ELEC462001 System Programming

Event-Driven Programming: Writing a Video Game (i)

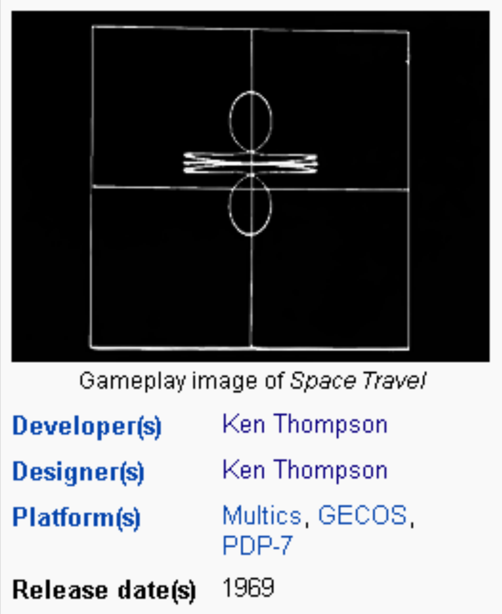
Prof. Seokin Hong

Kyungpook National University

Fall 2018

Space Travel

- Developed by Dennis Ritchie and Ken Thompson at Bell Labs in 1969.
- Simulates travel in the solar system.
- Thompson developed his own operating system, which later formed the core of the Unix operating system.

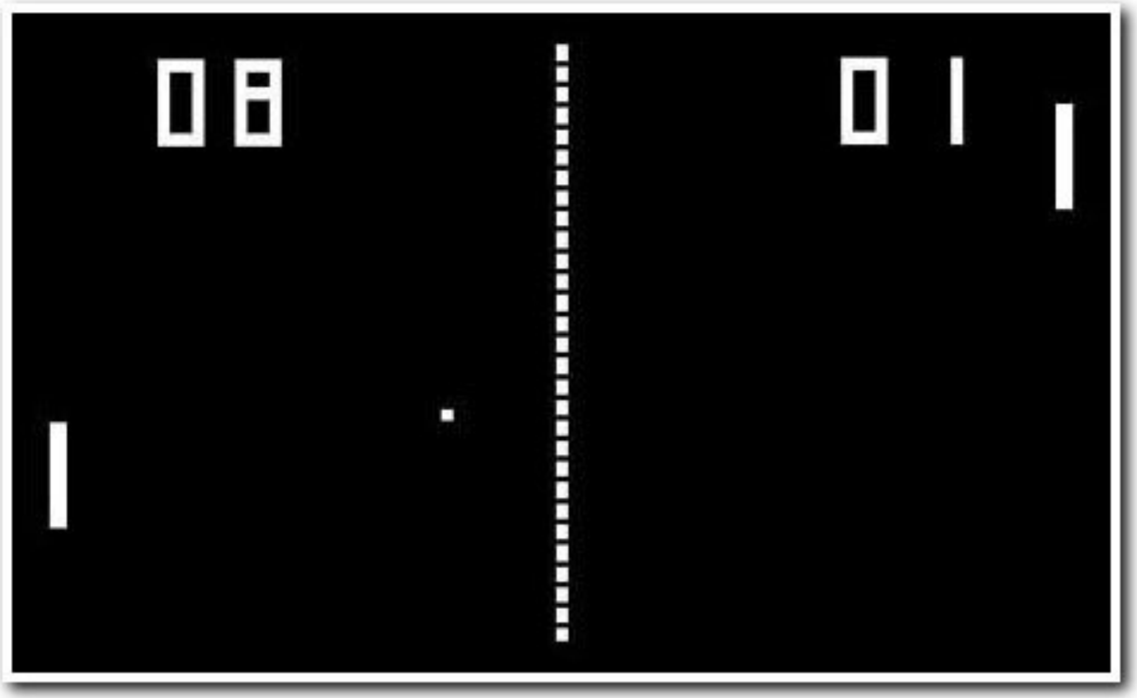


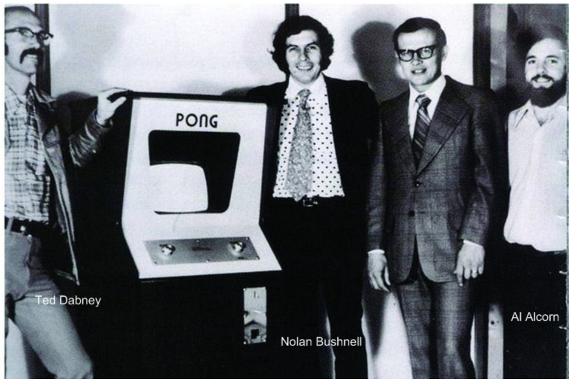
Gameplay image of *Space Travel*

Developer(s)	Ken Thompson
Designer(s)	Ken Thompson
Platform(s)	Multics, GECOS, PDP-7
Release date(s)	1969

PONG (one of the earliest arcade video games)

- Table tennis sports game.
- Developed by Atari and released in 1972.
- First commercially successful video game.





