Event-Driven Programming:

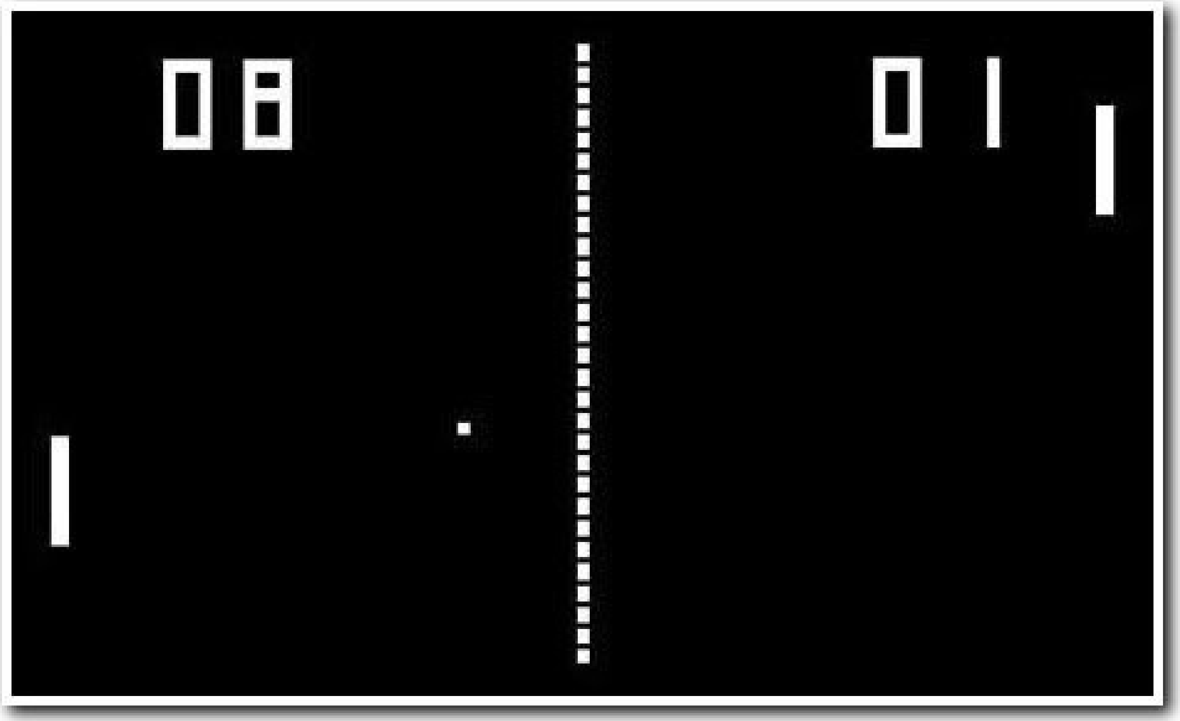
Writing a Video Game

Objectives

- **Programs driven by asynchronous events**
- The *curses* library : purpose and use
- Alarms and interval timers
  - *alarm, setitimer, getitimer*
- Reliable signal handling
  - *kill, pause, sigaction, sigprocmask*
- ~~Reentrant code,~~ critical sections
- ~~Asynchronous input~~

Video game

- PONG (one of the earliest arcade video games)



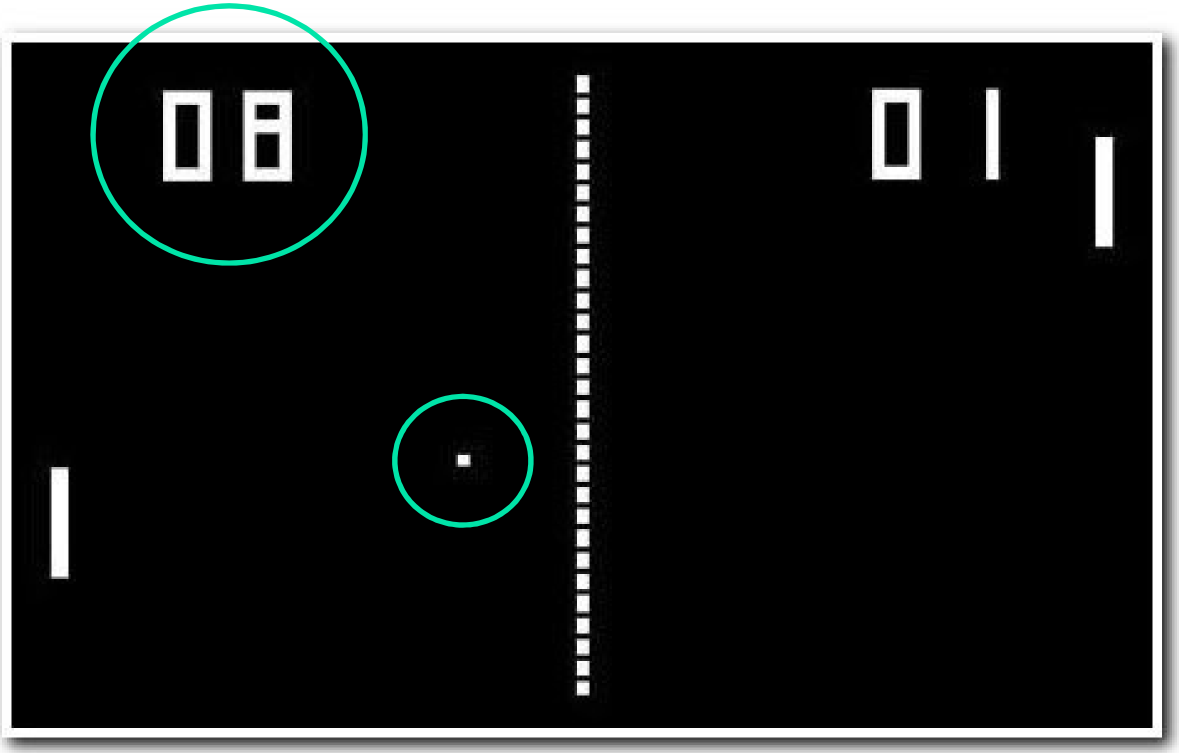
CONTROL THE PADDLE BY MOVING IT VERTICALLY TO HIT THE BALL BACK AND FORTH.
PLAY THREE STAGES, THEN YOU CAN EARN THE ITEM!

A TRIBUTE TO THE ORIGINAL ATARI PONG FROM
//SHIFT-REFRESH CONSULTING//

SPACE PROGRAMMING

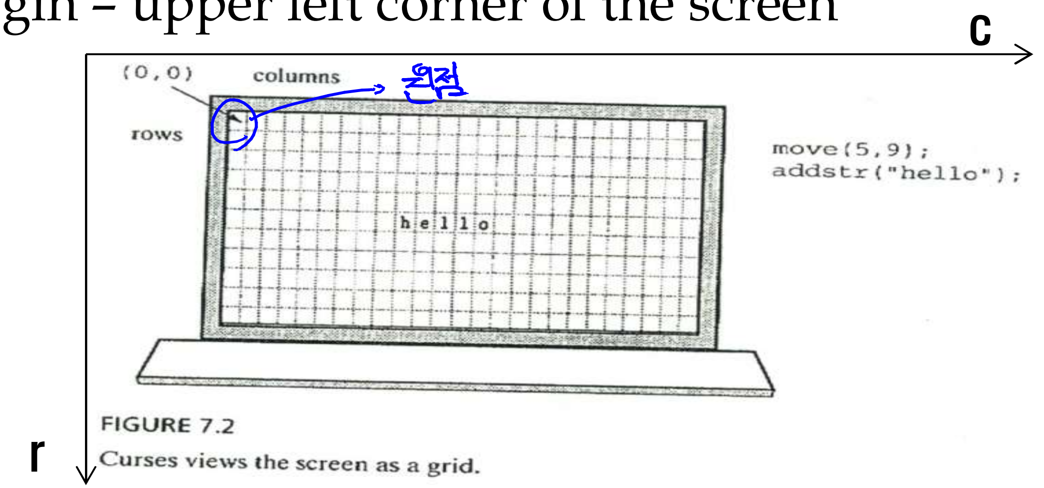
SPACE PROGRAMMING

How to draw images at specific location on the screen?



SPACE PROGRAMMING: The curses library

- Terminal control library
- The curse library is a set of functions that allow a programmer to set the position of the cursor and control the appearance of text on a terminal screen.
- The terminal screen
  - A grid of character cells
  - The origin – upper left corner of the screen



```
move(5,9);
addstr("hello");
```

FIGURE 7.2
Curses views the screen as a grid.

Basic curses functions

vi /usr/include/curses.h

	Basic curses functions
initscr()	Initializes the curses library and the tty
endwin()	Turns off curses and resets the tty
refresh()	Makes screen look the way you want
move(r(열), c(행))	Moves cursor to screen position
addstr(s)	Draws string s on the screen at current position
addch(c)	Draws char c on the screen at current position
clear()	Clears the screen
standout()	Turns on standout mode (usually reverse video)
standend()	Turns off standout mode

Hello1.c (1/1)

```
#include        <stdio.h>
#include        <curses.h>

main()
{
        initscr() ;             /* turn on curses       */

                                /* send requests        */
        clear();                        /* clear screen */
        move(10,20);                    /* row10,col20  */
        addstr("Hello, world");         /* add a string */
        move(LINES-1,0);                /* move to LL   */
             ↳ Header파일 찾아서 들어가보기 !!
        refresh();              /* update the screen    */
        getch();                /* wait for user input  */

        endwin();               /* turn off curses      */
}
```

Compile with curses library

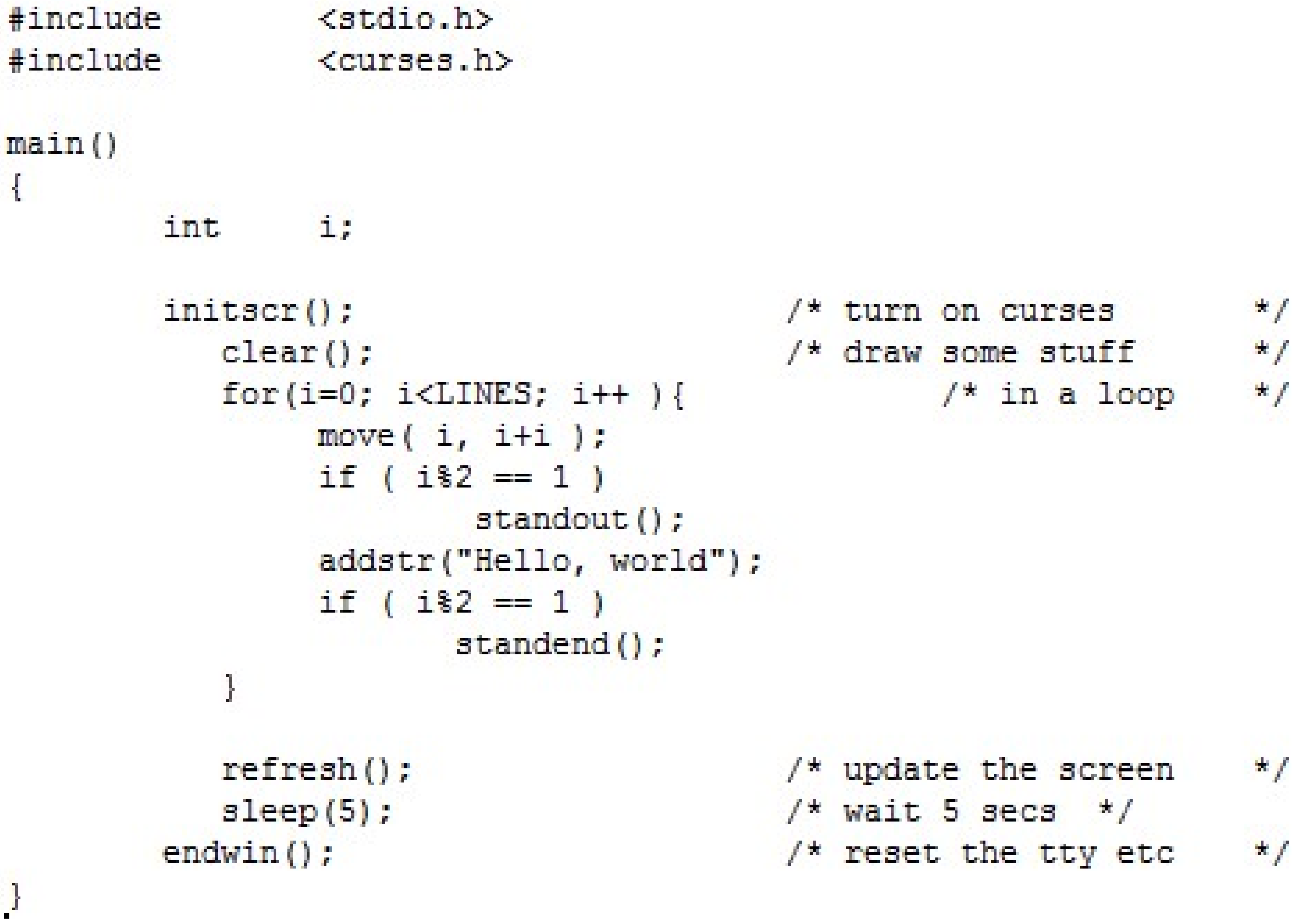
- Compiling method

```
$ gcc hello1.c –o hello1 –lcurses
$ ./hello1
```

- What "-lcurses" means?
  - -l curses (link curses library)

함수 Call 순서 기억해두기

Hello2.c (1/1)



```
#include        <stdio.h>
#include        <curses.h>

main()
{
        int     i;

        initscr();                      /* turn on curses       */
           clear();                     /* draw some stuff      */
           for(i=0; i<LINES; i++ ){             /* in a loop    */
                move( i, i+i );
                if ( i%2 == 1 )
                        standout();
                addstr("Hello, world");
                if ( i%2 == 1 )
                       standend();
           }

           refresh();                   /* update the screen    */
           sleep(5);                    /* wait 5 secs  */
        endwin();                       /* reset the tty etc    */
}
```

apt-file search curses.h apt-get update

Curses internals : virtual and real screens

- What does the *refresh* function do?
  - In Hello2.c, **comment out the refresh function** and recompile, and run the program.

