMATION #####
[14909.617953] ##### TASK INFORMATION of ''[13525] a.out'' #####
[14909.617960] - task state : Running or ready
[14909.617963] - Process Group Leader : [13525] a.out
[14909.617964] - Number of context switched : 0
[14909.617965] - Number of calling fork() :
[14909.617967] - it's parrent process : [13524] a.out
[14909.617968] - it's sibling process :
[14909.617970]   > [13526] a.out
[14909.617971]   > This process has 1 sibling process(es)
[14909.617973] - it's child process(es) :
[14909.617974]   > It has no child.
[14909.617975] ##### END OF INFORMATION #####
[14910.618656] ##### TASK INFORMATION of ''[13526] a.out'' #####
[14910.618662] - task state : etc.
[14910.618664] - Process Group Leader : [13526] a.out
[14910.618666] - Number of context switched : 0
[14910.618667] - Number of calling fork() :
[14910.618668] - it's parrent process : [13524] a.out
[14910.618669] - it's sibling process :
[14910.618670]   > [13525] a.out
[14910.618671]   > This process has 1 sibling process(es)
[14910.618672] - it's child process(es) :
[14910.618673]   > It has no child.
[14910.618674] ##### END OF INFORMATION #####
```

- 커널에서 task_struct를 얻는 방법

  커널에서 현재 실행중인 프로세스의 task_struct는 get_current() 함수를 사용해서 얻을 수 있다.

  또한 pid를 통해서 task_struct를 얻을 수 있는데, 이때의 pid는 pid번호 pid_t가 아닌 pid 구조체이다. 따라서 get_pid_task(find_get_pid(pid_t pid), PIDTYPE_PID)와 같이 pid 번호를 통해서 pid 구조체를 얻은 다음, 함수의 인자로 전달해서 얻을 수 있다.

- task_struct 구성요소

int pid;	프로세스 id 정보이다.
char comm[16];	프로세스 이름이다.
volatile long state;	프로세스의 스케줄링 상태를 나타낸다. <linux/sched.h>에 각 상태가 정의되어있다.
struct task_struct *group_leader;	프로세스 리더의 task_struct이다.
long unsigned int nivcsw;	프로세스의 context switching 횟수이다.
struct task_struct *parent;	부모 프로세스의 task_struct이다.
struct list_head sibling;	형제 프로세스 리스트의 list_head이다.
struct list_head children;	자식 프로세스 리스트의 list_head이다.

- 커널에서의 linked list

  커널에서의 linked list는 자료구조 내에 list_head를 추가하고, LIST_HEAD로 선언 및 초기화해서 생성한다. 생성된 list를 순회하려면 다음과 같은 함수를 사용한다.

  list_for_each_entry(pos, head, member)

- fork 횟수

  fork가 발생하면 새 프로세스를 생성하고 task_struct를 복제한다.

  기존 task_struct에는 fork 횟수를 담고 있는 영역이 존재하지 않는다.

  모든 프로세스의 task_struct를 fork 횟수 영역을 포함한 새로운 구조체로 감싸서 대체하는 방법을 생각해보았으나, task struct를 사용하는 다른 시스템에 영향을 끼칠 수 있어서 적용하지 못했다.

3. Reference

3-1. fork vs thread / https://iam777.tistory.com/489

Exit Status / https://www.gnu.org/software/bash/manual/html_node/Exit-Status.html

3-2. 리눅스 스케줄링 / https://jeongchul.tistory.com/95

3-3. 프로세스 상태 / https://sonseungha.tistory.com/248

커널 링크드 리스트 / https://pr0gr4m.tistory.com/entry/Linux-Kernel-5-Linked-List

Task_struct / https://linuxholic.tistory.com/entry/%EB%A6%AC%EB%88%85%EC%8A%A4-Taskstruct-%EA%B5%AC%EC%A1%B0