Ted Dabney

Nolan Bushnell

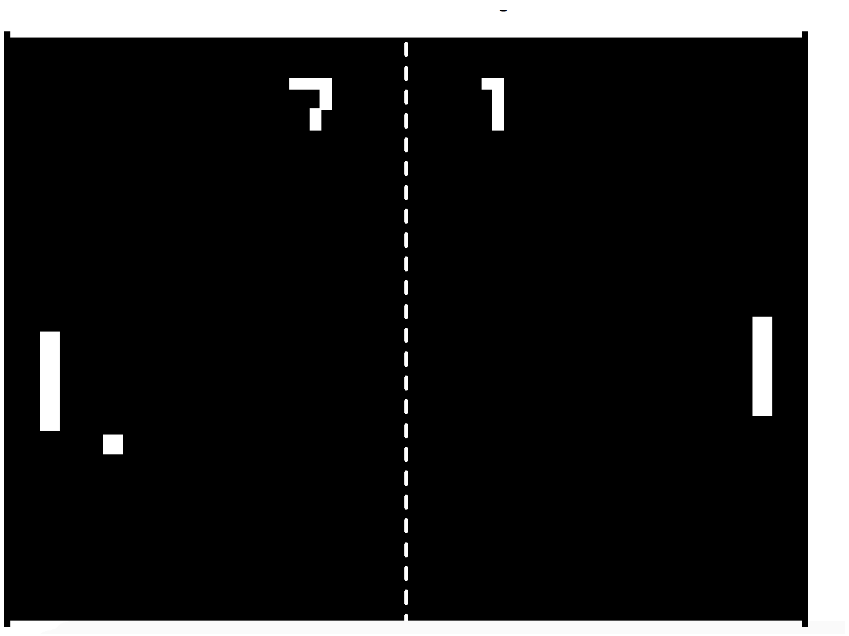
Al Alcorn

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 ORIGIANL ATARI PONG FROM
//SHIFT-REFRESH CONSULTING//

PONG (one of the earliest arcade video games)



http://www.ponggame.org/

(a) Ball keeps moving at some speed.
(b) Ball bounces off walls and paddle.
(c) User presses keys to move paddle up and down.

Objectives

- Ideas and Skills
  - Programs driven by asynchronous events
  - The curses library: purpose and use
  - Alarms and interval timers
  - Reliable signal handling
  - Reentrant code, critical sections
  - Asynchronous input

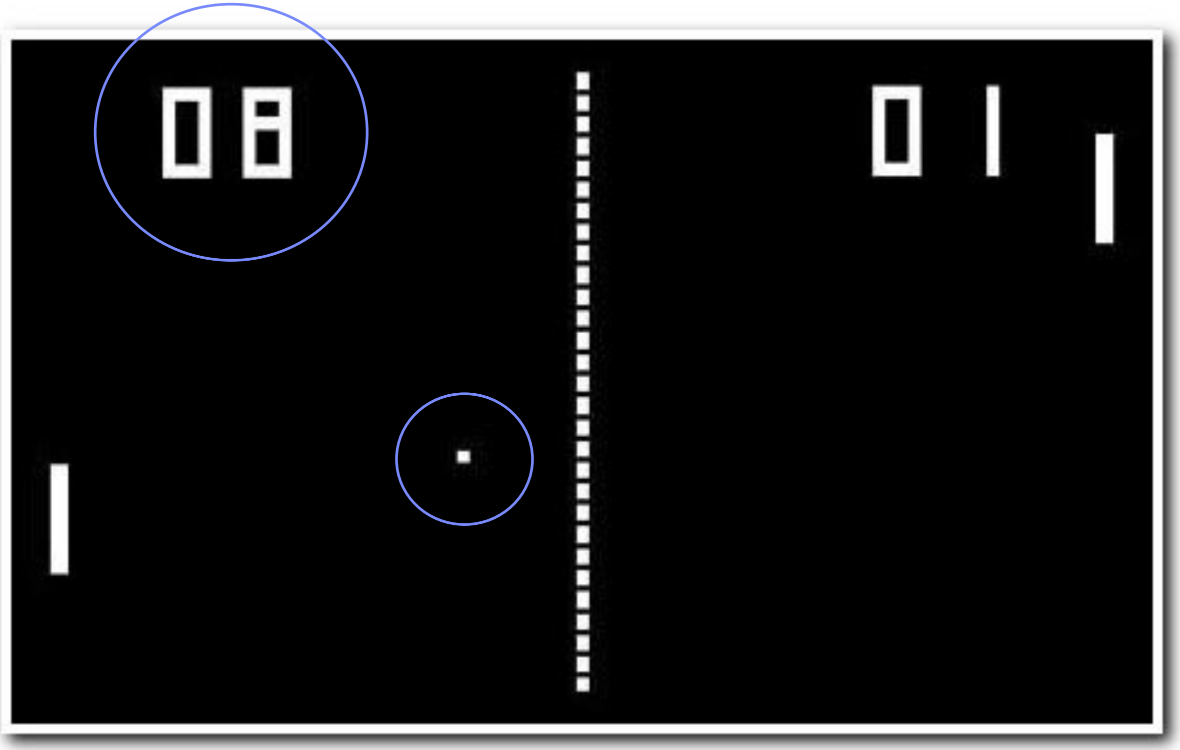
- System Calls
  - alarm, setitimer, getitimer
  - kill, pause
  - sigaction, sigprocmask
  - fcntl, aio_read

