Chapter 10: Realtime extensions (3/3)

김선영
sunyzero@gmail(dot)com
버 전: 2017-06-08

가메 출판사 http://www.kame.co.kr
저자홈페이지 http://sunyzero.tistory.com

POSIX scheduling API

- scheduling policy, priority
  - posix_spawn, pthread에서도 같은 의미로 적용

```
int  sched_setscheduler(pid_t pid, int policy,
                const struct sched_param *param);
int  sched_getscheduler(pid_t pid);

int  sched_get_priority_max(int policy);
int  sched_get_priority_min(int policy);

int  sched_setparam(pid_t pid, const struct sched_param *param);
int  sched_getparam(pid_t pid, struct sched_param *param);
int  sched_yield(void);
```

POSIX Realtime scheduling

- Scheduling에는 policy, priority 속성을 가진다.
  - Scheduling policy
    - 스케줄링에 사용될 정책은 realtime, non-realtime으로 구분된다.
  - Scheduling priority
    - Realtime scheduling policy에서 사용할 우선순위

Scheduling policy and priority

- 리얼타임과 넌리얼타임 정책이 존재한다.
  - 각 정책은 프로세스 별로 따로 관리된다.
  - 리얼타임 정책이 우선순위가 더 높다.

  - Realtime scheduling
    - FIFO, RR
    - 명령어로 제어하는 경우에는 chrt를 사용한다.
  - Non-realtime scheduling (normal scheduling)
    - OTHER, BATCH, IDLE

Scheduling policy

Realtime and Non-realtime

Scheduling policy

SCHED_FIFO	**FIFO** 방식의 실시간 스케줄링 정책
SCHED_RR	**Round-robin** 방식의 실시간 스케줄링 정책
SCHED_OTHER	임플리먼테이션 고유 스케줄링. **(default)** 리눅스에서는 라운드 로빈 비실시간 스케줄링을 지칭한다
SCHED_BATCH	비표준이며 리눅스 기능. 배치 스타일의 비실시간 스케줄링
SCHED_IDLE	비표준이며 리눅스 기능. 낮은 우선순위로 비실시간 백그라운드 작업.

- SUS 표준 Realtime : SCHED_FIFO, SCHED_RR, SCHED_OTHER
- SUS 표준 Non-realtime : SCHED_OTHER
  - 리눅스에서는 비실시간 RR (I/O처리, 응답)

Policy : Realtime : FIFO, RR

- SCHED_FIFO (First-In-First-Out)
  - 실행 큐 순서대로 실행
  - yield되거나 I/O blocking 전까지 CPU를 사용
  - 만료된 프로세스는 큐의 맨 끝으로 등록된다.
- SCHED_RR (Round robin)
  - 기본적으로 FIFO방식을 따른다.
  - quantum time이 만료되면 선점되어 스케줄러의 맨 뒤에 등록된다.

Policy : Non-realtime : OTHER, BATCH

- SCHED_OTHER (other, default policy)
  - 실행 큐 순서대로 실행, round-robin으로 처리
  - yield되거나 I/O blocking 전까지 CPU를 사용
  - 만료된 프로세스는 큐의 맨 끝으로 등록된다.
  - quantum time이 만료되면 선점되어 스케줄러의 맨 뒤에 등록된다.

Time quantum?

- CPU를 사용할 수 있는 단위 시간.
  - Time slice라고도 부른다.

- Time quantum에 대해서 알아보자.
  - SCHED_RR에서 의미가 있다.

Time quantum : SCHED_RR

- SCHED_RR에서는 interval (time quantum)을 알 수 있다.
  - SCHED_FIFO는 지원하지 않는다. 왜?

```
int sched_rr_get_interval(pid_t pid, struct timespec * tp);
```

  - time quantum
    - kernel 3.9이하 : nice value에 영향을 받음
    - kernel 3.9이상 : /proc/sys/kernel/sched_rr_timeslice_ms
      - default : 100ms

sched_timequantum.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>

int main()
{
    struct timespec ts_timequantum;

    if (sched_rr_get_interval(0, &ts_timequantum) == -1) {
        perror("sched_rr_get_interval");
        return EXIT_FAILURE;
    }

    printf("Time quantum: %lu.%lu\n",
         ts_timequantum.tv_sec, ts_timequantum.tv_nsec);
    return EXIT_SUCCESS;
} /* end : main */
```

Time quantum

- time quantum을 읽어보자. (sched_timequantum.c는 앞에)