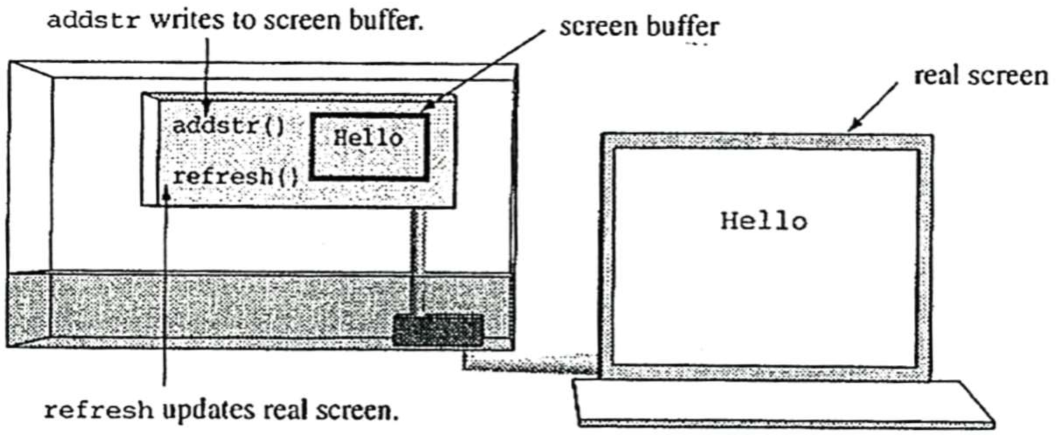


FIGURE 7.4

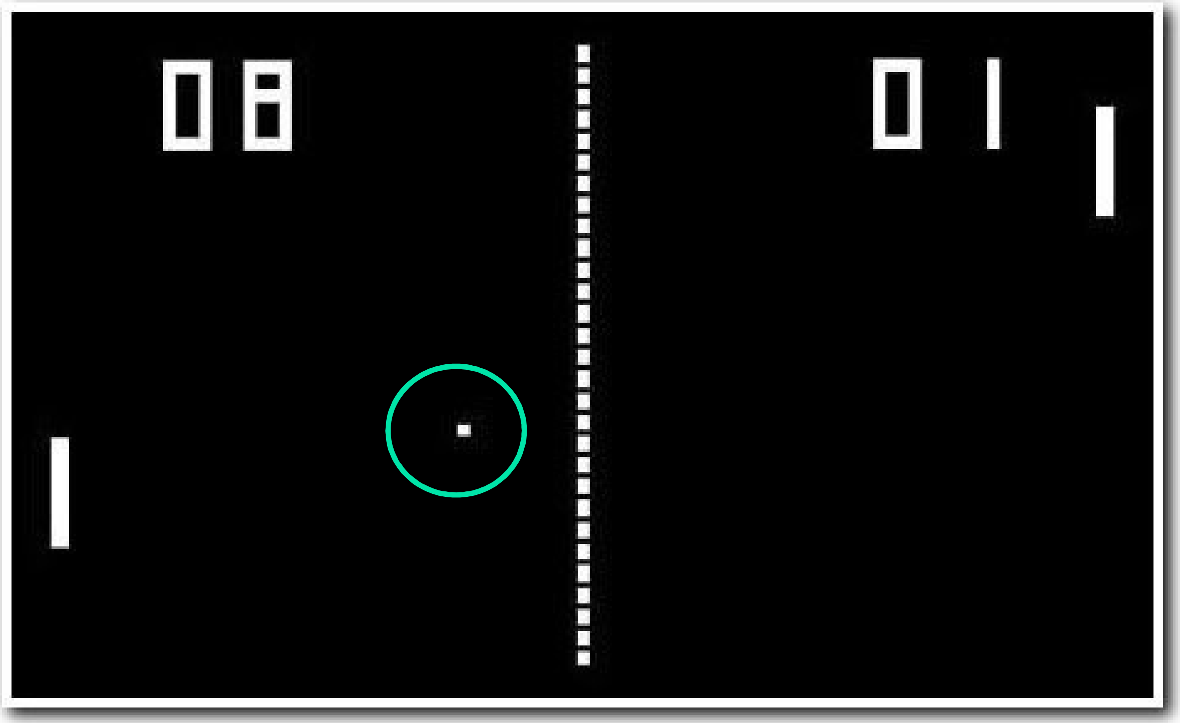
Curses keeps a copy of the real screen.

  - Compare the workspace screen to the copy of the real screen
  - Sends out through the terminal driver the characters

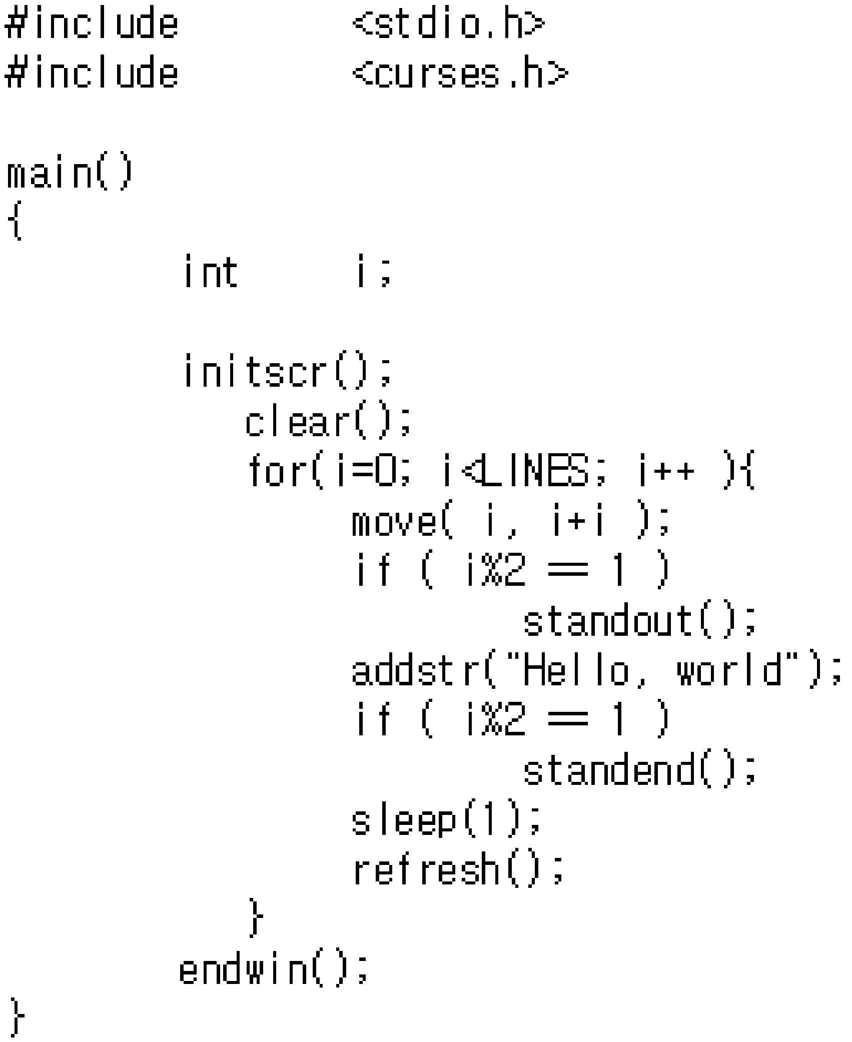
TIME HANDLING

TIME HANDLING

How to move or to show animated effects the images?



Hello3.c (Animation example 1)



```
#include        <stdio.h>
#include        <curses.h>

main()
{
        int     i;

        initscr();
           clear();
           for(i=0; i<LINES; i++ ){
                move( i, i+i );
                if ( i%2 == 1 )
                        standout();
                addstr("Hello, world");
                if ( i%2 == 1 )
                        standend();
                sleep(1);
                refresh();
           }
        endwin();
}
```

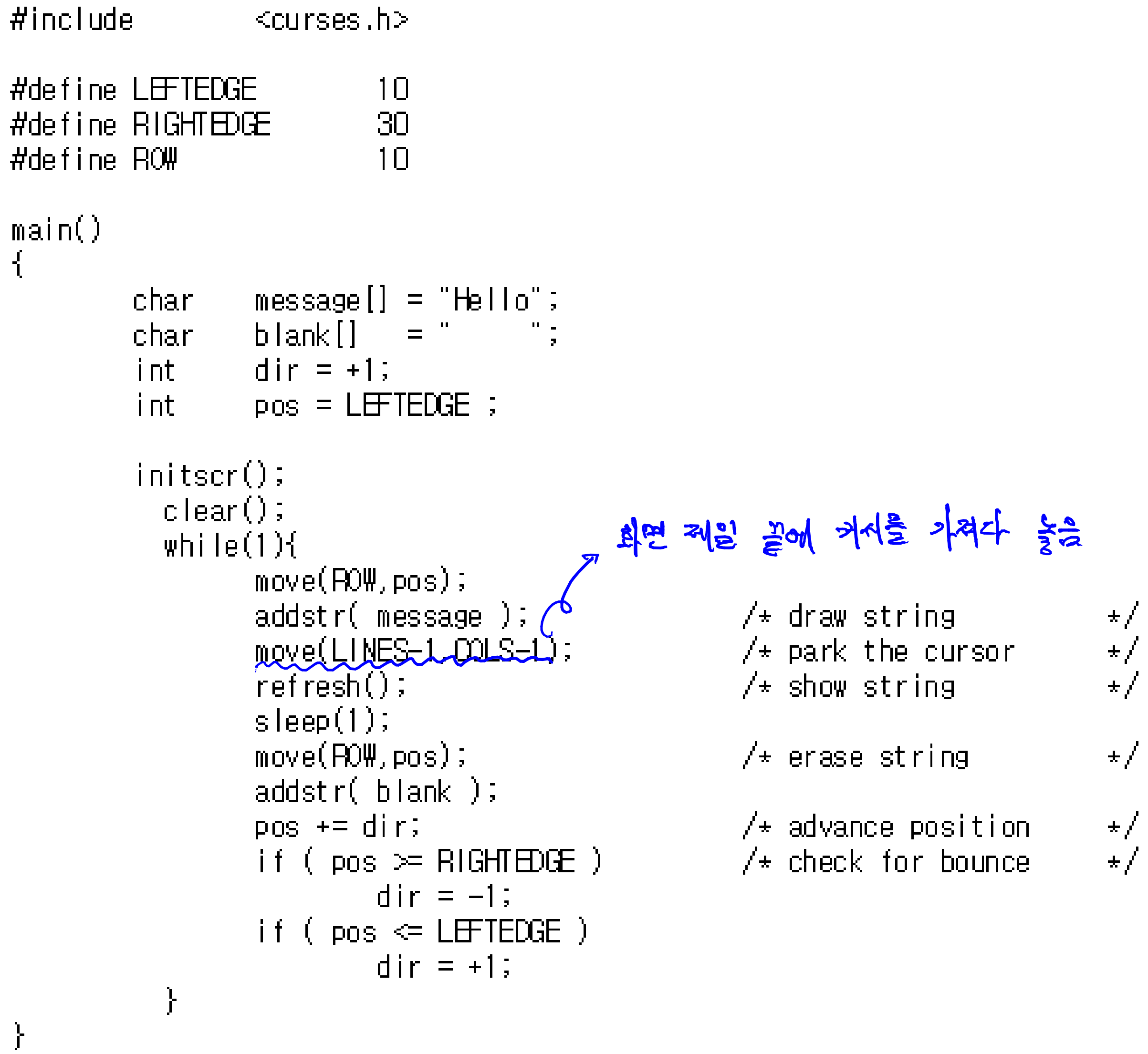
계속해서 버퍼를 지우지 않기 때문에
누적되어 쌓여갑니다.