Contents

- 7.1 Video Games and Operating Systems
- 7.2 The Project : Write Single-Player pong
- **7.3 Space Programming : The curses Library**
- 7.4 Time Programming : sleep
- 7.5 Programming with Time I : Alarams
- 7.6 Programming with Time II : Interval Timers
- 7.7 Signal Handling I : Using signal
- 7.8 Signal Handling II : Using sigaction
- 7.9 Protecting Data from Corruption
- 7.10 kill: Sending Signals from a Process
- 7.11 Using Timers and Signals: Video Games
- 7.12 Signals on Input : Asynchronous I/O

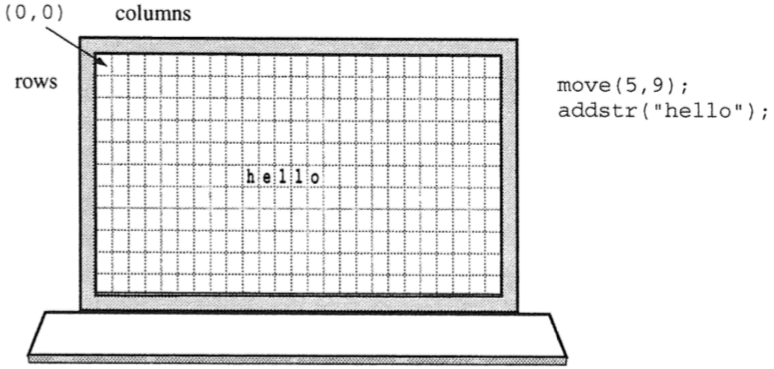
SPACE PROGRAMMING

- 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.

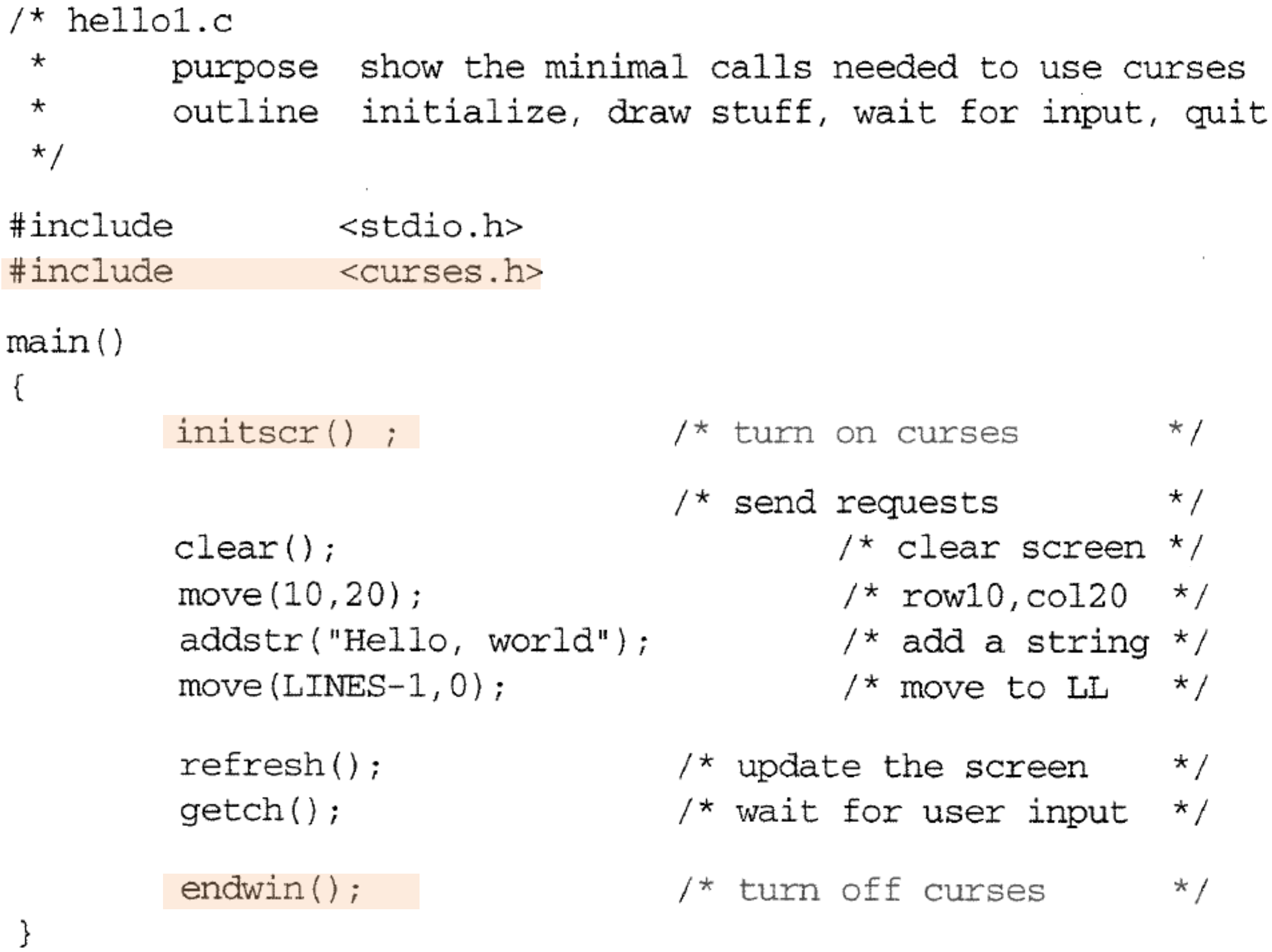
SPACE PROGRAMMING: The curses library

- **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

Standout mode is whatever special highlighting the terminal do..