```
# chrt -r 10 ./sched_timequantum
Time quantum : 0.100000000
# cat /proc/sys/kernel/sched_rr_timeslice_ms
100
# chrt -r 0 ./sched_timequantum
chrt: failed to set pid 0's policy: Invalid argument

# chrt -f 10 ./sched_timequantum
Time quantum : 0.0

# chrt -o 0 ./sched_timequantum
Time quantum : 0.11000000
# chrt -o 10 ./sched_timequantum
chrt: failed to set pid 0's policy: Invalid argument
```

Scheduling priority

Priority : Realtime

- Realtime은 non-realtime scheduling보다 우선 순위가 높다.
  - RT priority는 1~99의 순위를 가질 수 있다. (시스템에 따라 다를 수 있다.)
    - max, min값을 구해서 확인해야 한다.
    - 따라서 일반적인 경우에는 sched_setscheduler로 policy만 변경하고, 그 이후에 sched_setparam으로 priority를 올려가면서 검증한다.
  - RT pritory 10은 priority -11을 의미한다. (높은 우선 순위)

Priority : Non-realtime

- 보통의 우선 순위
  - 효율적인 I/O 입출력, 응답성을 위해서 쓰인다.
  - 잦은 context switching이 발생할 수 있다.
    - context switching으로 인해 실시간 처리가 힘들 수도 있다.
  - nice 우선순위는 같은 Time sharing의 non-realtime scheduling 프로세스 사이의 우선권을 의미한다.
    - nice : -19 ~ 0 ~ 20
    - 음수일수록 우선권을 가진다.

Priority : Realtime

- Realtime : SCHED_RR, 10

```
struct sched_param  schedp = { .sched_priority = 0 };
if (sched_setscheduler(0, policy, &schedp) == -1) {
    perror("sched_setscheduler");
    return -1;
}
schedp.sched_priority += 10;
if (sched_setparam(0, &schedp) == -1) {
    /* error */
}
```

non-realtime에서는 sched_setparam으로 priority를 조정하면 EINVAL이 발생!

  - nice = 0 :
    - 결과 : PRI = -11, NICE = -
  - nice = -5 :
    - 결과 : PRI = -11, NICE = -

Scheduling : Realtime : priority

- Realtime : SCHED_RR, 2
  - nice = 0 :
    - 결과 : PRI = -3, NICE = -
  - nice = -5 :
    - 결과 : PRI = -3, NICE = -

Realtime scheduling에서는
nice value의 의미를 사용하지 않는다.

Scheduling : Non-Realtime : priority

- Non-Realtime : SCHED_OTHER, 0
  - nice : 0
    - 결과 : PRI = 20, NICE = 0
  - nice : -5
    - 결과 : PRI = 15, NICE = -5

둘 중에 누가 더 높은 우선 순위인가?

20 vs 15

Priority를 알아보는 실험

- sched_quantum.c에 getchar()을 넣어둔다.

```
# chrt -r 10 ./sched_timequantum
```

- htop을 띄우고 filter <F4>를 이용해서 ./sched_ 를 검색하자.
  - 혹은 watch -n 1 ps -o pid,priority,nice,cmd $(pgrep -fx ".*./sched_timequantum")
- 그리고 나서 다른 터미널에서 chrt나 renice를 통해서 제어해보자.

```
# chrt -p 2 $(pgrep -fx ".*./sched_timequantum")

# renice -5 $(pgrep -fx ".*./sched_timequantum")
```

Run : chrt ...

```
# chrt -r 10 ./sched_timequantum
```

```
# ps -o pid,priority,nice,cmd $(pgrep -fx ".*./sched_timequantum")
   PID PRI  NI CMD
 45963 -11   - ./sched_timequantum

# chrt -p 2 $(pgrep -fx ".*./sched_timequantum")
# ps -o pid,priority,nice,cmd $(pgrep -fx ".*./sched_timequantum")
   PID PRI  NI CMD
 45963  -3   - ./sched_timequantum

