Chapter 10: Realtime extensions (1/3)

김선영
sunyzero@gmail(dot)com
버 전: 2017-05-27

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

POSIX Realtime extensions

- IEEE std 1003.1b-1993
  - 리얼타임 시그널,
  - 우선순위 스케줄링, 우선순위를 가지는 I/O, 동기 I/O, 파일싱크
  - 타이머,
  - AIO (Asynchronous I/O)
  - mapped file, memory locking
- IEEE std 1003.1d-1999
- IEEE std 1003.1j-2000 Advanced RT ext.

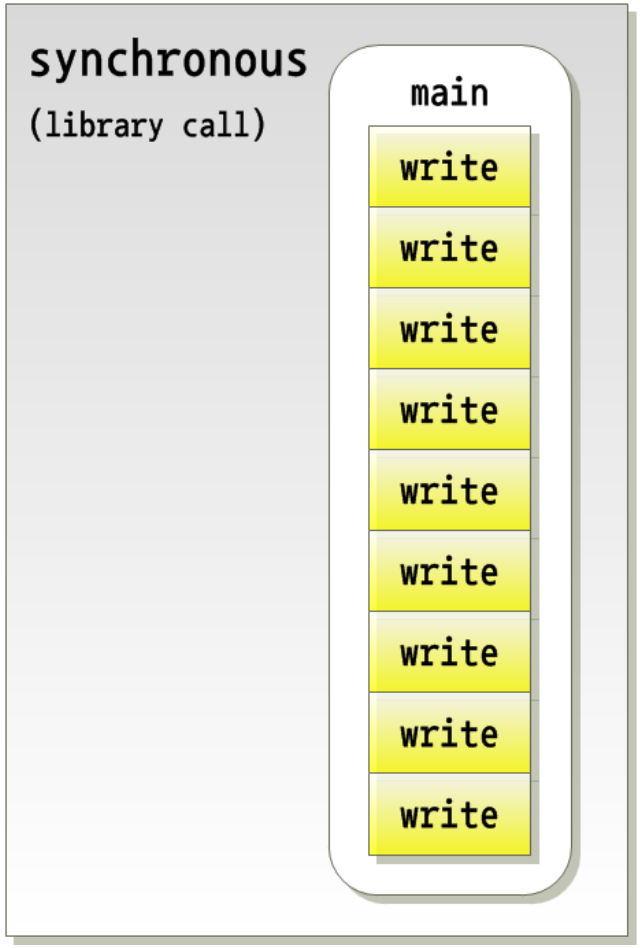
Realtime의 필요성

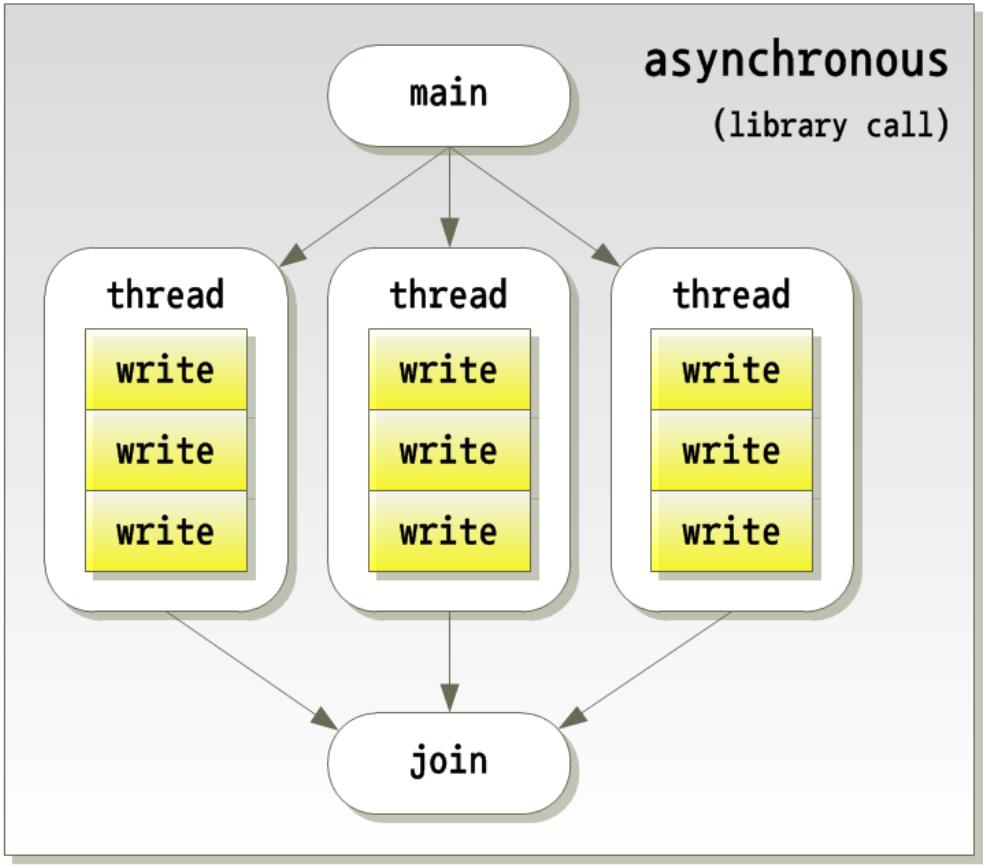
- 자동화 기기
- 군용 무기(Military Spec.)
  - 유도 무기의 속도가 빨라질수록 realtime 센서 데이터의 간격도 짧다
    - 유도 미사일의 지도/영상 데이터 비교
    - 유도 미사일의 적외선/거리 센서의 반응 속도