Curses Example 1: hello1.c



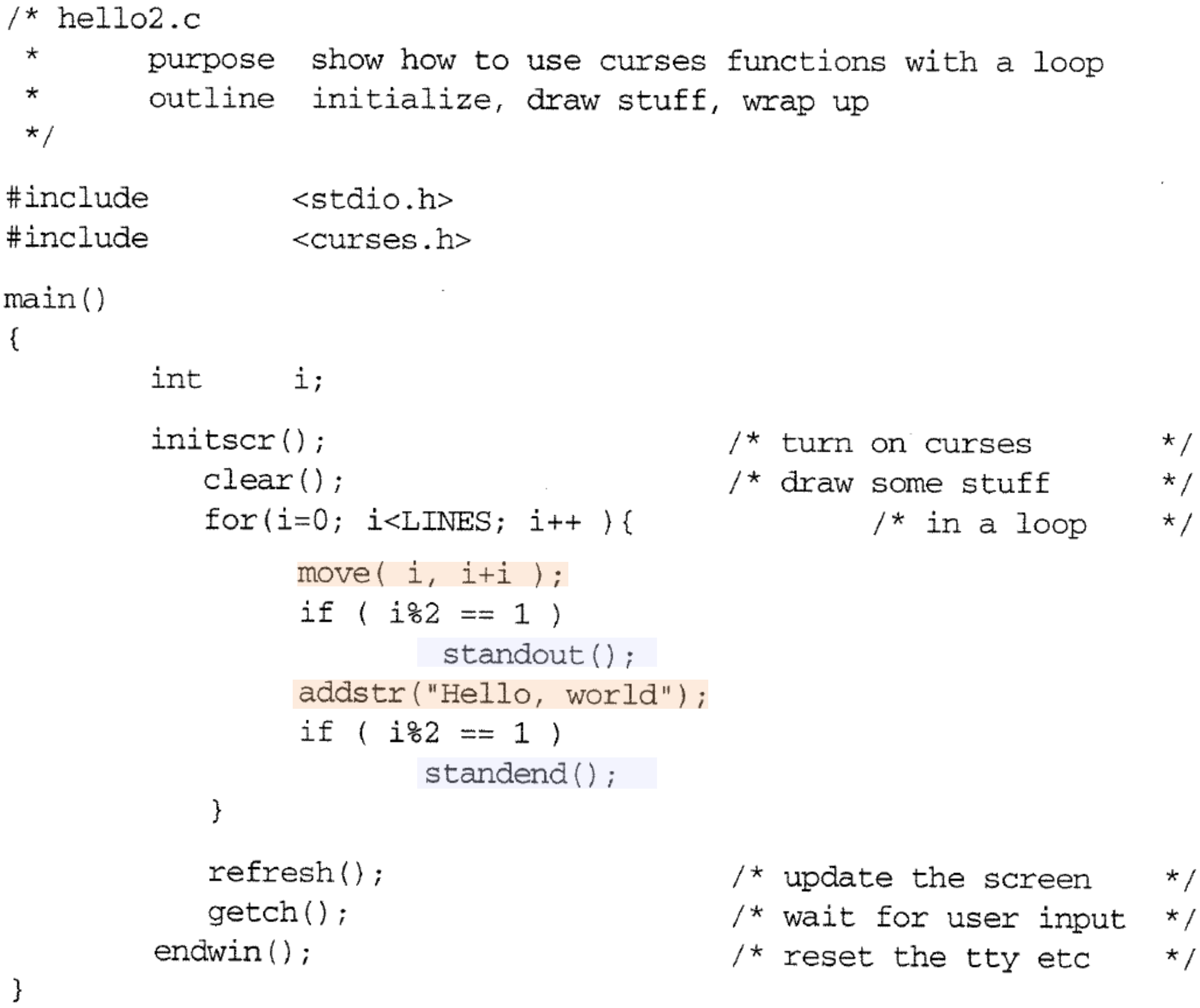
```
/* hello1.c
 *      purpose  show the minimal calls needed to use curses
 *      outline  initialize, draw stuff, wait for input, quit
 */

#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   */

        refresh();                      /* update the screen    */
        getch();                        /* wait for user input  */

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

- Compiling method

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

- What “-lcurses” means?
  - -l curses   (link curses library)

Curses Example 2: hello2.c



```c
/* hello2.c
 *      purpose  show how to use curses functions with a loop
 *      outline  initialize, draw stuff, wrap up
 */