# renice -5 $(pgrep -fx ".*./sched_timequantum")
45963 (process ID) old priority 0, new priority -5
# ps -o pid,priority,nice,cmd $(pgrep -fx ".*./sched_timequantum")
   PID PRI  NI CMD
 45963  -3   - ./sched_timequantum
```

다른 터미널에서 명령을 내려본다.

Realtime SCHED의 CX

- Context Switching이 발생하는 것을 살펴보자.
  - pthread_pi_cputime.c을 수정하여
  - pthead_pi_cputime_sched.c로 만들자.
    - scheduler를 선택할 수 있도록 수정된 예제

pthread_pi_cputime_sched.c (1/7)

```
#define _GNU_SOURCE
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <time.h>
#include <sched.h>
#define MY_SCHED_PRIO  10
#define NUM_THREADS     3

static int num_steps=1000000000;
double pi, step;

struct start_arg {
    int     idx;
    int     i_start;
    int     i_end;
    double  sum;
} start_arg[NUM_THREADS];

struct timespec diff_ts(struct timespec t1, struct timespec t2);
```

pthread_pi_cputime_sched.c (2/7)

```
void *start_func(void *arg){
    int     i;    double  x, sum = 0.0;
    struct start_arg *p_arg = (struct start_arg *) arg;
#ifdef _POSIX_THREAD_CPUTIME
    int     ret;
    struct timespec     ts1, ts2, ts_diff;
    clockid_t   clock_cpu;
    if ((ret = pthread_getcpuclockid(pthread_self(), &clock_cpu)) != 0) {
        return NULL;
    }
    clock_gettime(clock_cpu, &ts1);
    printf("[T%d] 1: clock_gettime = %ld.%09ld\n", p_arg->idx, ts1.tv_sec, ts1.tv_nsec);
#endif
    for (i=p_arg->i_start; i<p_arg->i_end; i++) {
        x = (i+0.5) * step;
        sum += 4.0/(1.0 + x*x);
    }
    p_arg->sum = sum;
#ifdef _POSIX_THREAD_CPUTIME
    clock_gettime(clock_cpu, &ts2);
    ts_diff = diff_ts(ts1, ts2);
    printf("[T%d] 2: elapsed cpu time = %ld.%09ld\n",
           p_arg->idx, ts_diff.tv_sec, ts_diff.tv_nsec);
#endif
    return NULL;
}
```

pthread_pi_cputime_sched.c (3/7)

```
int sched_set_options(int policy, int prio)
{
    struct sched_param  schedp = { .sched_priority = prio };
    if (sched_setscheduler(0, policy, &schedp) == -1) {
        perror("sched_setscheduler");
        return -1;
    }
    printf("> sched_setscheduler : %d\n", policy);
    return 0;
}

void sched_print_options()
{
    struct sched_param  schedp;
    printf("* current sched_getscheduler = %d\n", sched_getscheduler(0));
    sched_getparam(0, &schedp);
    printf("* current sched_getparam = %d\n", schedp.sched_priority);
}
```

pthread_pi_cputime_sched.c (4/7)

```
int main(int argc, char *argv[])
{
    int     i;
    double  sum;
    pthread_t   pt_id[2];

    step = 1.0/(double) num_steps;
    start_arg[0].i_start = 0;
    start_arg[0].i_end   = num_steps>>1;
    start_arg[1].i_start = start_arg[0].i_end + 1;
    start_arg[1].i_end   = start_arg[1].i_start + (num_steps>>2);
    start_arg[2].i_start = start_arg[1].i_end + 1;
    start_arg[2].i_end   = num_steps;

    sched_print_options();
    if (argc != 2) {
        printf("%s <1 | 2 | 3 | 5>\n", argv[0]);
        printf("1:SCHED_FIFO, 2:SCHED_RR, 3:SCHED_BATCH, 5:SCHED_IDLE\n");
        exit(0);
    }
```

pthread_pi_cputime_sched.c (5/7)

```
switch((char)atoi(argv[1])) {
    case 1:
        sched_set_options( SCHED_FIFO, MY_SCHED_PRIO);
        break;
    case 2:
        sched_set_options( SCHED_RR, MY_SCHED_PRIO);
        break;
    case 3:
        sched_set_options( SCHED_BATCH, 0);
        break;
    case 5:
        sched_set_options( SCHED_IDLE, 0);
        break;
   default:
        sched_set_options( SCHED_OTHER, 0);
        break;
} /* end : switch */
printf("%d~%d, %d~%d, %d~%d\n",
        start_arg[0].i_start, start_arg[0].i_end ,
        start_arg[1].i_start, start_arg[1].i_end ,
        start_arg[2].i_start, start_arg[2].i_end );
```

pthread_pi_cputime_sched.c (6/7)

