chapter09. Times and Timers

objectives

1. Learn how time is represented
2. Experiment with interval timers
3. Explore interactions of timers and signals
4. Use timers to assess performance

POSIX Times

system : Epoch 이후로 초 단위로 시간을 저장 중

- Epoch : 1970년 1월 1일 00:00으로 define(UTC, GMT)

POSIX : 실제 시간과 시스템 시간을 맞추는 걸 구현 x

Time in seconds

```
#include <time.h>

time_t time(time_t *tloc);
//Epoch 시점부터 몇 초가 흘렀는지 알려줌(UTC 시간)

//tloc : NULL이 아니면 시간을 tloc 포인터에 저장 가능(time_t : long type)

//return (Epoch 시점부터 흐른 초) -> successful
//return (time_t) -1 -> unsuccessful
//error는 define x
```

```
#include <time.h>

double difftime(time_t time1, time_t time0);
//time 간의 차이를 계산

//return (double)(time1 - time2) -> successful
//error는 define x
```

Displaying date and time

```c
#include <time.h>

struct tm *gmtime(const time_t *timer);
//UTC로 표시되는 timer 입력 받아 tm 구조체로 return return

struct tm *localtime(const time_t *timer);
//현재 시간(Epoch 이후의 초) 중 원하는 시간 단위를 tm 구조체의 filed에서 return

struct tm{
  int tm_sec; /* seconds [0,59]*/
  int tm_min; /* minutes [0,59] */
  int tm_hour; /* hours [0,23] */
  int tm_mday; /* day of the months [1,31] */
  int tm_mon; /* months [0, 11] */
  int tm_year; /* years since 1900 */
  int tm_wday; /* days since Sunday [0,6] */
  int tm_yday; /* days since January 1 [0, 365] */
  int tm_isdst; /* flag for daylight-saving time */
}

char* asctime(const struct tm *timeptr);
//localtime()의 반환값을 string으로 변경
//ex. asctime(localtime(time(Null)));

char* ctime(const time_t *clock);
//asctime()이랑 기능은 같지만 사람이 보기 편한 형태로 출력
// YYYY/MM/DD 형식과 유사한 str의 pointer 값 return
//ex. ctime(time(Null));

// str을 가지고 있기 위해 static store 사용
```

- asctime, ctime, localtime → non thread-safe
  - 여러 thread가 동시에 호출하면 충돌 발생 가능(내부적으로 static 저장소 이용)

    → 나중 호출이 덮어 씌울 수 있음

    → 추가적인 buffer parameter로 해결

```c
#include <sys/time.h>

int gettimeofday(struct timeval *restrict tp, void *restrict tzp);
//Epoch 시점 이후의 시간을 ms까지 return

//tp : output parameter
//tzp : 사라진 parameter로 Null
struct timeval{
  time_t tv_sec; /* seconds since the Epoch */
  time_t tv_usec; /* and microseconds */
}
```

- Program 9.1 ⇒ gettimeofday를 사용하여 running time check.

```c
#include <stdio.h>
#include <sys/time.h>
#define MILLION 1000000L
```

```
void function_to_time(void);

int main(void) {
   long timedif;
   struct timeval tpend;
   struct timeval tpstart;

   if (gettimeofday(&tpstart, NULL)) {
      fprintf(stderr, "Failed to get start time\n");
      return 1;
   }
   function_to_time();                              /* timed code goes here */
   if (gettimeofday(&tpend, NULL)) {
      fprintf(stderr, "Failed to get end time\n");
      return 1;
   }
   timedif = MILLION*(tpend.tv_sec - tpstart.tv_sec) +
                     tpend.tv_usec - tpstart.tv_usec;
   printf("The function_to_time took %ld microseconds\n", timedif);
   return 0;
}
```

```
#include <unistd.h>
#include <sys/time.h>

void function_to_time() {
   int i;
   for (i=0;i<100000000;i++) ;
}

void function_to_time() {
   struct timeval tlast;
   gettimeofday(&tlast, NULL);
   gettimeofday(&tlast, NULL);
   gettimeofday(&tlast, NULL);
   gettimeofday(&tlast, NULL);
   gettimeofday(&tlast, NULL);
   gettimeofday(&tlast, NULL);
   gettimeofday(&tlast, NULL);
   gettimeofday(&tlast, NULL);
   gettimeofday(&tlast, NULL);
   gettimeofday(&tlast, NULL);
}
```