Hello4.c (Animation example 2)

- **Modify Hello3.c**
  - You just insert **two lines**

```
Hello, world
     Hello, world
         Hello, world
```

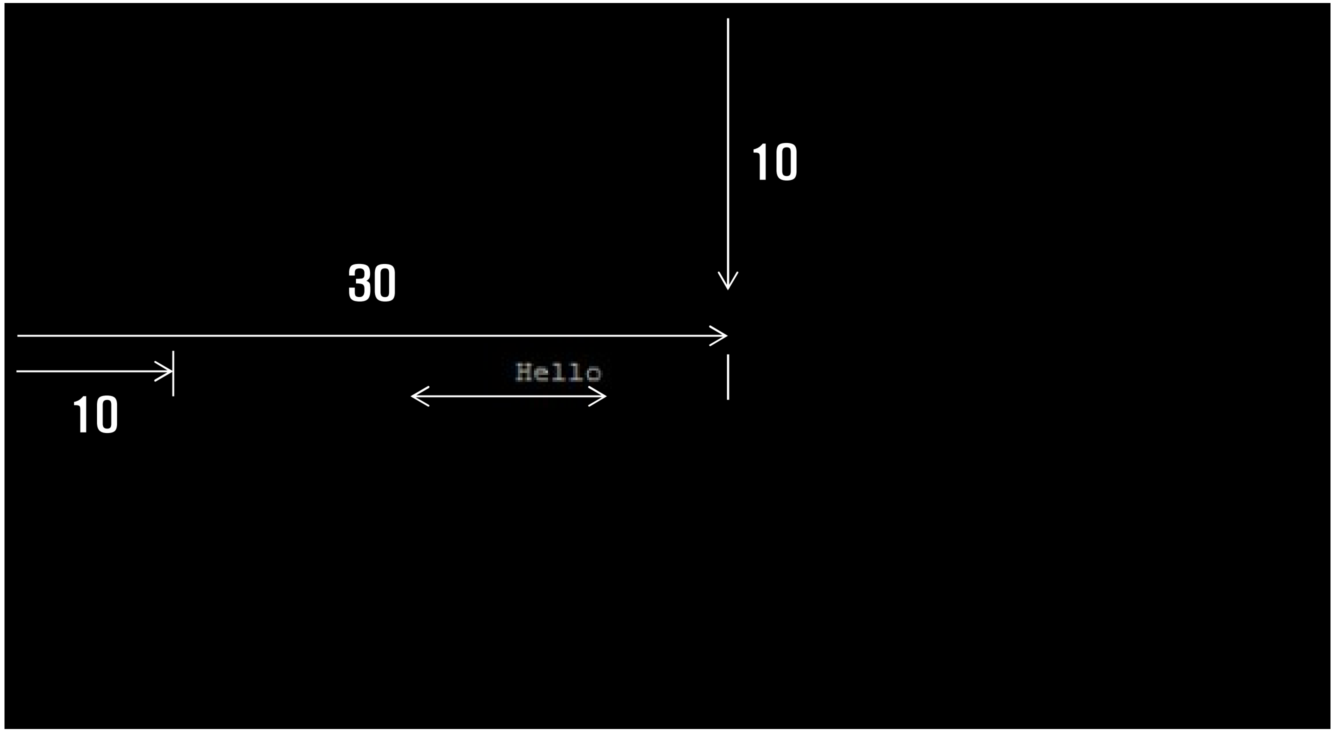
Hello5.c (Animation example 3)



```
#include        <curses.h>

#define LEFTEDGE        10
#define RIGHTEDGE       30
#define ROW             10

main()
{
        char    message[] = "Hello";
        char    blank[]   = "     ";
        int     dir = +1;
        int     pos = LEFTEDGE ;

        initscr();
          clear();
          while(1){
                move(ROW,pos);
                addstr( message );              /* draw string          */
                move(LINES-1,COLS-1);           /* park the cursor      */
                refresh();                      /* show string          */
                sleep(1);
                move(ROW,pos);                  /* erase string         */
                addstr( blank );
                pos += dir;                     /* advance position     */
                if ( pos >= RIGHTEDGE )         /* check for bounce     */
                        dir = -1;
                if ( pos <= LEFTEDGE )
                        dir = +1;
          }
}
```

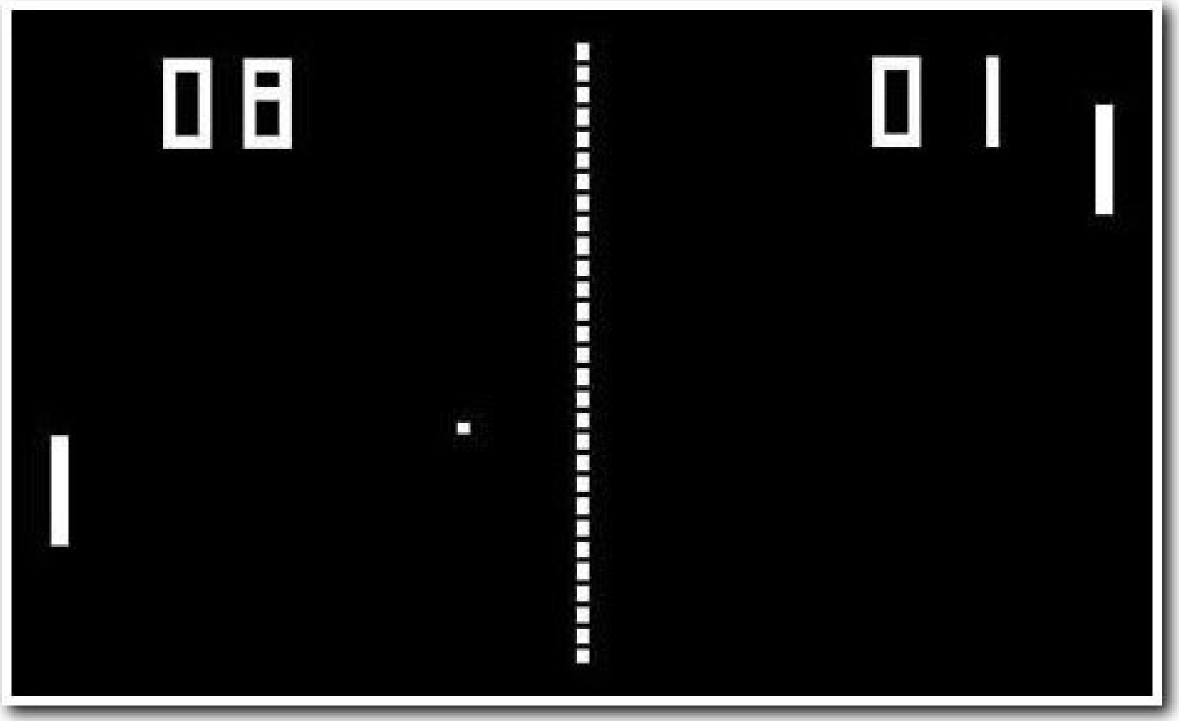
화면 제일 끝에 커서를 가져다 놓음

Hello5.c



```
Hello
```

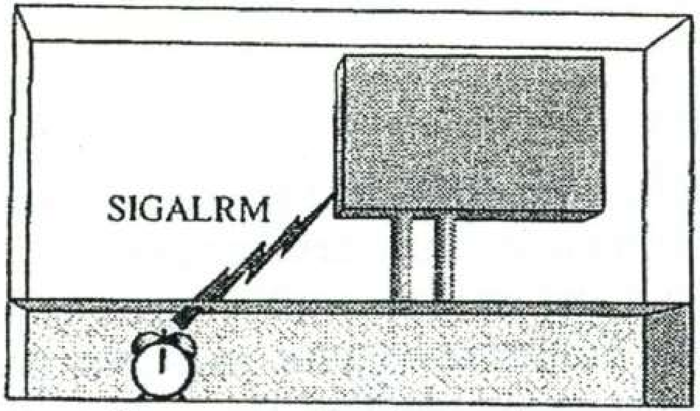
TIME HANDLING



The ball is getting faster every **10 secs**. How?

PROGAMMING WITH TIME I : ALARMS

- Adding a Delay : sleep(n)
- How sleep() Works: Using alarms in Unix
  - Set an alarm for the number of seconds you want to sleep
  - Pause until the alarm goes off



Every process has its own timer.

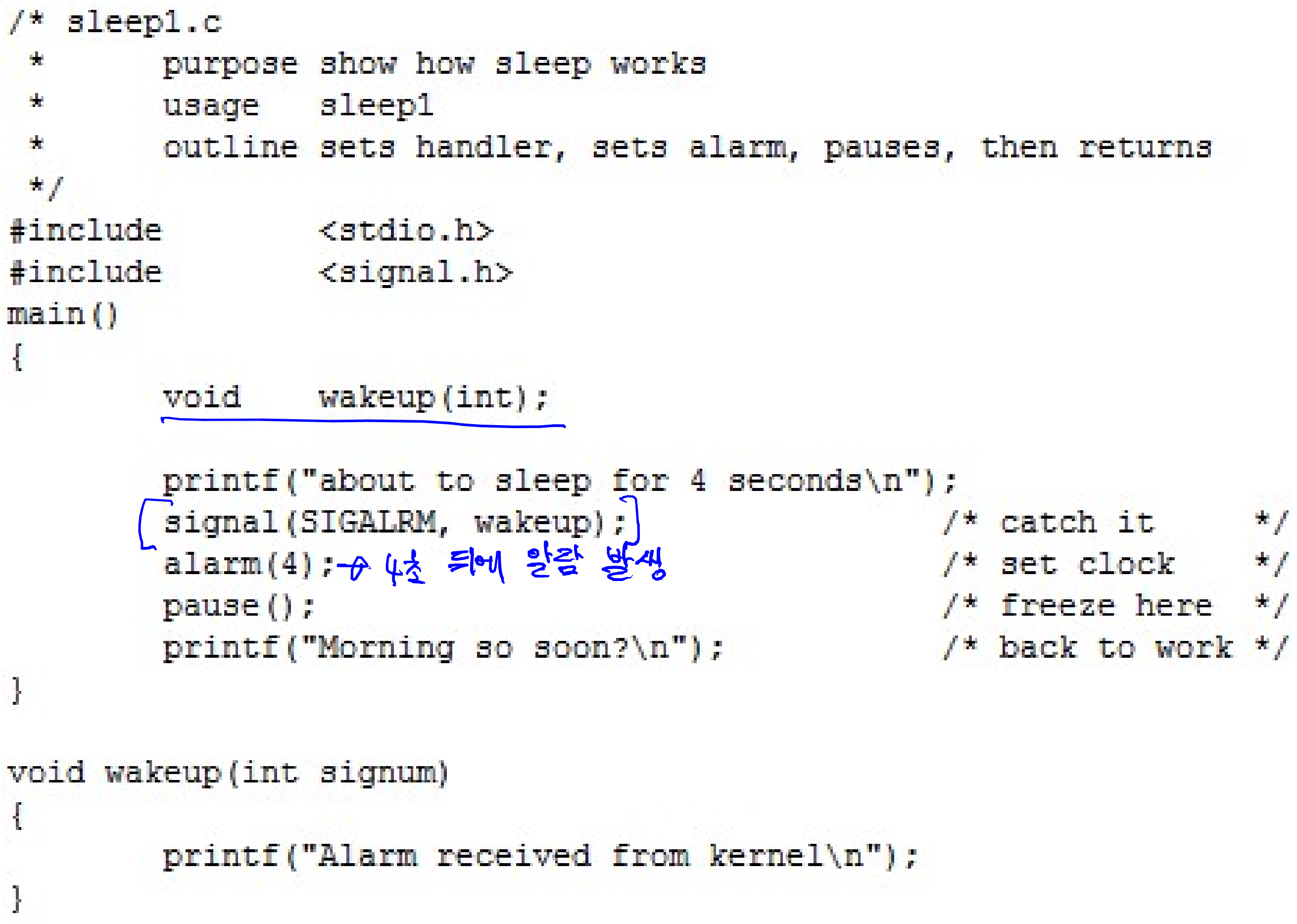
FIGURE 7.7

A process sets an alarm then suspends execution.

How the sleep function works:

- signal(SIGALRM, handler);
- alarm(n);
- pause();

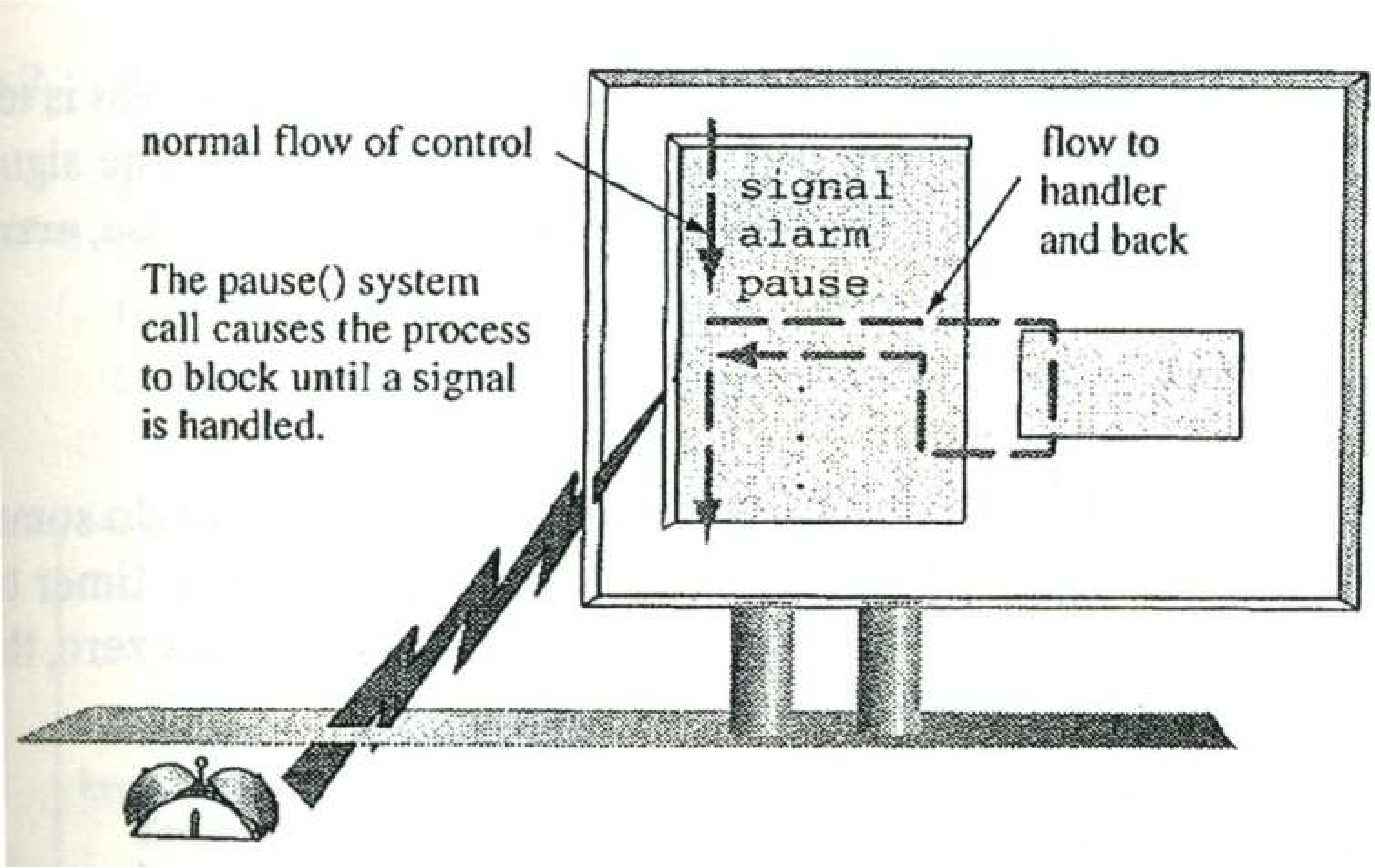
sleep1.c



```
/* sleep1.c
 *      purpose show how sleep works
 *      usage   sleep1
 *      outline sets handler, sets alarm, pauses, then returns
 */
#include        <stdio.h>
#include        <signal.h>
main()
{
        void    wakeup(int);

        printf("about to sleep for 4 seconds\n");
        signal(SIGALRM, wakeup);                /* catch it     */
        alarm(4);→ 4초 뒤에 알람 발생            /* set clock    */
        pause();                                /* freeze here  */
        printf("Morning so soon?\n");           /* back to work */
}

void wakeup(int signum)
{
        printf("Alarm received from kernel\n");
}
```