Async. vs Sync.

- 동기 vs 비동기
  - 동기(synchronous)
  - 비동기(asynchronous)
- 관점
  - 시스템 레벨, 라이브러리 레벨에 따라서 동기, 비동기의 구분은 다르다
  - 관점이 생략되면 주로 시스템 레벨을 지칭하지만, 문맥으로 판단하는게 가장 올바르다.

Async. vs Sync. : lib. level

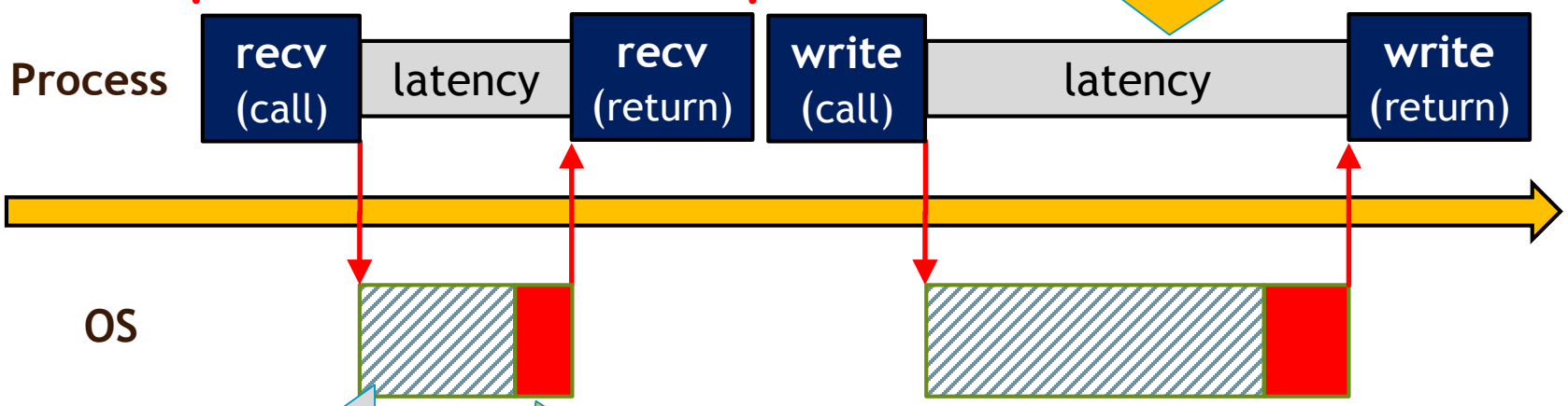




Synchronous : system call

```
len_buf = recv(sd, buf, sz_buf, 0);
write(fd, buf, len_buf);

len_buf2 = recv(sd, buf2, sz_buf2, 0);
write(fd, buf2, len_buf2);
```