Using real-time clocks

- clock
  - clock resolution(고정된 intervel) 값만큼 증가하는 counter
  - fixed intervals : 시간의 정확도 결정 -> clock resolution으로 세밀함 결정
  - POSIX:TMR Timers Extension : clokid_t type인 clock 포함
  - clokid_t = CLOCK_REALTIME → 실시간으로 기본적인 시간 정보를 알려줌

```
#include <time.h>

int clock_getres(clockid_t clock_id, struct timespec *res);
//clock resolution 반환
int clock_gettime(clockid_t clock_id, struct timespec *tp);
//Epoch 시점부터 현재까지의 시간을 ns 단위까지 변환
int clock_settime(clockid_t clock_id, const struct timespec *tp);
//clock_id를 tp에 설정
```

```
struct timespec{
  time_t tv_sec; /* seconds */
  long tv_nsec; /* nanoseconds */
}
```

```
//return 0 -> successful
//return -1 -> unsuccessful
```

- example

```
#include <stdio.h>
#include <time.h>
#define MILLION 1000000L

void function_to_time(void);
  int main (void) {
  long timedif;
  struct timespec tpend, tpstart;
  if (clock_gettime(CLOCK_REALTIME, &tpstart) == -1) {
    perror("Failed to get starting time");
    return 1;
  }
  function_to_time(); /* timed code goes here */
  if (clock_gettime(CLOCK_REALTIME, &tpend) == -1) {
    perror("Failed to get ending time");
    return 1;
  }
  timedif = MILLION*(tpend.tv_sec-tpstart.tv_sec)+(tpend.tv_nsec-tpstart.tv_nsec)/1000;
  printf("The function_to_time took %ld microseconds\n", timedif);
  return 0;
}
```

Sleep functions

```
#include <unistd.h>
unsigned sleep(unsigned seconds);
//seconds만큼 지나거나 호출한 thread가 signal을 받을 때까지 호출한 thread를 suspend

//return 0 -> 요청 시간 경과
//return (unslept time) -> 중간에 interrupt 되면 남아있던 시간 반환

#include <time.h>
int nanosleep(const struct timespec *rqtp, struct timespec *rmtp);
//rqtp(nano 단위 시간 설정 가능)에 의해 설정된 시간이 끝나거나 thread가 signal 받을 때까지
//thread suspend
```

```
//rmpt :
// -> not NULL : 남아있는 시간을 포함

//return 0 -> successful
//return -1, errno -> unsuccessful
```

Timer

POSIX:XSI interval Timer

A computer system typically has a small number of hardware interval timers

- clock : 시간이 계속 증가 / timer : 시간이 계속 감소
- OS : 이러한 h/w timer을 사용하기 위해 다양한 s/w timer 제공
- time-sharing OS : process scheduling(context-switch)를 위해 interval timer 사용 가능

```
struct timeval{
  time_t tv_sec; /* seconds since the Epoch */
  time_t tv_usec; /* and microseconds */
}
struct itimerval{
  struct timeval it_value; /* time until next expiration */
    //초기값 저장
  struct timeval it_interval; /* value to reload into the timer */
    //timer 반복 여부 -> 한 번이면 0, 여러 번이면 만료 시점(횟수) 설정
}
```

```
#include <sys/time.h>
int getitimer(int which, struct itimerval *value);
//'which' timer의 남아있는 시간을 'value'에 저장

//timer 종류(which 종류)

//1.ITIMER_REAL : 일반적인 경우, 벽시계처럼 작동
//   -> decrements in real time + 다 끝나면 SIGALRM 생성

//2.ITIMER_VIRTUAL : 가상시간으로 작동(process가 실행 중일 때 작동), 중간에 중단 가능
//   -> decrements in virtual time + process가 끝나거나 timer가 끝나면 SIGVTALRM 생성

//3.ITIMER_PROF : process가 실행 중일 때 작동(virtual과 동일)
//   -> devrements in virtual time and system time for the process + SIGPROF 생성

int setitimer(int which, const struct itimerval *restrict value,
              struct itimerval *restrict ovalue);
//'which' timer의 시간을 'value'로 설정

//ovalue
//   -> not NULL : 원래 값 저장
//   -> value->it_interval != 0 -> 종료될 때까지 반복해서 시작
```

```
//   -> value->it_interval == 0 -> 반복 x
//   -> value->it_value == 0 -> 기존에 작동하던 timer 중지
```