PROGAMMING WITH TIME I : ALARMS

- How sleep() Works: Using Alarms in Unix



normal flow of control
The pause() system call causes the process to block until a signal is handled.
signal
alarm
pause
flow to handler and back

PROGAMMING WITH TIME I : ALARMS

alarm	
PURPOSE	Set an alarm timer for delivery of a signal
INCLUDE	#include<unistd.h>
USAGE	unsigned old = alarm(unsigned seconds)
ARGS	seconds - how long to wait
RETURNS	-1 if error old time left on timer

pause	
PURPOSE	Wait for signal
INCLUDE	#include <unistd.h>
USAGE	Result = pause()
ARGS	No args
RETURNS	-1 always

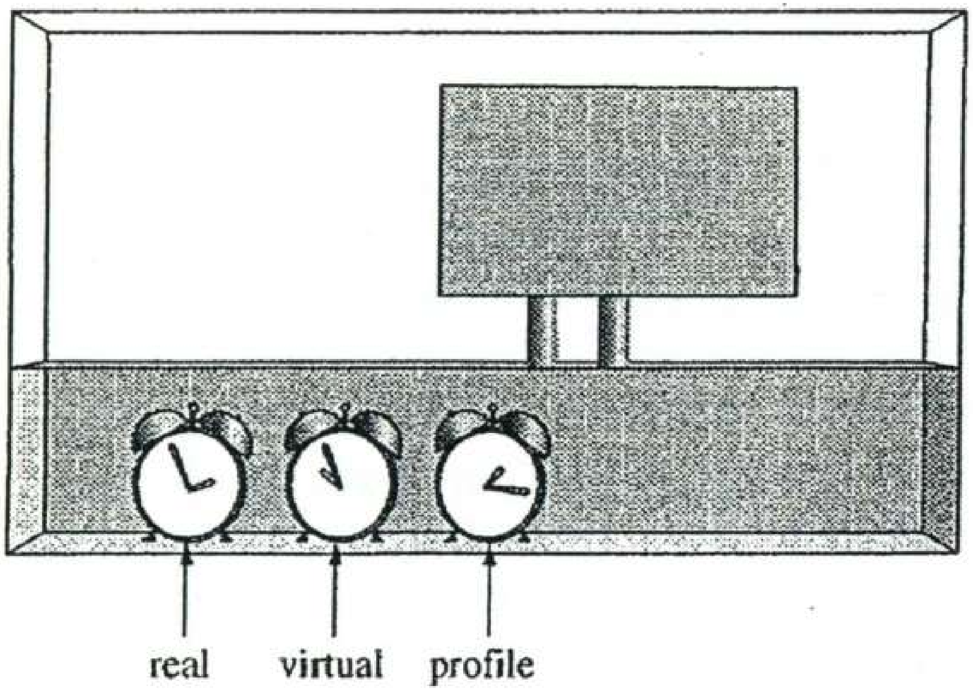
PROGAMMING WITH TIME 2: INTERVAL TIMERS

- The ball is getting faster every **10.5 secs**.
  - For a finer delay : usleep(n)
  - usleep(n)   // suspends the current process for *n* microseconds

- Taxi meter device
  - The basic fare is 1,000 won for 2 mins. (initial)
  - It increases 100 won every 30 secs. (repeat)
  - Need to set interval times

PROGAMMING WITH TIME 2: INTERVAL TIMERS

Each process has three timers.

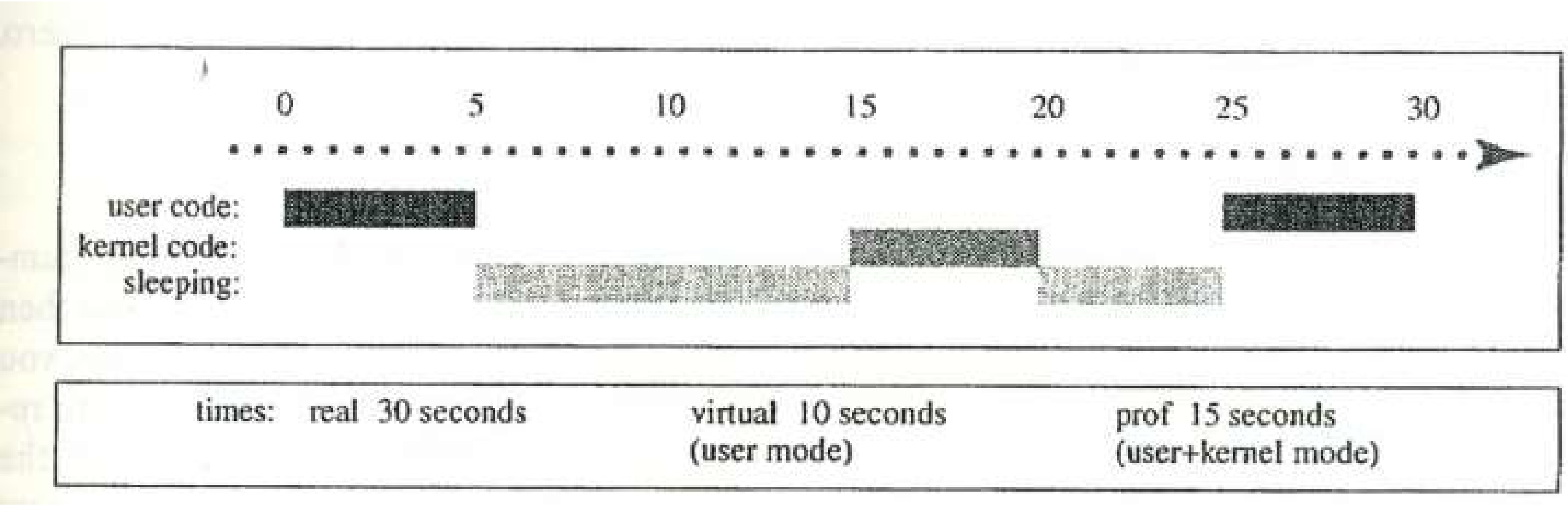


Every process has three timers.

Each timer has two settings: the time until the first alarm and the interval between repeating alarms.

PROGAMMING WITH TIME 2: INTERVAL TIMERS

- Three Kinds of Timers : Real, Process, Profile



times: real 30 seconds virtual 10 seconds (user mode) prof 15 seconds (user+kernel mode)

PROGAMMING WITH TIME 2: INTERVAL TIMERS

- The kernel provides timers to measure each of these types
  - ITIMER_REAL
    - Ticks in real time
    - Send *SIGALRM*
  - ITIMER_VIRTUAL
    - Only ticks when the process runs in user mode (Football game)
    - Send *SIGVTALRM*
  - ITIMER_PROF
    - Ticks when the process runs in user mode and when the kernel is running system calls made by this process
    - Send *SIGPROF*

PROGAMMING WITH TIME 2: INTERVAL TIMERS

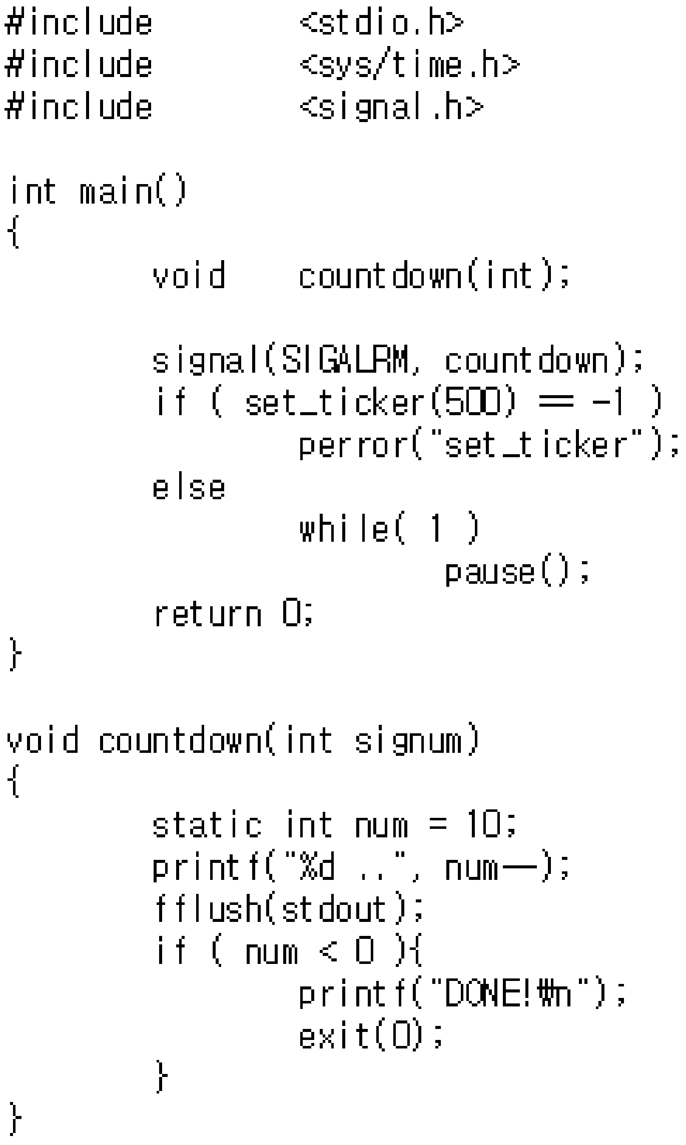
- Programming with the Interval Timers
  - 1. Decide on an initial interval and a repeating interval
  - 2. Set values in a struct itimerval
    - Initial interval and repeating interval
  - 3. Pass the structure to the timer by calling setitimer

PROGAMMING WITH TIME 2: INTERVAL TIMERS

- Details of Data Structures

```
struct itimerval
{
        struct timeval it_value;           /* time to next timer expiration */
        struct timeval it_interval;         /* reload it_value with this  */
};

struct itimerval
```

ticker_demo.c (1/2)



```
#include        <stdio.h>
#include        <sys/time.h>
#include        <signal.h>

int main()
{
        void    countdown(int);

        signal(SIGALRM, countdown);
        if ( set_ticker(500) == -1 )
                perror("set_ticker");
        else
                while( 1 )
                        pause();
        return 0;
}

void countdown(int signum)
{
        static int num = 10;
        printf("%d ..", num--);
        fflush(stdout);
        if ( num < 0 ){
                printf("DONE!₩n");
                exit(0);
        }
}
```