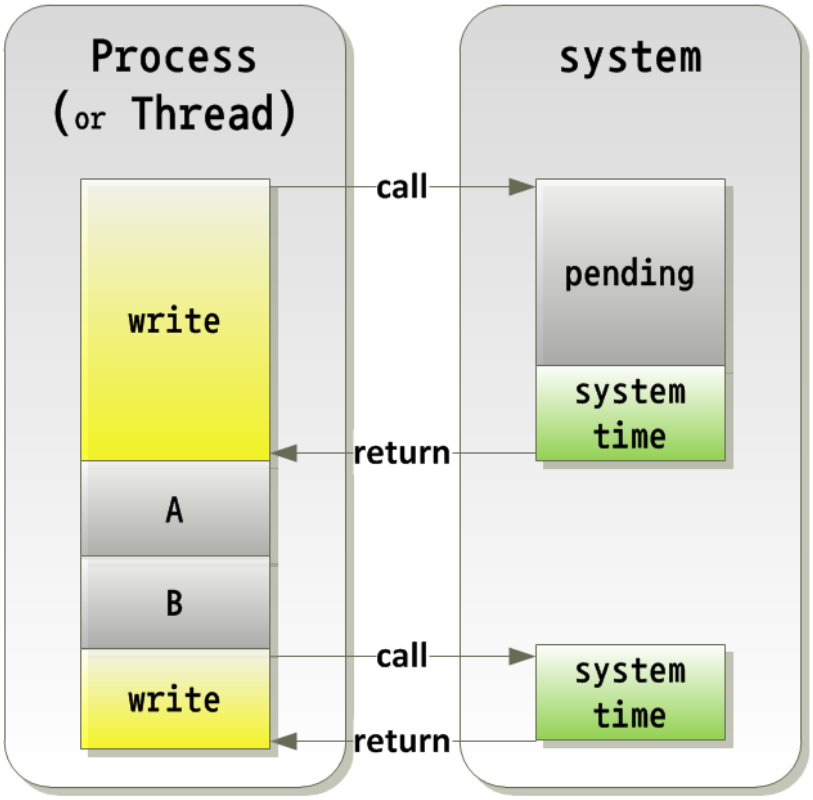
latency는 시스템의 부하상태, 스케줄링 우선순위 등 여러가지 요인에 따라서 달라질 수 있다.

kernel 내부에서 I/O 처리 대기

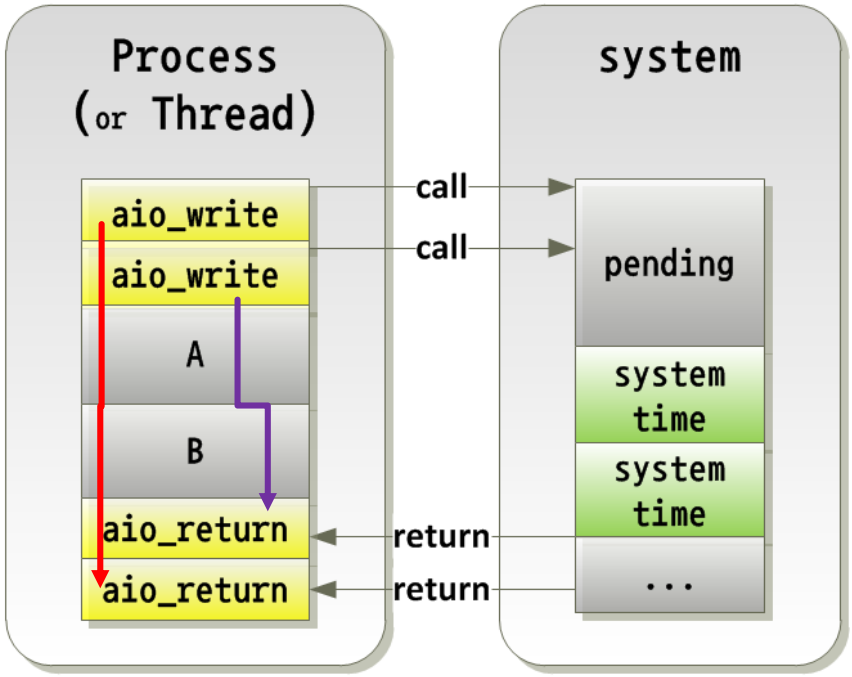
(대기 요인이나 시간은 여러 요인에 따라서 달라진다.)