- example

```
//program 9.7 -> 2초마다 signal handler(* 출력) 실행

//ITIMER_PROF timer 사용 : SIGPROF가 캐치되면 signal handler 실행
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>

/* ARGSUSED */
static void myhandler(int s) {
   char aster = '*';
   int errsave;
   errsave = errno;
   write(STDERR_FILENO, &aster, 1);
   errno = errsave;
}

static int setupinterrupt(void) {          /* set up myhandler for  SIGPROF */
   struct sigaction act;
   act.sa_handler = myhandler;
   act.sa_flags = 0;
   return (sigemptyset(&act.sa_mask) || sigaction(SIGPROF, &act, NULL));
}

static int setupitimer(void) {    /* set ITIMER_PROF for 2-second intervals */
   struct itimerval value;
   value.it_interval.tv_sec = 2;
   value.it_interval.tv_usec = 0;
   value.it_value = value.it_interval;
   return (setitimer(ITIMER_PROF, &value, NULL));
}

int main(void) {
   if (setupinterrupt()) {
      perror("Failed to set up handler for SIGPROF");
      return 1;
   }
   if (setupitimer() == -1) {
      perror("Failed to set up the ITIMER_PROF interval timer");
      return 1;
   }
   for ( ; ; );                        /* execute rest of main program here */
}
```

```
//program 9.8 -> 함수의 성능 측정을 timer로

//ITIMER_VIRTUAL 사용
#include <stdio.h>
#include <sys/time.h>
#define MILLION 1000000L
```

```
void function_to_time(void);

int main(void) {
   long diftime;
   struct itimerval ovalue, value;

   ovalue.it_interval.tv_sec = 0;
   ovalue.it_interval.tv_usec = 0;
   ovalue.it_value.tv_sec = MILLION;                    /* a large number */
   ovalue.it_value.tv_usec = 0;
   if (setitimer(ITIMER_VIRTUAL, &ovalue, NULL) == -1) {
      perror("Failed to set virtual timer");
      return 1;
   }
   function_to_time();                                  /* timed code goes here */
   if (getitimer(ITIMER_VIRTUAL, &value) == -1) {
      perror("Failed to get virtual timer");
      return 1;
   }
   diftime = MILLION*(ovalue.it_value.tv_sec - value.it_value.tv_sec) +
               ovalue.it_value.tv_usec - value.it_value.tv_usec;
   printf("The function_to_time took %ld microseconds or %f seconds.\n",
            diftime, diftime/(double)MILLION);
   return 0;
}
```

timer 초기값 (적당히 큰 값)
① timer 설정
② 남아있는 시간을 value에 저장
process가 실행 중 일 때만 동작

POSIX:TMR interval Timer

CLOCK_REALTIME과 같이 small number clock 존재

→ process는 이런 독립적인 timer를 여러 개 생성 가능

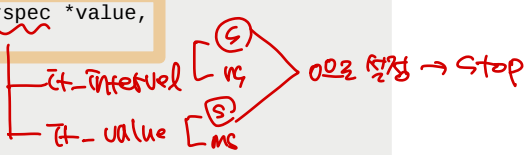
```
struct timespec{
①time_t tv_sec; /* seconds */
②long tv_nsec; /* nanoseconds */
}

struct itimerspec{
①struct timespec it_interval; //timer period
②struct timespec it_value; //expiration
}

//timeval보다 더 나은 solution
```

ns까지 표현 가능

```
#include <signal.h>
#include <time.h>

int timer_create(clockid_t clock_id, struct sigevent *restrict evp,
  timer_t *restrict timerid);
//create(process 단위의 timer) -> timer 객체 생성
//fork()에 의해 상속 x

//clock_id : timer의 종류
//timerid : 생성된 timer의 id를 반환 -> output parameter
//evp : timer가 끝나고 async. notification(siganl) 받을 지 결정
```

```
//  -> NULL : default siganl(SIGALRM for CLOCK_REALTIME)
//  -> evp->sigev_signo : desired signal number
//  -> evp->sigev_notify : timer가 종료될 때 수행될 action을 설정
//        1. SIGEV_SIGNAL : timer가 종료될 때 signal 생성
//        2. SIGEV_NONE : sinal를 생성하는 것으로부터 timer를 prevent

int timer_delete(timer_t timerid);
//timer 삭제

//return 0 -> successful
//return -1 -> fail
```

```
#include <time.h>

int timer_getoverrun(timer_t timerid);
//반복되어 실행되는 timer의 경우 overrun 발생
//overrun : 동일한 타이머의 이전 종료로부터 signal이 pending
//          -> pending 중에 다음 타이머 종료 시 생성된 signal이 사라짐
//          -> 이와 같이 lost된 signal 개수를 뜻함

//timerid의 overrun 개수를 구함

//return overrun number

int timer_gettime(timer_t timerid, struct itimerspec *value);
//남아 있는 timer 값 반환

int timer_settime(timer_t timerid, int flags, const struct itimerspec *value,
    struct itimerspec *ovalue);
//timer를 start or stop

//flags : timer -> relative time or absolute time 결정
//ovalue
//  -> not NULL : 이전 값을 저장 -> output parameter
```