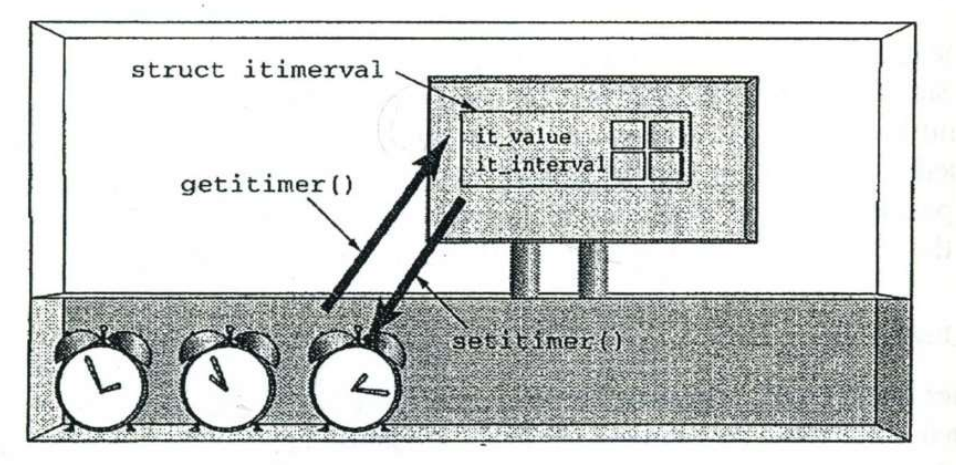


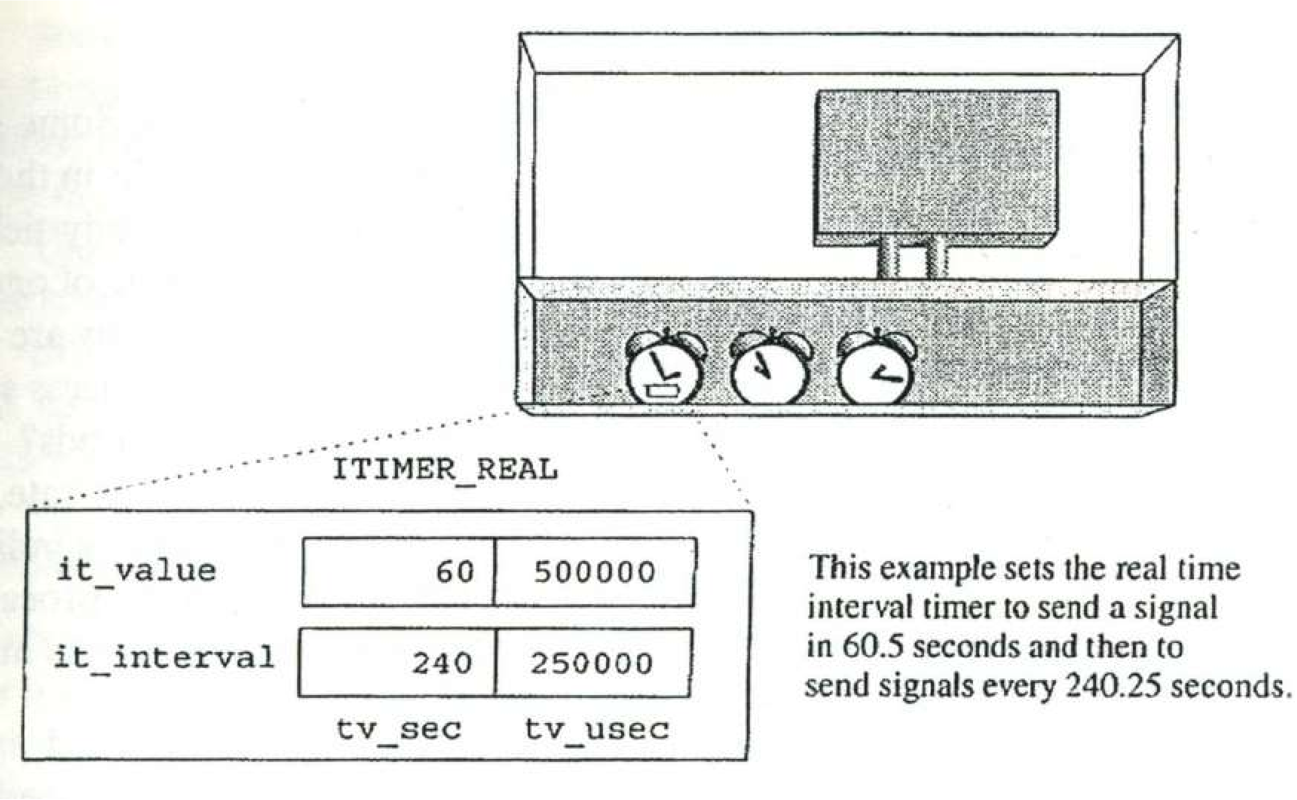
FIGURE 7.11

Reading and writing timer settings.

ticker_demo.c (2/2)

```
int set_ticker( int n_msecs )
{
        struct itimerval new_timeset;
        long    n_sec, n_usecs;

        n_sec = n_msecs / 1000 ;                /* int part     */
        n_usecs = ( n_msecs % 1000 ) * 1000L ;  /* remainder    */

        new_timeset.it_interval.tv_sec  = n_sec;        /* set reload       */
        new_timeset.it_interval.tv_usec = n_usecs;      /* new ticker value */
        new_timeset.it_value.tv_sec     = n_sec  ;      /* store this       */
        new_timeset.it_value.tv_usec    = n_usecs ;     /* and this         */

        return setitimer(ITIMER_REAL, &new_timeset, NULL);
}
```

PROGAMMING WITH TIME 2: INTERVAL TIMERS



ITIMER_REAL

it_value	60	500000
it_interval	240	250000
	tv_sec	tv_usec

This example sets the real time interval timer to send a signal in 60.5 seconds and then to send signals every 240.25 seconds.

getitimer	
PURPOSE	Get value of interval timer
INCLUDE	#include<sys/time.h>

USAGE	result = getitimer(int which,
	struct itimerval *val);

ARGS	which timer being read or set val pointer to current settings
RETURNS	-1 on error 0 on success

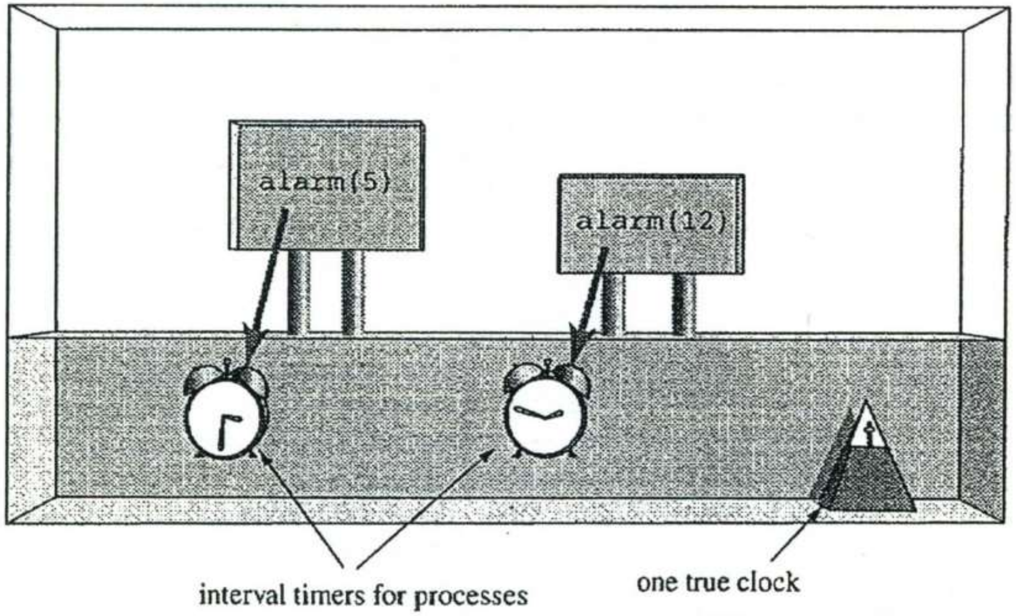
setitimer	
PURPOSE	Set value of interval timer
INCLUDE	#include<sys/time.h>

USAGE	result = setitimer(int which,
	const struct itimerval *newval,
	struct itimerval *oldval);

ARGS	which timer being read or set newval pointer to settings to be installed oldval pointer to settings being replaced
RETURNS	-1 on error 0 on success

PROGAMMING WITH TIME 2: INTERVAL TIMERS

- How Many Clocks Does the Computer Have?
  - Every process on the system have three separate clock?
    - ps –A

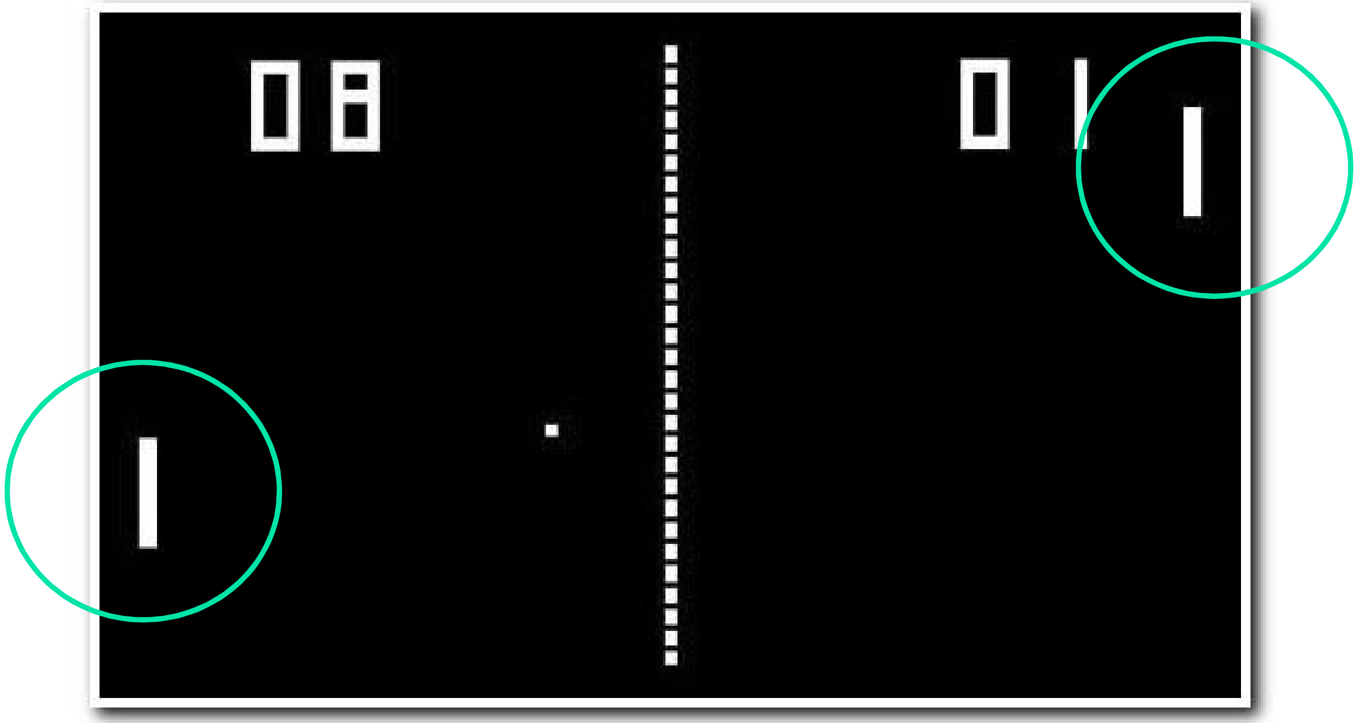


Each process sets its private timer by calling `alarm`. The kernel updates all process timers at each signal from its clock.

SIGNAL HANDLING

SIGNAL HANDLING

How to move the bar when users type the keyboard?

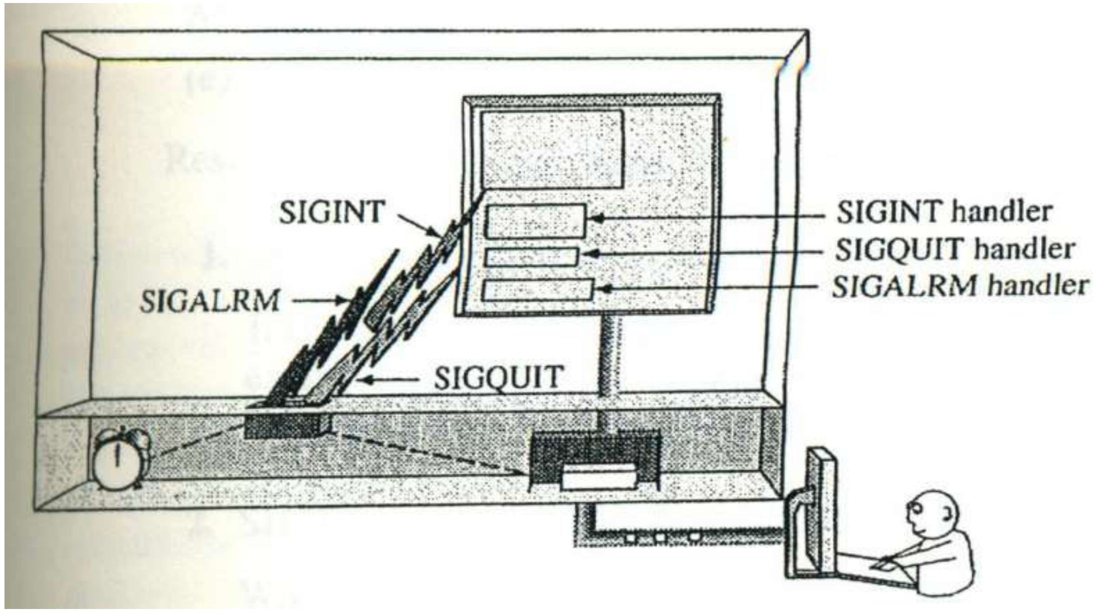


SIGNAL HANDLING 1: USING *signal*

- Old-Style Signal Handling
  - default action
    - signal(SIGALRM, SIG_DFL)
  - ignore the signal
    - signal(SIGALRM, SIG_IGN)
  - invoke a function
    - signal(SIGALRM, handler)

SIGNAL HANDLING 1: USING *signal*

- The original signal model **works fine if only one signal arrives**
  - What happens when **multiple signals arrive**?



sigdemo3.c (1/2)

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

#define INPUTLEN        100

main(int ac, char *av[])
{
        void    inthandler(int);
        void    quithandler(int);
        char    input[INPUTLEN];
        int     nchars;

        signal( SIGINT,  inthandler );          /* set handler */
        signal( SIGQUIT, quithandler );         /* set handler */

        do {
                printf("₩nType a message₩n");
                nchars = read(0, input, (INPUTLEN-1));
                if ( nchars == -1 )
                        perror("read returned an error");
                else {
                        input[nchars] = '₩0';
                        printf("You typed: %s", input);
                }
        }
        while( strncmp( input , "quit" , 4 ) != 0 );
}
```

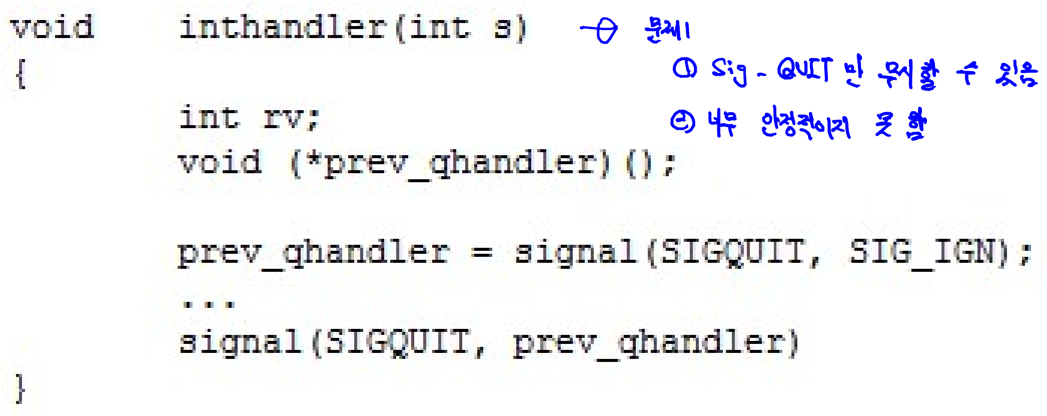
sigdemo3.c (2/2)