실제 처리 시간

Async. vs Sync. : sys. call level



system level에서는
스케줄링등 여러 요인으로
처리 시간이 달라지지만
sync.는 처리 순서는 항상 같다.



Async처리인 aio_return의 경우
처리 순서가 달라질 수 있다.

SIGEV event

- POSIX Realtime event mechanism
  - Event information
  - Asynch. Callback : threading, Signaling
- POSIX의 이벤트 알림이 필요한 경우에 사용되는 표준 기능

sigevent

- e.g. mq_notify(…)

```
int mq_notify(mqd_t mqdes, const struct sigevent *notification);
```

```
typedef struct sigevent {
    int   sigev_notify;           /* SIGEV_SIGNAL, SIGEV_NONE, SIGEV_THREAD */
    int   sigev_signo;            /* signal number */
    union sigval   sigev_value;   /* signal value */
    void   (*sigev_notify_function)(union sigval);    /* 통지 쓰레드 함수 */
    pthread_attr_t  *sigev_notify_attributes;         /* 통지 쓰레드 속성 */
};
typedef union sigval {
    int sival_int;     /* int 값을 전달하는 경우의 공용체 멤버 */
    void *sival_ptr;   /* 포인터를 전달하는 경우의 공용체 멤버 */
} sigval_t;
```

sigevent (con't)

- sigev_notify

SIGEV_NONE	이벤트 통지를 사용하지 않습니다.
SIGEV_SIGNAL	이벤트의 통지로 시그널을 발생시킵니다. *** 사용 멤버 : sigev_signo, sigev_val**
SIGEV_THREAD	이벤트의 통지로 쓰레드를 생성하여 처리합니다. *사용 멤버 **: sigev_notify_function,** **sigev_notify_attributes,** **sigev_val**

sigevent (con't)

- SEGEV_SIGNAL

```
typedef struct sigevent {
    int   sigev_notify;           /* SIGEV_SIGNAL, SIGEV_NONE, SIGEV_THREAD */
    int   sigev_signo;            /* signal number */
    union sigval   sigev_value;   /* signal value */
    void   (*sigev_notify_function)(union sigval);    /* 통지 쓰레드 함수 */
    pthread_attr_t  *sigev_notify_attributes;         /* 통지 쓰레드 속성 */
};
typedef union sigval { /* siginfo_t의 si_val에 전달 */
    int sival_int;    /* int 값을 전달하는 경우의 공용체 멤버 */
    void *sival_ptr;  /* 포인터를 전달하는 경우의 공용체 멤버 */
} sigval_t;
```

sigevent (con't)

- siginfo_t 구조체 선언 (padding, 비표준 부분 제외)

```
siginfo_t {
    int      si_signo;  /* 발생된 시그널 번호 */
    int      si_errno;  /* 에러 코드값:용도에 따라서 다르게 사용됨 */
    int      si_code;   /* 시그널의 발생 이유 */
    pid_t    si_pid;    /* 시그널을 보낸 프로세스의 PID */
    uid_t    si_uid;    /* 시그널을 보낸 실제 유저 ID (effiect user id) */
    int      si_status; /* Exit 값이나 시그널 */
    clock_t  si_utime;  /* 소요된 User time */
    clock_t  si_stime;  /* 소요된 System time */
    sigval_t si_value;  /* Signal value:시그널 발생시 전달할 값 (공용체) */
    int      si_int;    /* POSIX.1b signal */
    void *   si_ptr;    /* POSIX.1b signal */
    void *   si_addr;   /* Memory location which caused fault */
    int      si_band;   /* Band event */
    int      si_fd;     /* File descriptor */
}
```

sigevent (con't)

- si_code : negative/0 (from process)