- example → program 9.12
  - func_to_time() 수행 시간 측정 ⇒ interval timer로 real time 측정

```
#include <stdio.h>
#include <time.h>
#define MILLION 1000000L
#define THOUSAND 1000

void function_to_time(void);

int main(void) {
   long diftime;
   struct itimerspec nvalue, ovalue;
   timer_t timeid;
                                ①Create
   if (timer_create(CLOCK_REALTIME, NULL, &timeid) == -1) {
      perror("Failed to create a timer based on CLOCK_REALTIME");
      return 1;
   }
   ovalue.it_interval.tv_sec = 0;
```

```
    ovalue.it_interval.tv_nsec = 0;
    ovalue.it_value.tv_sec = MILLION;              /* a large number */
    ovalue.it_value.tv_nsec = 0;  ② timer 설정 → 10000초
    if (timer_settime(timeid, 0, &ovalue, NULL) == -1) {
       perror("Failed to set interval timer");
       return 1;
    }
    function_to_time();  → 남아있는 timer값   /* timed code goes here */
    if (timer_gettime(timeid, &nvalue) == -1) {
       perror("Failed to get interval timer value");
       return 1;
    }
    diftime = MILLION*(ovalue.it_value.tv_sec - nvalue.it_value.tv_sec) +
       (ovalue.it_value.tv_nsec - nvalue.it_value.tv_nsec)/THOUSAND;
    printf("The function_to_time took %ld microseconds or %f seconds.\n",
           diftime, diftime/(double)MILLION);
    return 0;
}
```

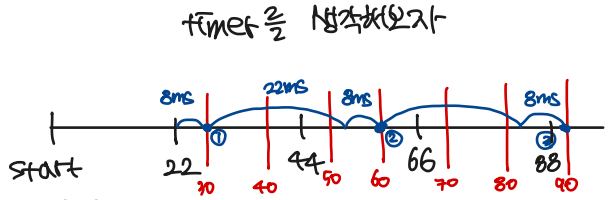
Timer drift

timer 반복적으로 사용하는 경우, delay가 발생하여 예상 시점보다 뒤로 밀림

→ timer 특성 상 태생적인 문제

⇒ 완전 0으로는 만들 수 X 줄여야 함.

- reason : 반복적으로 실행되는 경우 만료되었다가 다시 시작하는 latency 발생
- example : 10ms 단위로 resolution이 설정되는 timer에서 22ms 마다 반복하는 timer를 생성했을 때



① 22ms → 8ms×1 → 30ms
② 44ms → 8ms×2 → 60ms
③ 66ms → 8ms×3 → 90ms
→ 차이가 8ms씩 커짐

- solution
  1. using absolute time → time interval을 고정
     ① T = current time + 22ms
     ② timer를 22ms로
     ③ signal handler → (T - current time + 22ms) 로 interval ↓ 매번 갱신
        ⇒ T = T + 22ms
  2. using POSIX:TMR timers with absolute time
     a. timer_settime(..)의 flag ⇒ TIMER_ABSOLUTE option 설정 (Epoch 시점부터 얼마나?)
        i. timer_settime(..)의 value→it_value : real time을 나타냄
     b. 현재 시간의 absolute 값을 clock_gettime()으로 사용한 후 22ms을 추가 → T로 저장
        ↳ 현재 시간 absolute 값
     c. time T, flag TIMER_ABSOLUTE로 timer를 set
     d. timer signal handler가 T에 22ms를 더하고 만료될 때 다시 T로 timer set