```
void inthandler(int s)
{
        printf(" Received signal %d .. waiting₩n", s );
        sleep(2);
        printf("  Leaving inthandler ₩n");
}

void quithandler(int s)
{
        printf(" Received signal %d .. waiting₩n", s );
        sleep(3);
        printf("  Leaving quithandler ₩n");
}
```

- Answers those questions
  - What happened if a SIG_INT arrived while the process is in the SIG_QUIT handler? (^\ ➔ ^C ➔ ^\)
  - What happened if a second SIG_INT arrived while the process is still in the SIG_INT handler? Or a third SIG_INT? (^C ➔ ^C ➔ ^C)
  - What happens if a signal arrives while the program is blocking on input? (qui ➔ ^C ➔ t Enter)
  - Is the handler disabled after each use? (^C ➔ (handler) ➔ ^C)

SIGNAL HANDLING 1: USING *signal*

- The original signal system has two other weaknesses.
  - Don’t know why the signal was sent
    - Tells the handler which signal invoked it (using signum)
    - However, it does not tell the handler **why the signal was generated**
  - Cannot **safely** block other signals while in a handler
    - Want to ignore SIGQUIT when it responds to SIGINT
    - Do not want to ignore SIGQUIT, want to block it until SIGINT processed



```
void      inthandler(int s)
{
          int rv;
          void (*prev_qhandler)();

          prev_qhandler = signal(SIGQUIT, SIG_IGN);
          ...
          signal(SIGQUIT, prev_qhandler)
}
```

-θ 문제!

① Sig-QUIT만 무시할 수 있음

② 너무 안정적이지 못 함

SIGNAL HANDLING 2: *sigaction*

- *sigaction* is the POSIX replacement for *signal*
  - Specify which signal to handle and how you want to handle that signal

sigaction	
PURPOSE	Specify handling for a signal
INCLUDE	#include<signal.h>
USAGE	int res = sigaction(int signum, const struct sigaction *action, struct sigaction *prevaction);
ARGS	signum signal to handle action pointer to struct describing action prevaction pointer to struct to receive old action
RETURNS	-1 on error 0 on success

SIGNAL HANDLING 2: *sigaction*

- Customized Signal Handling

```
 struct sigaction
{
          /* use only one of these two */
          void (*sa_handler)(int);
          void (*sa_sigaction)(int, siginfo_t *, void *);

          sigset_t sa_mask;
          int sa_flags;           /* enable various behaviors */
}
```

Using an old-style handler	Using a new-style handler

struct sigaction action;
action.sa_handler = handler_old;
	struct sigaction action;
	action.**sa_sigaction** = handler_new;

How do you tell the kernel you are using the new-style handler?

➔ **Set SA_SIGINFO bit in sa_flags**

SIGNAL HANDLING 2: *sigaction*

- sa_flags
  - A set of bits that control how the handler does

Flag	Meaning
SA_RESETHAND	Reset the handler when invoked. This enables mousetrap mode.
SA_NODEFER	Turn off automatic blocking of a signal while it is being handled. This allows recursive calls to a signal handler.
SA_RESTART	Restart, rather than return, system calls on slow devices and similar system calls. This enables BSD mode.
SA_SIGINFO	Use the value in sa_sigaction for the Handler function. If this bit is not set, use the value in sa_handler.
…	…

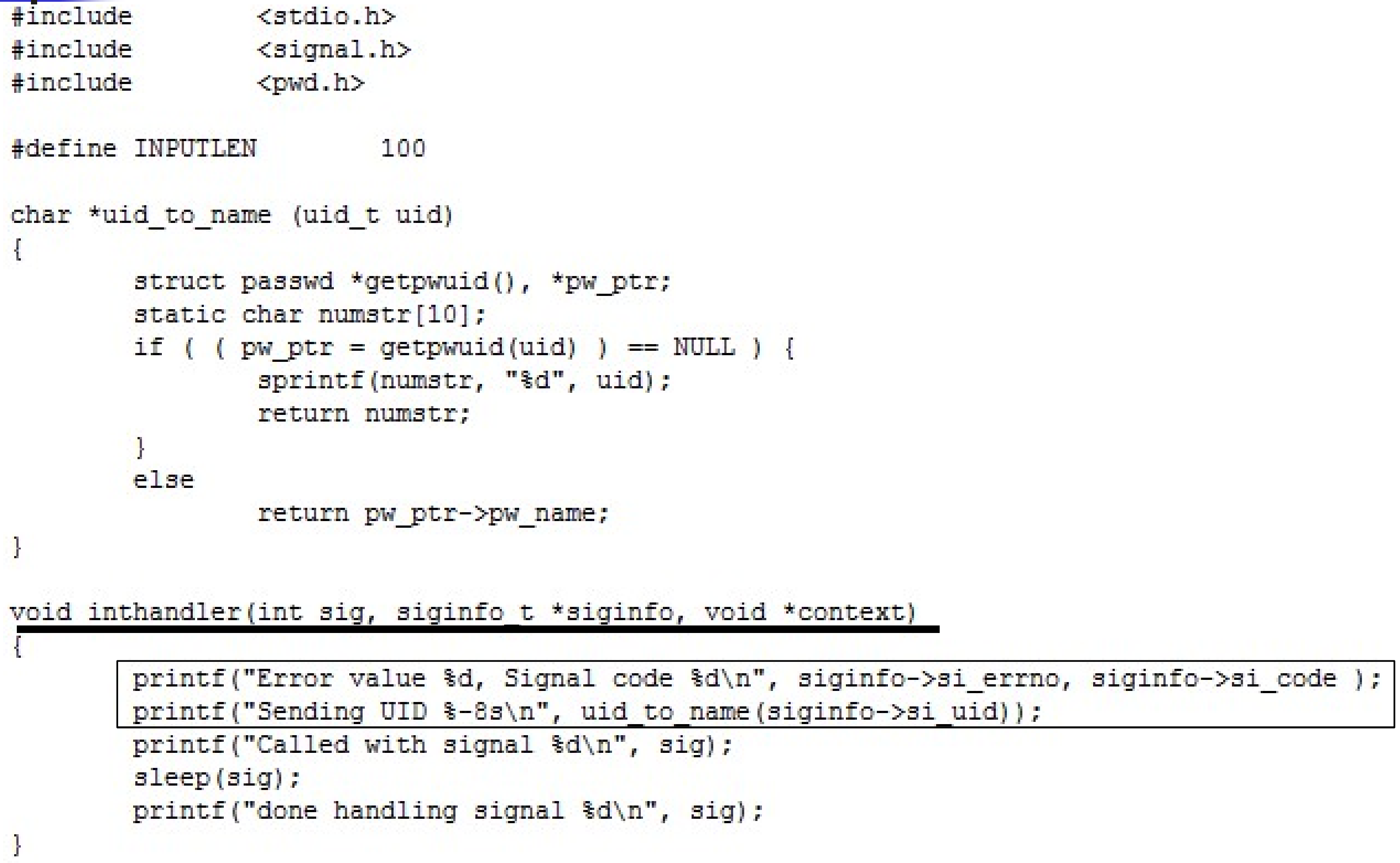
- sa_mask
  - Decide if we want to block any other signal while in the handler

sigactdemo1.c (old version)

```
#include        <stdio.h>
#include        <signal.h>
#define INPUTLEN        100

main()
{
        struct sigaction newhandler;            /* new settings         */
        sigset_t         blocked;               /* set of blocked sigs */
        void             inthandler();          /* the handler          */
        char             x[INPUTLEN];

        /* load these two members first */
        newhandler.sa_handler = inthandler;       /* handler function    */
        newhandler.sa_flags = SA_RESETHAND | SA_RESTART;  /* options     */

        /* then build the list of blocked signals */
        sigemptyset(&blocked);                  /* clear all bits       */
        sigaddset(&blocked, SIGQUIT);           /* add SIGQUIT to list */

        newhandler.sa_mask = blocked;           /* store blockmask      */

        if ( sigaction(SIGINT, &newhandler, NULL) == -1 )
                perror("sigaction");
        else
                while( 1 ){
                        fgets(x, INPUTLEN, stdin);
                        printf("input: %s", x);
                }
}
```

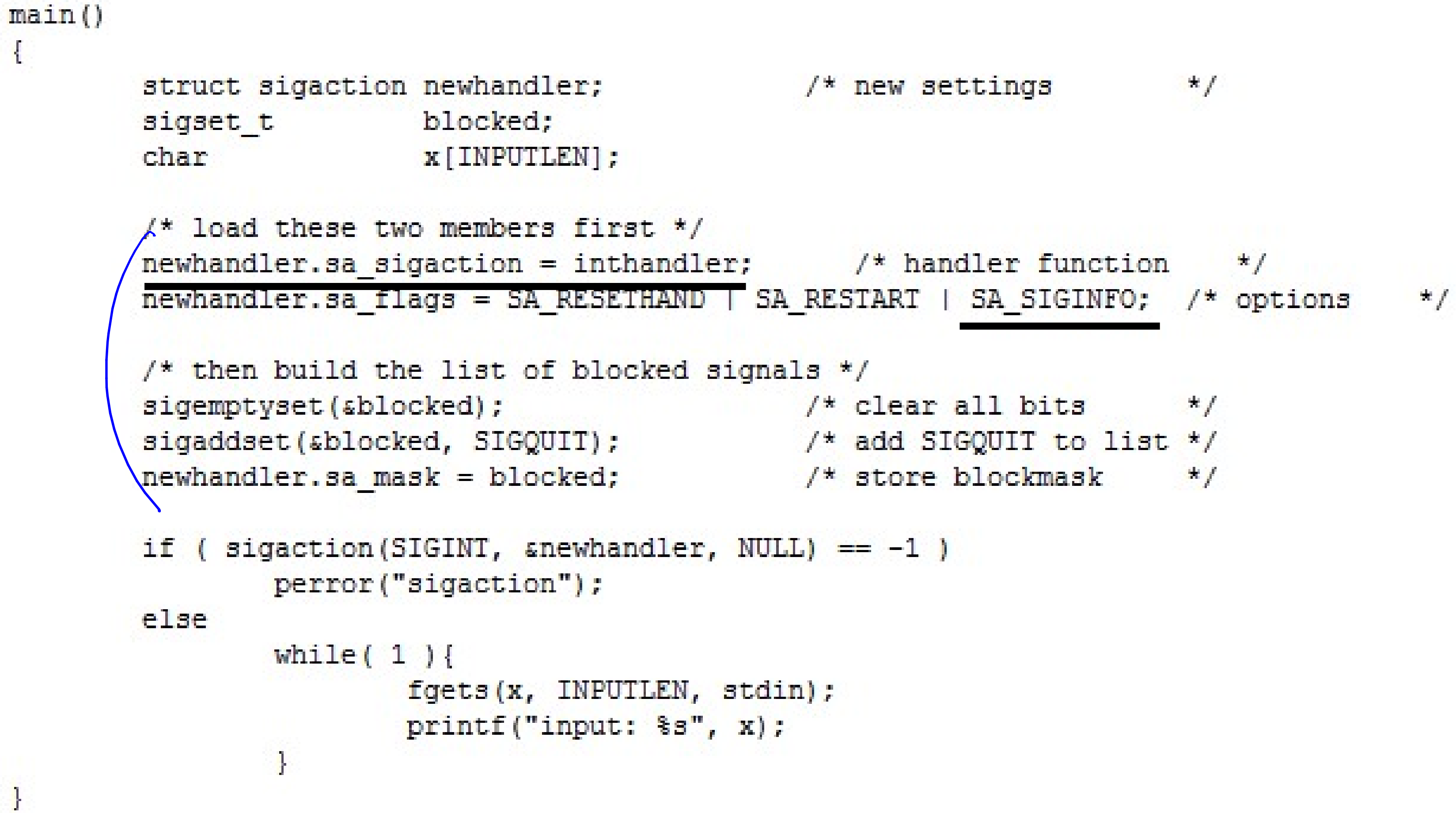
```
void inthandler(int s)
{
        printf("Called with signal %d₩n", s);
        sleep(s);
        printf("done handling signal %d₩n", s);
}
```

sigactdemo2.c (new version) (1/2)



```
#include        <stdio.h>
#include        <signal.h>
#include        <pwd.h>

#define INPUTLEN        100

char *uid_to_name (uid_t uid)
{
        struct passwd *getpwuid(), *pw_ptr;
        static char numstr[10];
        if ( ( pw_ptr = getpwuid(uid) ) == NULL ) {
                sprintf(numstr, "%d", uid);
                return numstr;
        }
        else
                return pw_ptr->pw_name;
}

void inthandler(int sig, siginfo_t *siginfo, void *context)
{
        printf("Error value %d, Signal code %d\n", siginfo->si_errno, siginfo->si_code );
        printf("Sending UID %-8s\n", uid_to_name(siginfo->si_uid));
        printf("Called with signal %d\n", sig);
        sleep(sig);
        printf("done handling signal %d\n", sig);
}
```

sigactdemo2.c (new version) (2/2)



```
main()
{
        struct sigaction newhandler;            /* new settings        */
        sigset_t         blocked;
        char             x[INPUTLEN];

        /* load these two members first */
        newhandler.sa_sigaction = inthandler;      /* handler function    */
        newhandler.sa_flags = SA_RESETHAND | SA_RESTART | SA_SIGINFO;  /* options     */