```
for (i=3; i--; ) { /* create pthread */
        sleep(1);
        start_arg[i].idx = i;
        if (pthread_create(&pt_id[i], NULL, start_func, &start_arg[i])){
            perror("pthread_create");
            return 1;
        }
    }
    for (i=3; i--; ) { /* join pthread : explicit barrier */
        if (pthread_join(pt_id[i], NULL)){
            perror("pthread_join");
            return 1;
        }
    }
    sleep(1);
    sum = start_arg[0].sum + start_arg[1].sum + start_arg[2].sum;
    printf("PI = %.8f (sum = %.8f)\n", step*sum, sum);
    return EXIT_SUCCESS;
} /* end : main */
```

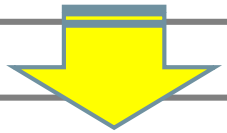
pthread_pi_cputime_sched.c (7/7)

```
struct timespec diff_ts(struct timespec t1, struct timespec t2)
{
    struct timespec t;
    t.tv_sec = t2.tv_sec - t1.tv_sec;
    t.tv_nsec = t2.tv_nsec - t1.tv_nsec;
    if (t.tv_nsec < 0) {
        t.tv_sec--;
        t.tv_nsec += 1000000000;
    }
    return t;
}
```

Run : capabilities

```
[root@fdev 10.realtime]# setcap cap_sys_nice=ep ./pthread_pi_cputime_sched

[root@fdev 10.realtime]# getcap ./pthread_pi_cputime_sched
./pthread_pi_cputime_sched = cap_sys_nice+ep
```



```
[sunyzero@fdev 10.realtime]$ perf stat ./pthread_pi_cputime_sched
... 일반 유저로 sched 설정이 가능해졌다.
```

매번 빌드할 때마다 setcap 설정해주기 귀찮으니
root 계정 암호를 알고 있다면 그것을 사용하자.
혹은 sudo를 설정해두고 사용해도 된다.

sched : perf , CX : FIFO

- 1(SCHED_FIFO)를 사용한 경우

```
# perf stat -d ./pthread_pi_cputime_sched 1
sched_setscheduler : 1
...생략...
 Performance counter stats for './pthread_pi_cputime_sched 1':
      12483.741041      task-clock (msec)         #    1.226 CPUs utilized
                12      context-switches          #    0.001 K/sec
                 6      cpu-migrations            #    0.000 K/sec
                68      page-faults               #    0.005 K/sec
    44,094,424,061      cycles                    #    3.532 GHz
    32,068,742,773      stalled-cycles-frontend   #   72.73% frontend cycles idle
     9,190,887,897      stalled-cycles-backend    #   20.84% backend cycles idle
    44,048,226,705      instructions              #    1.00  insn per cycle
                                                  #    0.73  stalled cycles per insn
     2,008,635,542      branches                  #  160.900 M/sec
            63,334      branch-misses             #    0.00% of all branches
    24,015,739,640      L1-dcache-loads           # 1923.761 M/sec
           312,525      L1-dcache-load-misses     #    0.00% of all L1-dcache hits
           128,704      LLC-loads                 #    0.010 M/sec
   <not supported>      LLC-load-misses

      10.182770880 seconds time elapsed
```

sched : perf , CX : RR

- 2(SCHED_RR)를 사용한 경우

```
# perf stat -d ./pthread_pi_cputime_sched 2
sched_setscheduler : 2
...생략...
      12449.675324      task-clock (msec)         #    1.225 CPUs utilized
                13      context-switches          #    0.001 K/sec
                 5      cpu-migrations            #    0.000 K/sec
                67      page-faults               #    0.005 K/sec
    44,070,607,892      cycles                    #    3.540 GHz
    32,048,677,934      stalled-cycles-frontend   #   72.72% frontend cycles idle
     9,140,887,073      stalled-cycles-backend    #   20.74% backend cycles idle
    44,041,195,965      instructions              #    1.00  insn per cycle
                                                  #    0.73  stalled cycles per insn
     2,007,457,393      branches                  #  161.246 M/sec
            45,388      branch-misses             #    0.00% of all branches
    24,013,736,878      L1-dcache-loads           # 1928.865 M/sec
           177,271      L1-dcache-load-misses     #    0.00% of all L1-dcache hits
            89,482      LLC-loads                 #    0.007 M/sec
   <not supported>      LLC-load-misses