#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    */
           getch();                             /* wait for user input  */
        endwin();                               /* reset the tty etc    */
}
```

Curses Internals: Virtual and Real Screens

- Curses minimizes data flow by working with virtual screens.

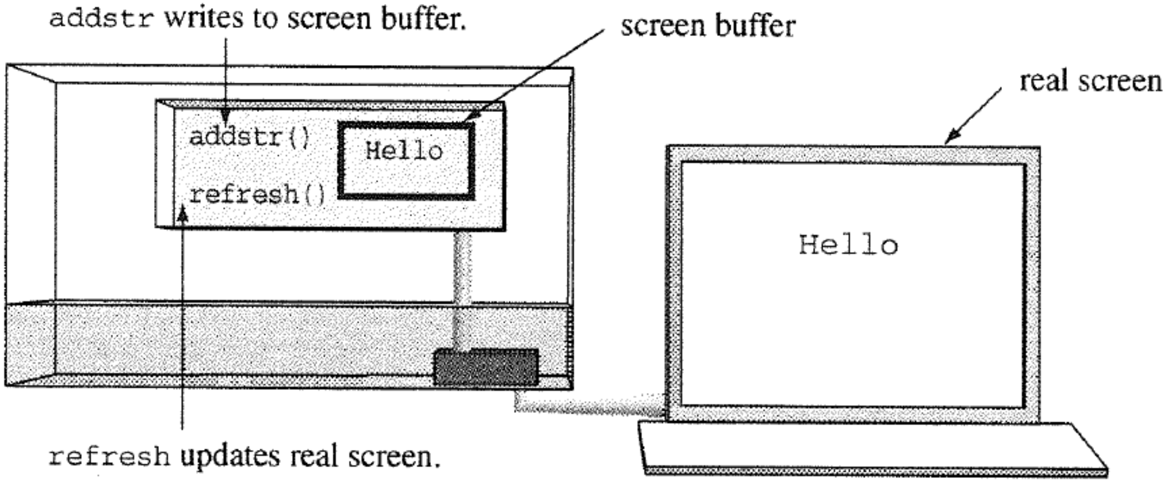


FIGURE 7.4

Curses keeps a copy of the real screen.

- In Hello2.c, **comment out the refresh function** and recompile, and run the program.
- Compare the workspace screen to the copy of the real screen
- Sends out through the terminal driver the characters

Curses Internals: Virtual and Real Screens

- Curses keeps two internal versions of the screen.
  - a copy of the real screen
  - a workspace screen
    - records changes to the screen

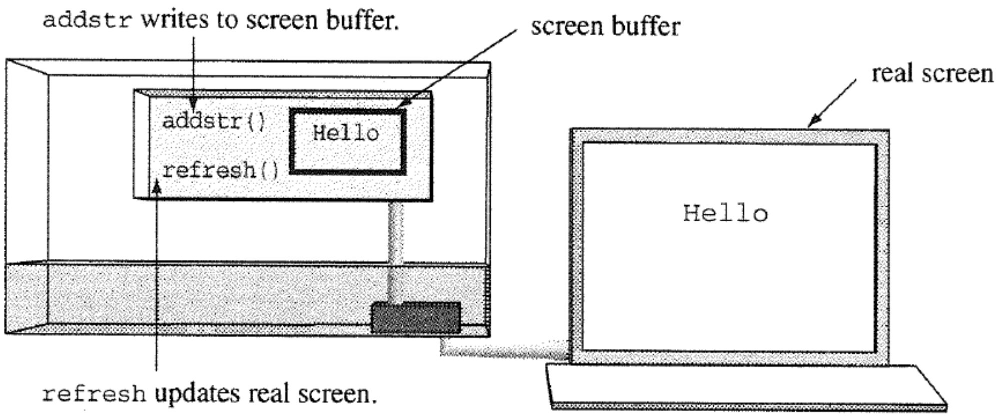


FIGURE 7.4

Curses keeps a copy of the real screen.

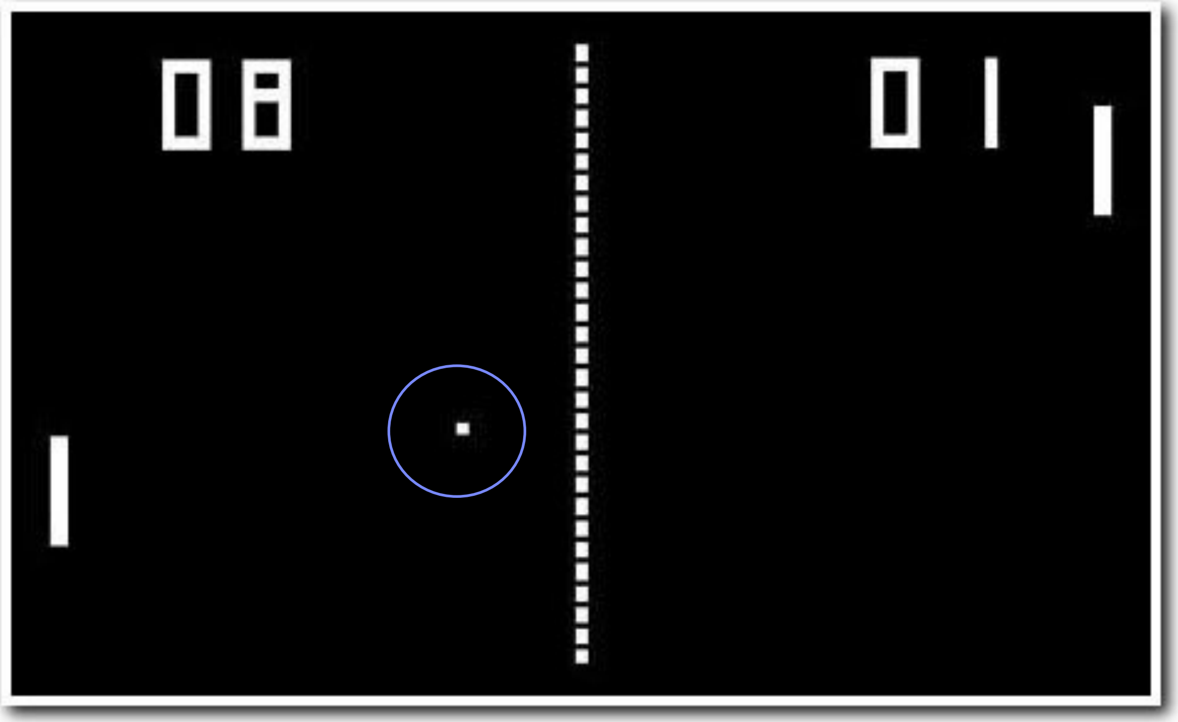
  - Most functions in the curses library affect only this workspace screen, like disk buffering.