        /* then build the list of blocked signals */
        sigemptyset(&blocked);                  /* clear all bits      */
        sigaddset(&blocked, SIGQUIT);           /* add SIGQUIT to list */
        newhandler.sa_mask = blocked;           /* store blockmask     */

        if ( sigaction(SIGINT, &newhandler, NULL) == -1 )
                perror("sigaction");
        else
                while( 1 ){
                        fgets(x, INPUTLEN, stdin);
                        printf("input: %s", x);
                }
}
```

PROTECTING DATA FROM CORRUPTION

- Critical sections
  - A section of code (that modifies a data structure) is called a critical section if interruptions to that section of code can produce incomplete or damaged data.
  - When you program with signals, you must determine which parts of your code are critical sections and arrange to protect those sections.

- The simplest way to protect critical sections
  - Block or ignore signals that call handlers (that use or change the data)

Blocking Signals: sigprocmask and sigsetops

- Blocking signals in a signal handler
  - Set the ***sa_mask*** member

- Blocking signals for a process
  - A process has a set of signals it is blocking (signal mask).
  - Modify that set of blocked signal using *sigprocmask*

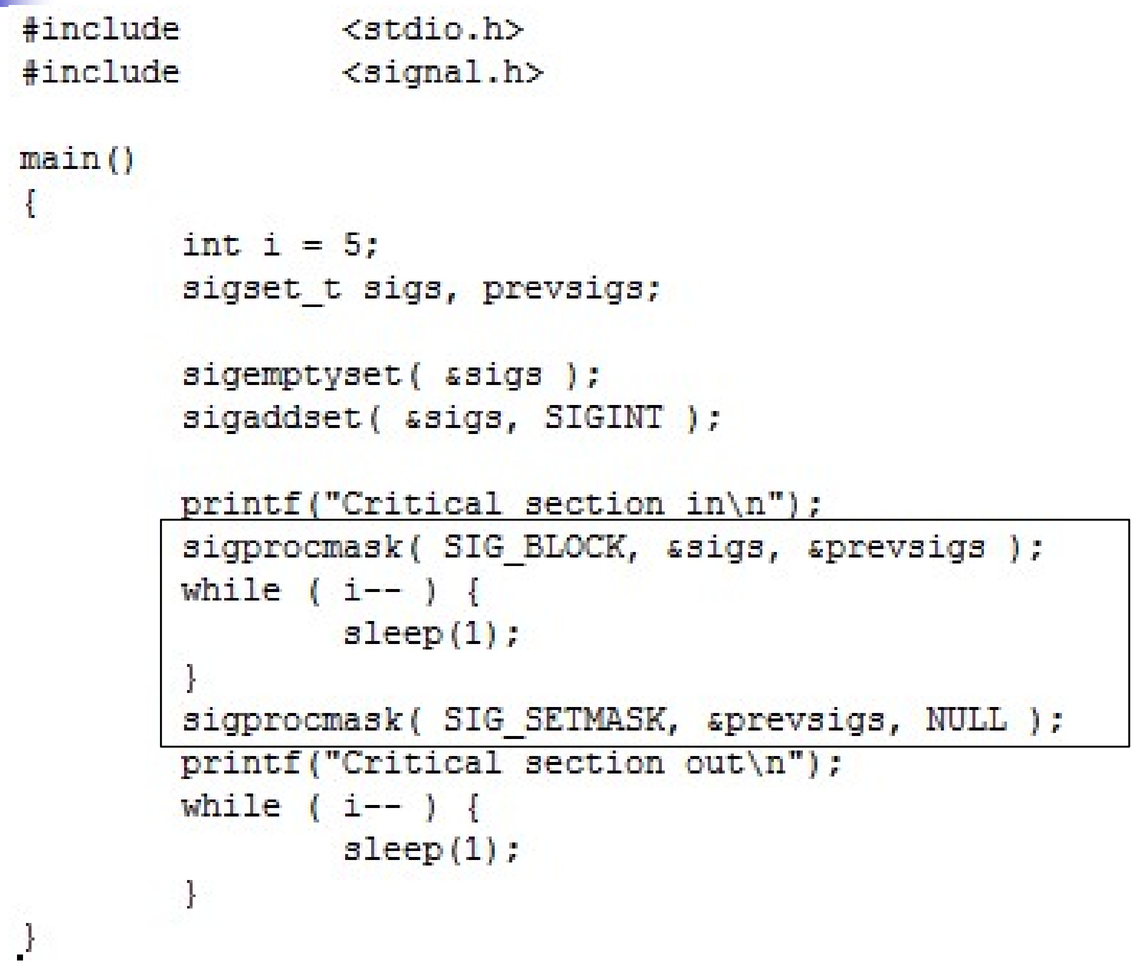
How to modify the signal mask?

sigprocmask	
PURPOSE	Modify current signal mask
INCLUDE	#include<signal.h>
USAGE	int res = sigprocmask(int how, const sigset_t *sigs sigset_t *prev);
ARGS	how how to modify the signal mask (SIG_BLOCK, SIG_UNBLOCK, SIG_SET) sigs pointer to list of signals to use prev pointer to list of previous signal mask (or NULL)
RETURNS	-1 on error 0 on success

Building Signal Sets with *sigsetops (signal set operations)*

- sigset_t – abstract set of signals that has methods for adding and removing signals
  - sigemptyset(sigset_t * setp)
    - Clear all signals from the list pointed to by setp
  - sigfillset(sigset_t * setp)
    - Add all signals to the list pointed to by setp
  - sigaddset(sigset_t * setp,int signum)
    - Add signum to the set pointed by setp
  - sigdelset(sigset_t * setp, int signum)
    - Remove signum from the set pointed to by setp

sigprocmask.c



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

main()
{
        int i = 5;
        sigset_t sigs, prevsigs;

        sigemptyset( &sigs );
        sigaddset( &sigs, SIGINT );

        printf("Critical section in\n");
        sigprocmask( SIG_BLOCK, &sigs, &prevsigs );
        while ( i-- ) {
                sleep(1);
        }
        sigprocmask( SIG_SETMASK, &prevsigs, NULL );
        printf("Critical section out\n");
        while ( i-- ) {
                sleep(1);
        }
}
```

Kill: Sending signals from a process

- Signals arise from interval timers, from the terminal driver, from the kernel, and from processes.
- A process sends a signal to another process
  - Using **kill** system call

- The process sending the signal must **have the same user ID** as the target process, or the sending process must be **owned by the superuser**
- Used at interprocess communication

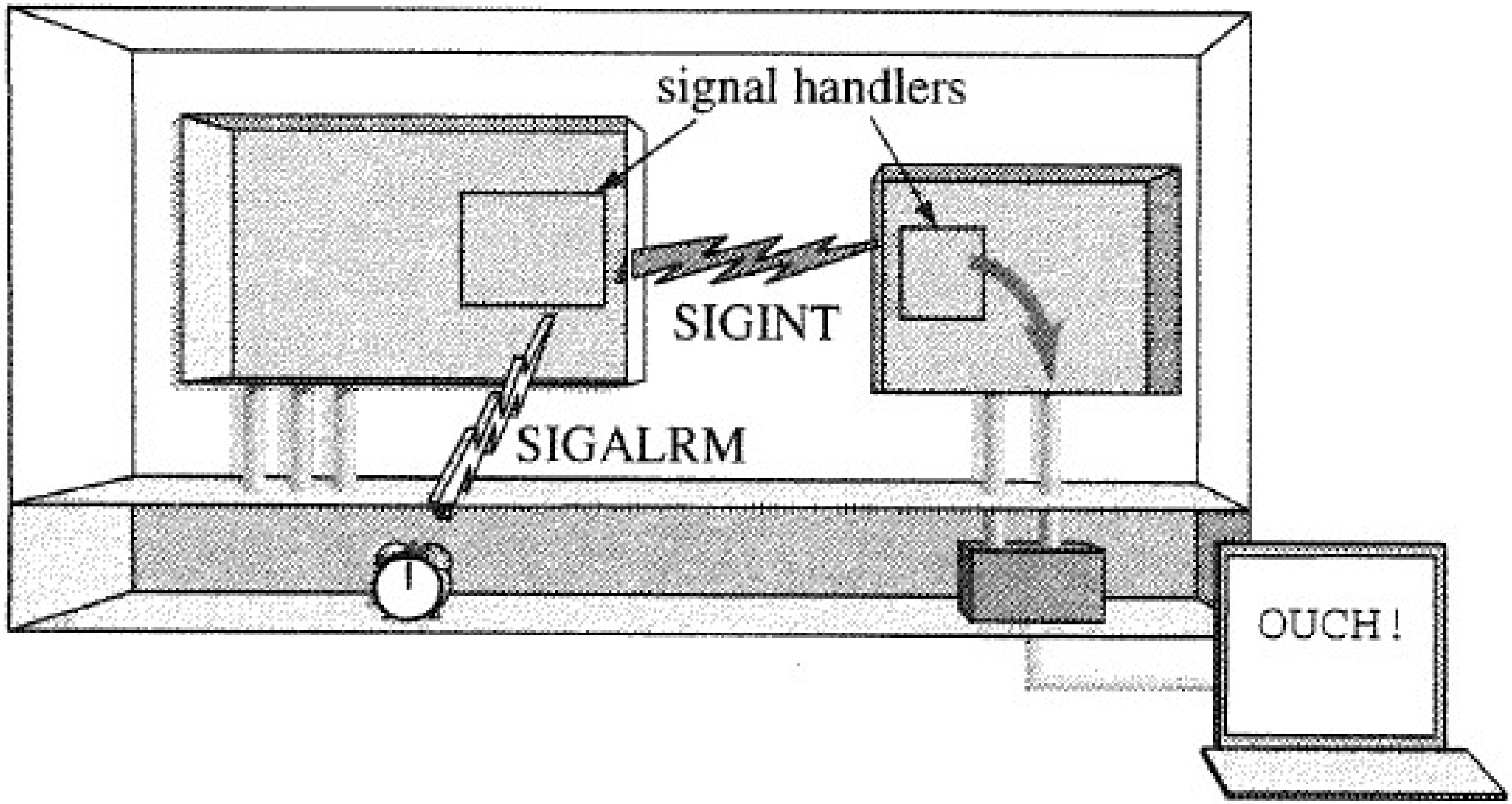
Kill: Sending signals from a process

kill		
PURPOSE	Send a signal to a process	

INCLUDE	#include <sys/types.h>
	#include <signal.h>	

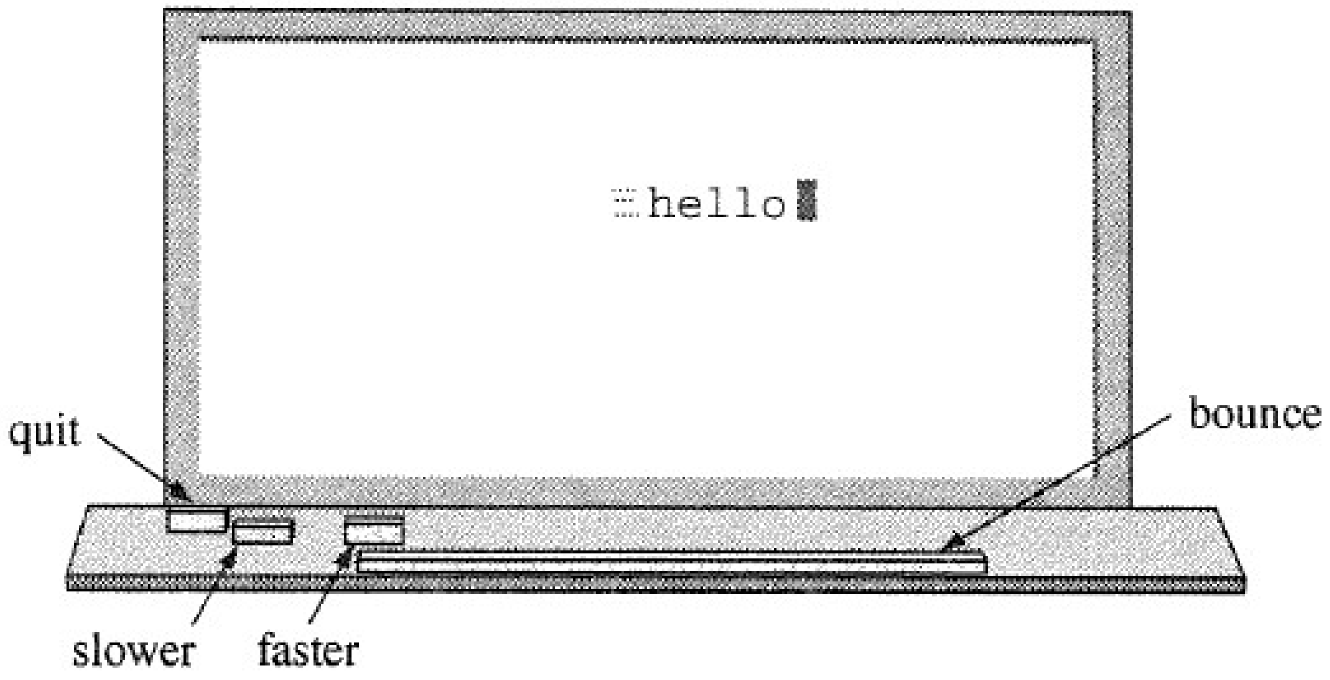
USAGE	int kill(pit_t pid, int sig)	
ARGS	pid sig	process id of target signal to throw
RETURNS	-1 0	on error on sucess