SI_SIGIO	대기된 **SIGIO** 에 의해서 발생한 시그널
SI_ASYNCIO	**AIO(Asynchronous I/O)** 의 완료에 의해서 발생한 시그널
SI_MESGQ	실시간 메시지큐**(POSIX MQ)**의 상태가 변화되어 발생한 시그널
SI_TIMER	타이머가 만료되어 발생한 시그널
SI_QUEUE	**sigqueue(2)** 함수에 의해서 전달되어진 시그널
SI_USER	**kill(2), raise(2)** 와 같은 함수에 의해서 전달되어진 시그널 **(**보통 **0** 의 값을 지님**)**

  - Ref. Implementation API manual
  - e.g. SIGSEGV -> SEGV_MAPERR or SEGV_ACCERR

Realtime Signaling

```
int sigwaitinfo(const sigset_t *set, siginfo_t *info);
int  sigtimedwait(const  sigset_t  *set,
                        siginfo_t *info, const struct timespec timeout);
int sigqueue(pid_t pid, int sig, const union sigval value);
union sigval {
    int sival_int;    /* int 값을 전달하는 경우의 공용체 멤버 */
    void *sival_ptr; /* 포인터를 전달하는 경우의 공용체 멤버 */
}
```

- RT signal recv'ing : sigwaitinfo, sigtimedwait
- RT singal queuing : sigqueue

Realtime Signaling (con't)

```
union sigval    si_val;
…생략…
si_val.sival_ptr = address to something;
if (sigqueue(getpid(), SIGRTMIN, si_val)) {
    /* error */
}
```

```
void h_sigev_rt1(int signo, siginfo_t *si, void *data)
{   /* RT 시그널 핸들러 */
    printf("\t[RTS] Ptr(%p)\n", si->si_value.sival_ptr);
}
…생략…
struct sigaction sa_rt1;
memset(&sa_rt1, 0, sizeof(sa_rt1));
sa_rt1.sa_flags = SA_SIGINFO | SA_RESTART; /* RTS이므로 SA_SIGINFO로 */
sa_rt1.sa_sigaction = h_sigev_rt1; /* SA_SIGINFO -> sa_handler 대신… */
sigaction(SIGRTMIN, &sa_rt1, NULL);
```

Realtime Signaling (con't)

```
union sigval    si_val;
…생략…
si_val.sival_int = number; /* int type 설정 가능 */
if (sigqueue(getpid(), SIGRTMIN, si_val)) {
    /* error */
}
…생략…
void h_sigev_rt1(int signo, siginfo_t *si, void *data)
{   /* RT 시그널 핸들러 */
    printf("\t[RTS] Integer value (%d)\n",
            si->si_value.sival_int);
}
```

union으로
구성되어있기에

Practice: siginfo_t

- 기존의 signal handler 함수를 siginfo_t 함수형으로 바꿔보자.
  - siginfo_t의 si_code, pid, uid, time등을 출력해보자.

break time

RTS: communication

- RTS signal & signal queue
  - cond_var_rts.c
- RTS I/O multiplexing
  - io_rts.c
- Dir. event Notification
  - dir_rt.c

timer - obsolete

traditional timer

sleep() **usleep() - BSD**	초, 마이크로초의 단위로 프로세스를 재울 수 있다. 시간이 만료되거나 시그널이 도착하면 깨어난다.
alarm()	알람 시그널을 지정된 시간 이후에 발생시킨다. **(old fashioned)**
getitimer() **setitimer()**	구간 타이머**(interval timer)**로서 정해진 시간마다 알람을 발생한다.

- no guarantee on MT environment
- system scope (uncertain)

RT timer: func. list

- IEEE std 1003.1b, process scope

timer_create()	타이머를 생성: 생성시 타이머 만료 이벤트를 지정할 수 있다.
timer_delete()	타이머를 삭제
timer_settime()	타이머 속성(만료시간, 주기적 만료시간)을 설정하여 작동시킨다.
timer_gettime()	타이머의 시간속성을 읽어온다.
timer_getoverrun()	오버런(만료되어 전달된)된 타이머의 카운트 개수를 리턴한다.