  - The refresh function compares the workspace screen to the copy of the real screen,
  - then sends out through the terminal driver the characters to make the real screen match the working screen.

Contents

- 7.1 Video Games and Operating Systems
- 7.2 The Project : Write Single-Player pong
- 7.3 Space Programming : The curses Library
- 7.4 Time Programming : sleep
- 7.5 Programming with Time I : Alarams
- 7.6 Programming with Time II : Interval Timers
- 7.7 Signal Handling I : Using signal
- 7.8 Signal Handling II : Using sigaction
- 7.9 Protecting Data from Corruption
- 7.10 kill: Sending Signals from a Process
- 7.11 Using Timers and Signals: Video Games
- 7.12 Signals on Input : Asynchronous I/O

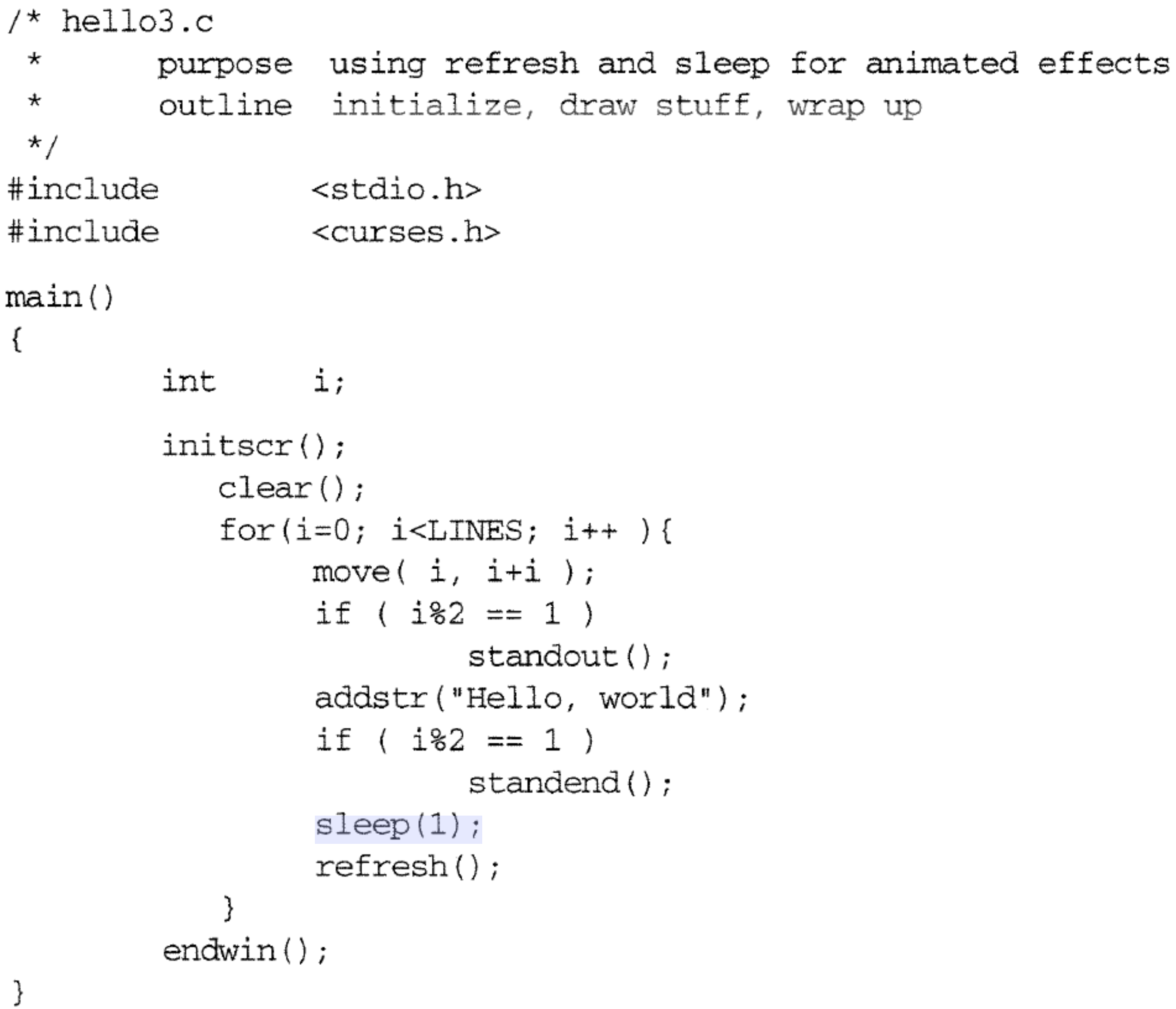
TIME HANDLING

How to move or to show animated effects the images?



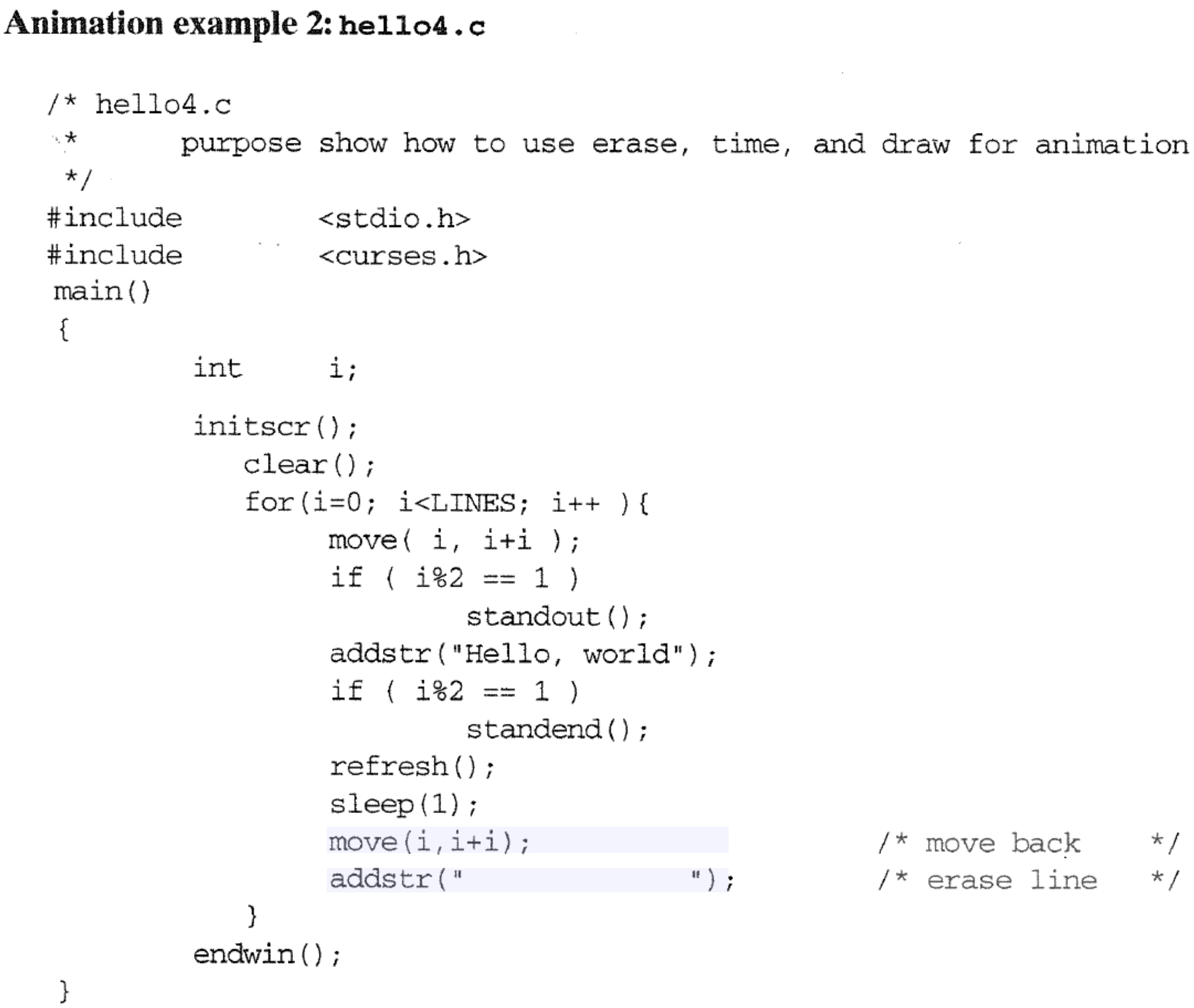
To write a video game, we have to put images at specific places at specific times

Animation example1: hello3.c