      10.159336672 seconds time elapsed
```

sched : perf , CX : OTHER

- 0(SCHED_OTHER)를 사용한 경우

```
# perf stat -d ./pthread_pi_cputime_sched 0
sched_setscheduler : 0
...생략...
      12380.274380      task-clock (msec)         #    1.226 CPUs utilized
               395      context-switches          #    0.032 K/sec
                 4      cpu-migrations            #    0.000 K/sec
                68      page-faults               #    0.005 K/sec
    44,082,281,461      cycles                    #    3.561 GHz
    32,055,336,891      stalled-cycles-frontend   #   72.72% frontend cycles idle
     9,154,101,409      stalled-cycles-backend    #   20.77% backend cycles idle
    44,053,430,108      instructions              #    1.00  insn per cycle
                                                  #    0.73  stalled cycles per insn
     2,009,607,497      branches                  #  162.323 M/sec
            68,435      branch-misses             #    0.00% of all branches
    24,016,810,178      L1-dcache-loads           # 1939.926 M/sec
           298,743      L1-dcache-load-misses     #    0.00% of all L1-dcache hits
           139,089      LLC-loads                 #    0.011 M/sec
   <not supported>      LLC-load-misses

      10.098680695 seconds time elapsed
```

sched : perf , CX : BATCH

- 3(SCHED_BATCH)를 사용한 경우

```
# perf stat -d ./pthread_pi_cputime_sched 3
sched_setscheduler : 3
...생략...
      12475.587267      task-clock (msec)         #    1.225 CPUs utilized
               103      context-switches          #    0.008 K/sec
                 3      cpu-migrations            #    0.000 K/sec
                70      page-faults               #    0.006 K/sec
    44,084,642,337      cycles                    #    3.534 GHz
    32,058,051,723      stalled-cycles-frontend   #   72.72% frontend cycles idle
     9,141,849,962      stalled-cycles-backend    #   20.74% backend cycles idle
    44,052,728,482      instructions              #    1.00  insn per cycle
                                                  #    0.73  stalled cycles per insn
     2,009,470,719      branches                  #  161.072 M/sec
            59,169      branch-misses             #    0.00% of all branches
    24,016,604,723      L1-dcache-loads           # 1925.088 M/sec
           275,898      L1-dcache-load-misses     #    0.00% of all L1-dcache hits
           116,328      LLC-loads                 #    0.009 M/sec
   <not supported>      LLC-load-misses

      10.181818789 seconds time elapsed
```

sched : perf , CX : IDLE

- 5(SCHED_IDLE)를 사용한 경우

```
# perf stat -d ./pthread_pi_cputime_sched 5
sched_setscheduler : 5
...생략...
      12420.149255      task-clock (msec)         #    1.229 CPUs utilized
               200      context-switches          #    0.016 K/sec
                 4      cpu-migrations            #    0.000 K/sec
                68      page-faults               #    0.005 K/sec
    44,091,095,206      cycles                    #    3.550 GHz
    32,063,893,683      stalled-cycles-frontend   #   72.72% frontend cycles idle
     9,165,243,328      stalled-cycles-backend    #   20.79% backend cycles idle
    44,053,188,318      instructions              #    1.00  insn per cycle
                                                  #    0.73  stalled cycles per insn
     2,009,581,766      branches                  #  161.800 M/sec
            68,670      branch-misses             #    0.00% of all branches
    24,016,744,036      L1-dcache-loads           # 1933.692 M/sec
           280,247      L1-dcache-load-misses     #    0.00% of all L1-dcache hits
           143,483      LLC-loads                 #    0.012 M/sec
   <not supported>      LLC-load-misses

      10.109637913 seconds time elapsed
```

Scheduling plan

- Realtime / pipelining model인 경우
  - 전처리에 가까운 프로세스가 높은 priority를 가지는 것이 유리하다.
  - 후처리에 가까운 프로세스의 속도는 priority보다는 병렬화가 유리하다.
- Non-realtime / pipelining model인 경우
  - 전처리에 가까운 프로세스가 높은 nice / batch를 가지는 것이 유리하다.
  - 후처리에 가까운 프로세스는 덜 높은 nice를 가지는 것이 유리하다.