RT timer: func. list (con't)

- IEEE std 1003.1b, process scope

clock_getres()	특정 시계의 정밀도**(resolution or precision)**를 가져온다**.**
clock_gettime()	특정 시계의 시간을 가져온다**.**
clock_settime()	특정 시계의 시간을 저장한다**.** **(CLOCK_REALTIME** 만 가능하며 수퍼유저의 권한이 필요하다**)**
clock_getcpuclockid()	**CPU** 사용시간을 측정하는 시계를 가져온다**.** **(_POSIX_CPUTIME** 매크로가 지정되어야만 사용 가능하다**)**
nanosleep()	**Nano second**의 **sleep**

RT clock

```
int clock_getres(clockid_t clk_id, struct timespec *res);
int clock_gettime(clockid_t clk_id, struct timespec *tp);
int clock_settime(clockid_t clk_id, const struct timespec *tp);
```

- clockid_t 타입 (POSIX spec.)

CLOCK_REALTIME	시스템 전역의 실제 시계를 사용 **(The UNIX Epoch** 시간**)**
CLOCK_MONOTONIC	**monotonic clock :** 특정시각을 기준으로 시간을 측정 **(e.g.** 리눅스의 경우 **boot time**을 기준, 대부분 부트 기준**)**

* monotonic clock의 필요성 - difference time을 구할 때 시스템 시간을 사용하면 NTP에 의해서 시간 보정시 시간이 달라질 수 있기 때문이다.

RT clock

- clockid_t 타입 (POSIX optional / Linux extensions)

CLOCK_MONOTONIC_RAW	하드웨어에 기반한 단조시계 **(보정을 거치지 않으므로 빠르다)** **- Linux specific.**
CLOCK_PROCESS_CPUTIME_ID	프로세스 단위 **CPU** 시간 측정 시계 **(_POSIX_CPUTIME** 매크로가 정의된 경우에 지원한다.**)**
CLOCK_THREAD_CPUTIME_ID	스레드 단위 **CPU** 시간 측정 시계 **(_POSIX_THREAD_CPUTIME** 매크로가 정의된 경우에 지원한다.**)**

clock resolution

```
$ ./clock_getres
clock precision = 0.000000001
```

```
$ ./clock_getres
clock precision = 0.000999848
```

```
$ ./clock_getres
clock precision = 0.004000250
```

- H/W에 따라서 다르다.

clock: example

```
struct timespec tspec;

clock_getres(CLOCK_REALTIME, &tspec);
printf("clock precision = %d.%09ld\n", tspec.tv_sec, tspec.tv_nsec);

clock_gettime(CLOCK_REALTIME, &tspec);
printf("REALTIME Clock = %d.%09ld\n", tspec.tv_sec, tspec.tv_nsec);

clock_gettime(CLOCK_MONOTONIC, &tspec);
printf("MONOTONIC Clock = %d.%09ld\n", tspec.tv_sec, tspec.tv_nsec);
```

RT timer : func.

```
int timer_create(clockid_t clockid, struct sigevent *restrict evp,
          timer_t *restrict timerid);
int timer_delete(timer_t timerid);
int timer_gettime(timer_t timerid, struct itimerspec *value);
int timer_settime(timer_t timerid, int flags,
          const struct itimerspec *restrict value,
          struct itimerspec *restrict ovalue);
int timer_getoverrun(timer_t timerid);
```

- 리얼타임 타이머는 앞서 리얼타임 클락을 기반으로 사용되는 타이머다.
  - 1회용, 인터벌 타이머를 지원한다.
  - 타이머 만료시 실행할 RT SIGEV를 지정할 수 있다.(스레드 생성, 시그널 전송)

RT timer : timer_settime

```
int timer_settime(timer_t timerid, int flags,
          const struct itimerspec *restrict value,
          struct itimerspec *restrict ovalue);
```

```
struct itimerspec {
    struct timespec it_interval;
    struct timespec it_value;
};
```

```
struct timespec {
    time_t tv_sec;   /* second */
    long tv_nsec;    /* nano second */
};
```

it_value	> 0.0	첫 타이머 만료시간을 설정. 지정된 시간후 타이머는 만료됩니다.
	0.0	타이머는 무효화 됨. 이미 설정된 타이머가 있다면 해제됩니다.
it_interval	> 0.0	해당 시간마다 타이머 만료가 호출 (인터벌).
	0.0	인터벌 타이머 기능은 무효화 됨. 타이머는 일회성으로 작동합니다.