```
/* hello3.c
 *      purpose  using refresh and sleep for animated effects
 *      outline  initialize, draw stuff, wrap up
 */
#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();
}
```

Animation example2: hello4.c



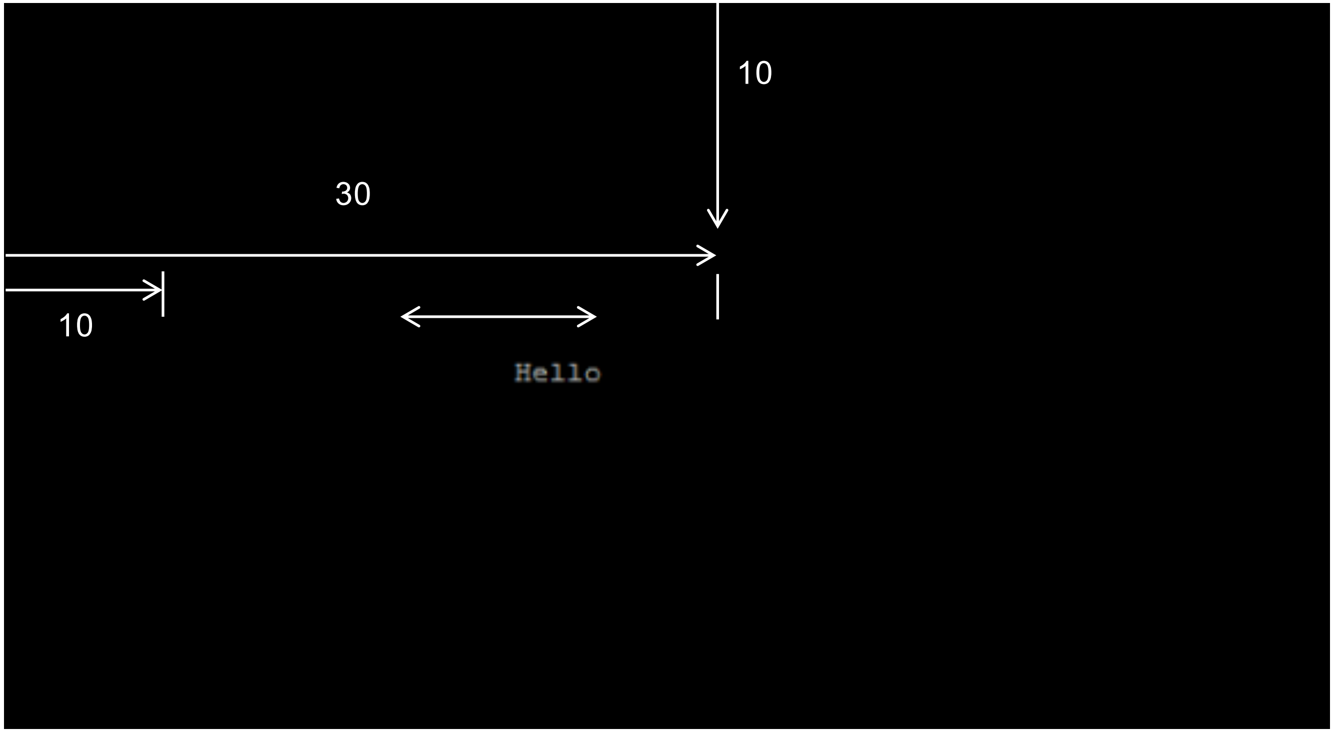
Animation example 2: hello4.c