Kill: Sending signals from a process



VIDEO GAME

USING TIMERS AND SIGNALS : VIDEO GAMES

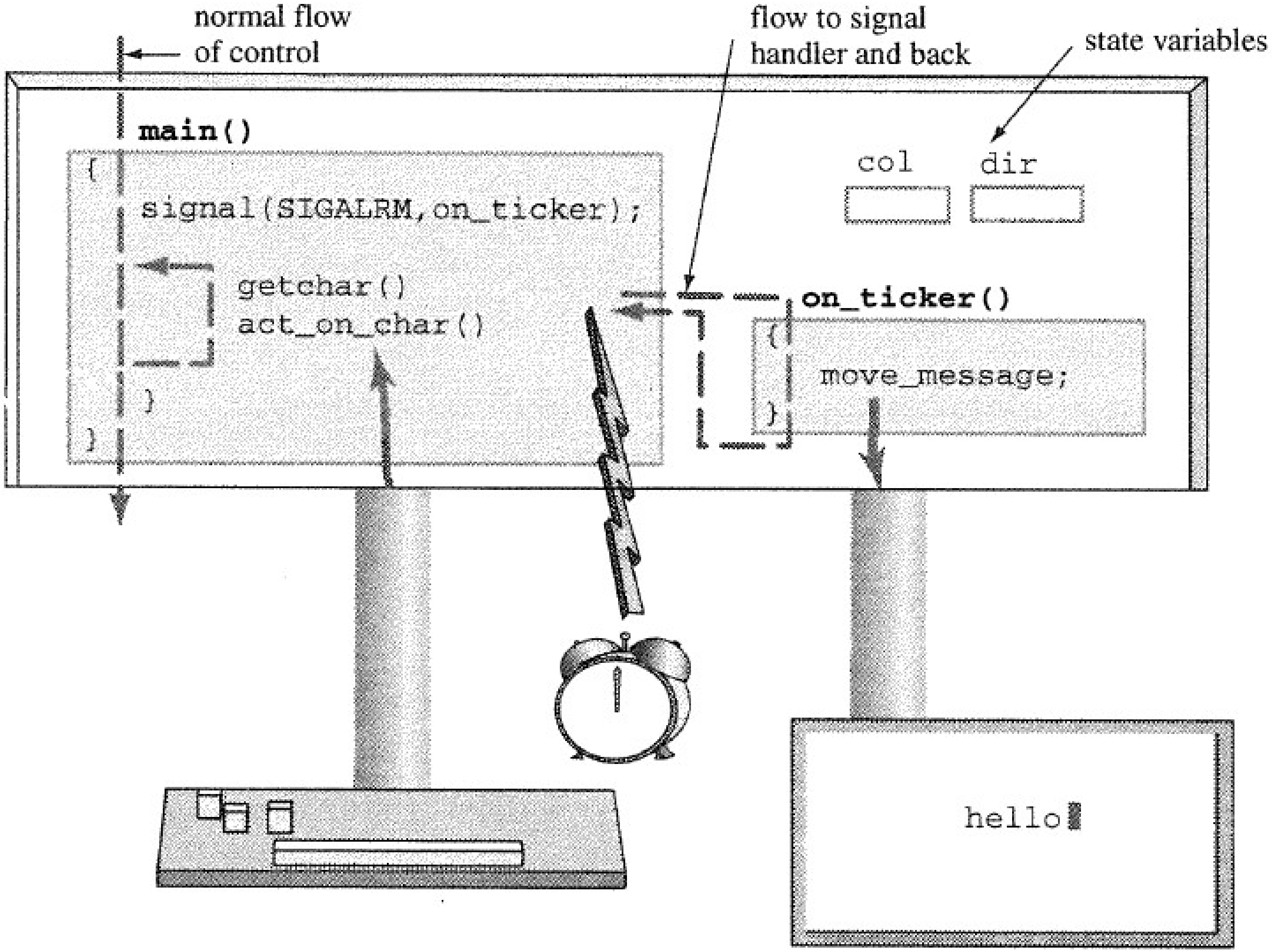


Moves a single word smoothly across the screen

space bar ➔ the message reverses direction

“**s**”and “**f**” makes the message move slower and faster

“**Q**”➔ quit the game


normal flow
of control
flow to signal
handler and back
state variables
main()
{
signal(SIGALRM,on_ticker);
getchar()
act_on_char()
}
}
col
dir
on_ticker()
{
move_message;
}
hello

CPU는 초시계가 있고 kernel 초시계 와 User 초시계가 따로 있습니다. [앞에서 한 내용]

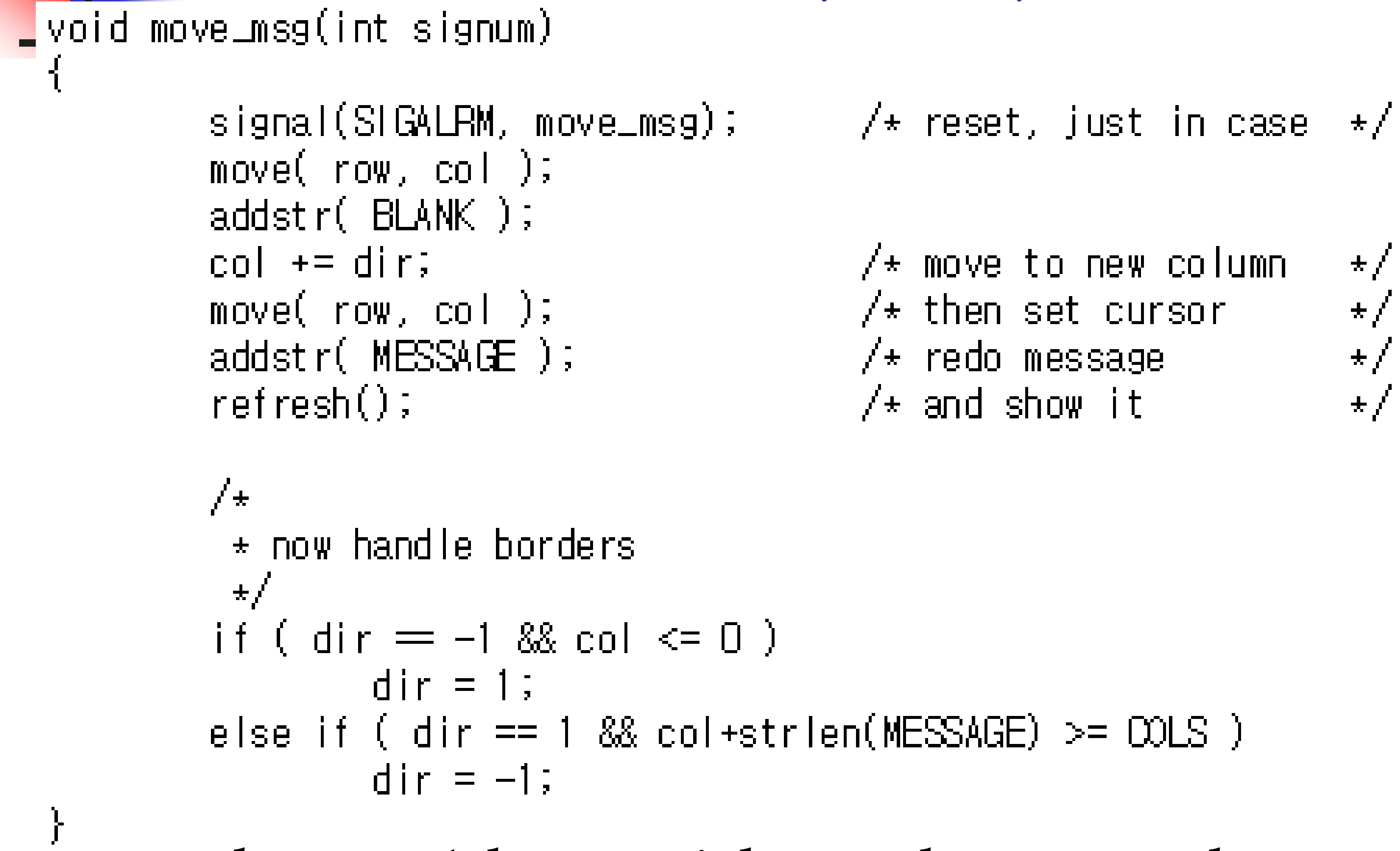
bounce1d.c (1/3)

```
#include        <stdio.h>
#include        <curses.h>
#include        <signal.h>

/* some global settings main and the handler use */

#define MESSAGE "hello"
#define BLANK   "     "

int     row;    /* current row          */
int     col;    /* current column       */
int     dir;    /* where we are going   */

int main()
{
        int     delay;          /* bigger => slower     */
        int     ndelay;         /* new delay            */
        int     c;              /* user input           */
        void    move_msg(int);  /* handler for timer    */
```

bounce1d.c (2/3)

```
        initscr();
        crmode();
        noecho();
        clear();

        row   = 10;             /* start here                  */
        col   = 0;
        dir   = 1;              /* add 1 to row number */
        delay = 200;            /* 200ms = 0.2 seconds */

        move(row,col);          /* get into position           */
        addstr(MESSAGE);        /* draw message                */
        signal(SIGALRM, move_msg );
        set_ticker( delay ); ← 알람 Signal 발생

        while(1)
        {
                ndelay = 0;
                c = getch();
                if ( c == 'Q' ) break;
                if ( c == ' ' ) dir = -dir;
                if ( c == 'f' && delay > 2 ) ndelay = delay/2;
                if ( c == 's' ) ndelay = delay * 2 ;
                if ( ndelay > 0 )
                        set_ticker( delay = ndelay );
        }
        endwin();
        return 0;
}
```

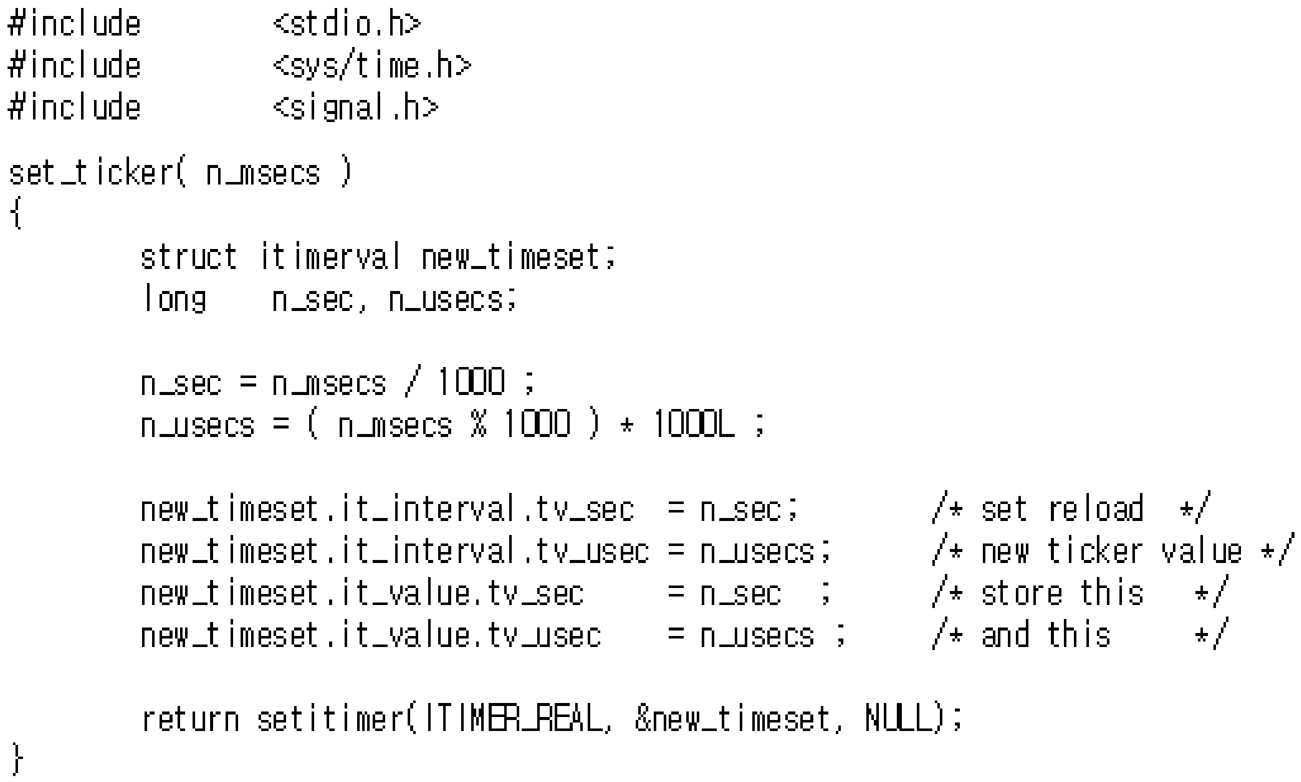
bounce1d.c (3/3)



```
void move_msg(int signum)
{
        signal(SIGALRM, move_msg);      /* reset, just in case  */
        move( row, col );
        addstr( BLANK );
        col += dir;                     /* move to new column   */
        move( row, col );               /* then set cursor      */
        addstr( MESSAGE );              /* redo message         */
        refresh();                      /* and show it          */

        /*
         * now handle borders
         */
        if ( dir == -1 && col <= 0 )
                dir = 1;
        else if ( dir == 1 && col+strlen(MESSAGE) >= COLS )
                dir = -1;
}
```

gcc bounce1d.c set_ticker.c –lcurses –o bounce1d

set_ticker.c



```
#include       <stdio.h>
#include       <sys/time.h>
#include       <signal.h>

set_ticker( n_msecs )
{
        struct itimerval new_timeset;
        long    n_sec, n_usecs;

        n_sec = n_msecs / 1000 ;
        n_usecs = ( n_msecs % 1000 ) * 1000L ;

        new_timeset.it_interval.tv_sec  = n_sec;        /* set reload  */
        new_timeset.it_interval.tv_usec = n_usecs;      /* new ticker value */
        new_timeset.it_value.tv_sec     = n_sec  ;      /* store this   */
        new_timeset.it_value.tv_usec    = n_usecs ;     /* and this     */

        return setitimer(ITIMER_REAL, &new_timeset, NULL);
}
```

gcc bounce1d.c set_ticker.c –lcurses –o bounce1d