* itimerspec 의 설정 (소수점이하는 tv_nsec 멤버의 나노초로 설정합니다.)

RT timer : timer_settime (con't)

```
struct itimerspec {
    struct timespec it_interval;
    struct timespec it_value;
};
```

```
struct timespec {
    time_t tv_sec;   /* second */
    long tv_nsec;    /* nano second */
};
```

flags	description
TIMER_ABSTIME	**itimerspec**의 시간을 절대시간으로 사용 **realtime clock**인 경우는 **UNIX epoch**를 기준으로 하고 **monotonic clock**인 경우는 특정시각을 기준으로 절대시간을 구함

* itimerspec 의 설정 (소수점이하는 tv_nsec 멤버의 나노초로 설정합니다.)

RT timer (con't)

```
if (timer_create(CLOCK_REALTIME, &sigev, &rt_timer) == -1) {
    perror("FAIL: timer_create()");    return -1;
}
rt_itspec.it_value.tv_sec = 2; /* initial timer */
rt_itspec.it_value.tv_nsec = 500000000; /* 500,000,000 = 0.5 sec */
rt_itspec.it_interval.tv_sec = 4; /* expire every 4 seconds */
rt_itspec.it_interval.tv_nsec = 0;

if (timer_settime(rt_timer, 0, &rt_itspec, NULL) == -1) { /* run timer */
    perror("FAIL: timer_settime()");
    return -1;
}
```

* itimerspec 의 설정 (소수점이하는 tv_nsec 멤버의 나노초로 설정합니다.)

RT timer (con't)

```
if (timer_create(CLOCK_MONOTONIC, &sigev, &rt_timer) == -1) {
    perror("FAIL: timer_create()");
    return -1;
}
clock_gettime(CLOCK_MONOTONIC, &tspec); /* time from monotonic clock */
tspec.tv_sec += 2;
tspec.tv_nsec = 500000000;
rt_itspec.it_value = tspec;
rt_itspec.it_interval.tv_sec = 4; /* expire eve
rt_itspec.it_interval.tv_nsec = 0;
if (timer_settime(rt_timer, TIMER_ABSTIME, &rt_itspec, NULL) == -1) {
    perror("FAIL: timer_settime()");
    return -1;
}
```

구조만 눈여겨 보자.
ABSTIME을 주의깊게..

RT timer: example

- rt_timer.c

```
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <sys/time.h>

#define GET_TIME0(a)    get_time0(a, sizeof(a)) == NULL ? "error" : a
char * get_time0(char *buf, size_t sz_buf);

int inst_timer(void);
void sa_sigaction_rtmin(int signum, siginfo_t *si, void *sv);
```

RT timer: example (con't)

```
int main()
{
    if (inst_timer() == -1) {
        return EXIT_FAILURE;
    }
    while (1) {
        pause();
    }

    return 0;
} /* end : main */
```

RT timer: example (con't)

```
int inst_timer(void) {
    struct sigaction   sa_rt1;
    struct sigevent    sigev;   /* signal event */
    timer_t    rt_timer;        /* timer id */
    struct itimerspec  rt_itspec;
    char    ts_now[20];

    memset(&sa_rt1, 0, sizeof(sa_rt1));
    sigemptyset(&sa_rt1.sa_mask);
    sa_rt1.sa_sigaction = sa_sigaction_rtmin; /* rt_timer handler */
    sa_rt1.sa_flags = SA_SIGINFO;

    if (sigaction(SIGRTMIN, &sa_rt1, NULL) == -1) {
        perror("FAIL: sigaction()");
        return -1;
    }
    sigev.sigev_notify = SIGEV_SIGNAL; /* notification with signal */
    sigev.sigev_signo = SIGRTMIN;
```

타이머를 설치하기 전에
SIGEV realtime event를 만들자.

RT timer: example (con't)

```
 /* create timer */
    if (timer_create(CLOCK_MONOTONIC, &sigev, &rt_timer) == -1) {
        perror("FAIL: timer_create()");
        return -1;
    }

    /* interval timer setting */
    printf("Enable timer at %s\n", GET_TIME0(ts_now));
    rt_itspec.it_value.tv_sec = 2 ;
    rt_itspec.it_value.tv_nsec = 500000000; /* 0.5 sec */
    rt_itspec.it_interval.tv_sec = 4; /* periodic timer with 4 sec. */
    rt_itspec.it_interval.tv_nsec = 0;
    if (timer_settime(rt_timer, 0, &rt_itspec, NULL) == -1) {
        perror("FAIL: timer_settime()");
        return -1;
    }
    return 0;
}
```

RT timer: example (con't)

```
void sa_sigaction_rtmin(int signum, siginfo_t *si, void *sv)
{
    char    ts_now[20];
    printf("[%d] -> RT timer expiration at %s\n",
          (int)getpid(), GET_TIME0(ts_now));
}
```

RT timer: example (con't)

```
char * get_time0(char *buf, size_t sz_buf) {
#define STR_TIME_FORMAT     "%H:%M:%S"
    struct timespec tspec;
    struct tm   tm_now;
    size_t  sz_ret;
    if (buf == NULL) return NULL;
    if (clock_gettime(CLOCK_REALTIME, &tspec) == -1) {
        return NULL;
    }

    localtime_r((time_t *)&tspec.tv_sec, &tm_now);
    if ((sz_ret = strftime(buf, sz_buf, STR_TIME_FORMAT, &tm_now)) == 0) {
        return NULL;
    }
    snprintf(buf + sz_ret, sizeof(buf), ".%09ld", (long)tspec.tv_nsec);
    return buf;
}
```

RT timer: example (con't)

- Run "rt_timer"

```
$ ./rt_timer
Enable timer at 17:46:41.
-> RT timer expiration at 17:46:44
-> RT timer expiration at 17:46:48
-> RT timer expiration at 17:46:52
```

CPU clock

- CPU time을 측정 : 성능 측정, 로깅에 주로 사용

```
int clock_getcpuclockid(pid_t pid, clockid_t *clock_id);
int pthread_getcpuclockid(pthread_t pthread_id, clock_t *clock_id)
```

  - CLOCK_PROCESS_CPUTIME_ID
  - CLOCK_THREAD_CPUTIME_ID

CPU clock: example

- cputime_process.c

```
#define _XOPEN_SOURCE   600
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int num_steps=200000000; /* integration 횟수: 2억번 (너무 많으면 줄이자.) */
struct timespec diff_timespec(struct timespec t1, struct timespec t2);
int main() {
    int ret;
#ifdef _POSIX_CPUTIME
    struct timespec     ts1, ts2, ts_diff;
    clockid_t   clock_cpu;
    if ((ret = clock_getcpuclockid(0, &clock_cpu)) != 0) {
        return EXIT_FAILURE;
    }
    clock_gettime(clock_cpu, &ts1);
#endif
```

CPU clock: example (con't)

- cputime_process.c

```
    int i;
    double x, step, sum = 0.0;
    step = 1.0/(double) num_steps;
    for (i=0; i<num_steps; i++) {
        x = (i+0.5) * step;
        sum += 4.0/(1.0 + x*x);
    }
    printf("pi = %.8f (sum = %.8f)\n", step*sum, sum);
    sleep(1);
#ifdef _POSIX_CPUTIME
    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &ts2);
    ts_diff = diff_timespec(ts1, ts2);
    printf("elapsed cpu time = %ld.%09ld\n", ts_diff.tv_sec, ts_diff.tv_nsec);
#endif
    return EXIT_SUCCESS;
} /* end : main */
```

pi를 구하면서 CPU를 혹사시킨다

하지만 중간에 1 sec.를 sleep해서 CPU clock과 real clock의 값이 차이나게 한다.

CPU clock: example (con't)

- cputime_process.c

```
struct timespec diff_timespec(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;
}
```

cputime_process: exec

```
$ time ./cputime_process
pi = 3.14159265 (sum = 628318530.71809554)
elapsed cpu time = 3.235577332

real    0m4.239s
user    0m3.235s
sys     0m0.001s
```

1초를 sleep 했기 때문에 real과 user의 시간이 달라졌다.