```
/* hello4.c
 *      purpose show how to use erase, time, and draw for animation
 */
#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();
                refresh();
                sleep(1);
                move(i,i+i);                            /* move back    */
                addstr("                   ");          /* erase line   */
           }
        endwin();
}
```

Animation example 3: `hello5.c`

```
/* hello5.c
 *    purpose  bounce a message back and forth across the screen
 *    compile  cc hello5.c -lcurses -o hello5
 */
#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;
          }
}
```

Hello5.c



```
Hello
```

How Are We Doing?

- We know
  - how to draw string anywhere on the screen,
  - how to create animation by introducing delays between drawings, erasings, and redrawings.

- Our programs are nice, but
  - One-second delays are too long;
  - we need better control of time.
  - We need to add user input.

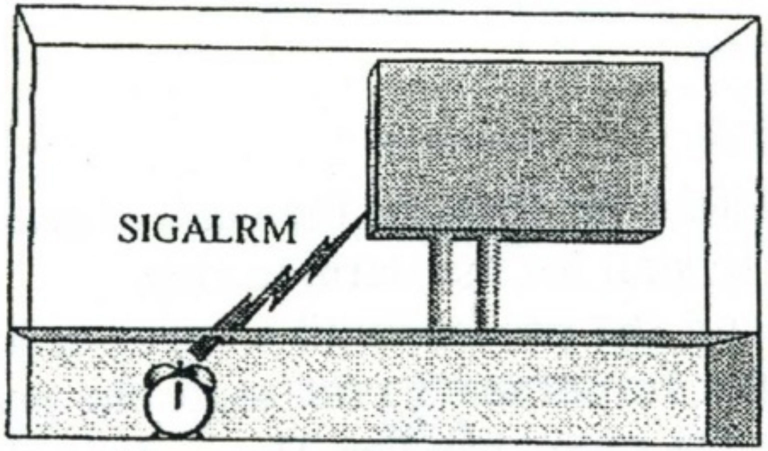
- New topics
  - programming with time and advanced signals

Contents

- 7.1 Video Games and Operating Systems
- 7.2 The Project : Write Single-Player pong
- 7.3 Space Programming : The curses Library
- 7.4 Time Programming : sleep
- 7.5 Programming with Time I : Alarams
- 7.6 Programming with Time II : Interval Timers
- 7.7 Signal Handling I : Using signal
- 7.8 Signal Handling II : Using sigaction
- 7.9 Protecting Data from Corruption
- 7.10 kill: Sending Signals from a Process
- 7.11 Using Timers and Signals: Video Games
- 7.12 Signals on Input : Asynchronous I/O

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.

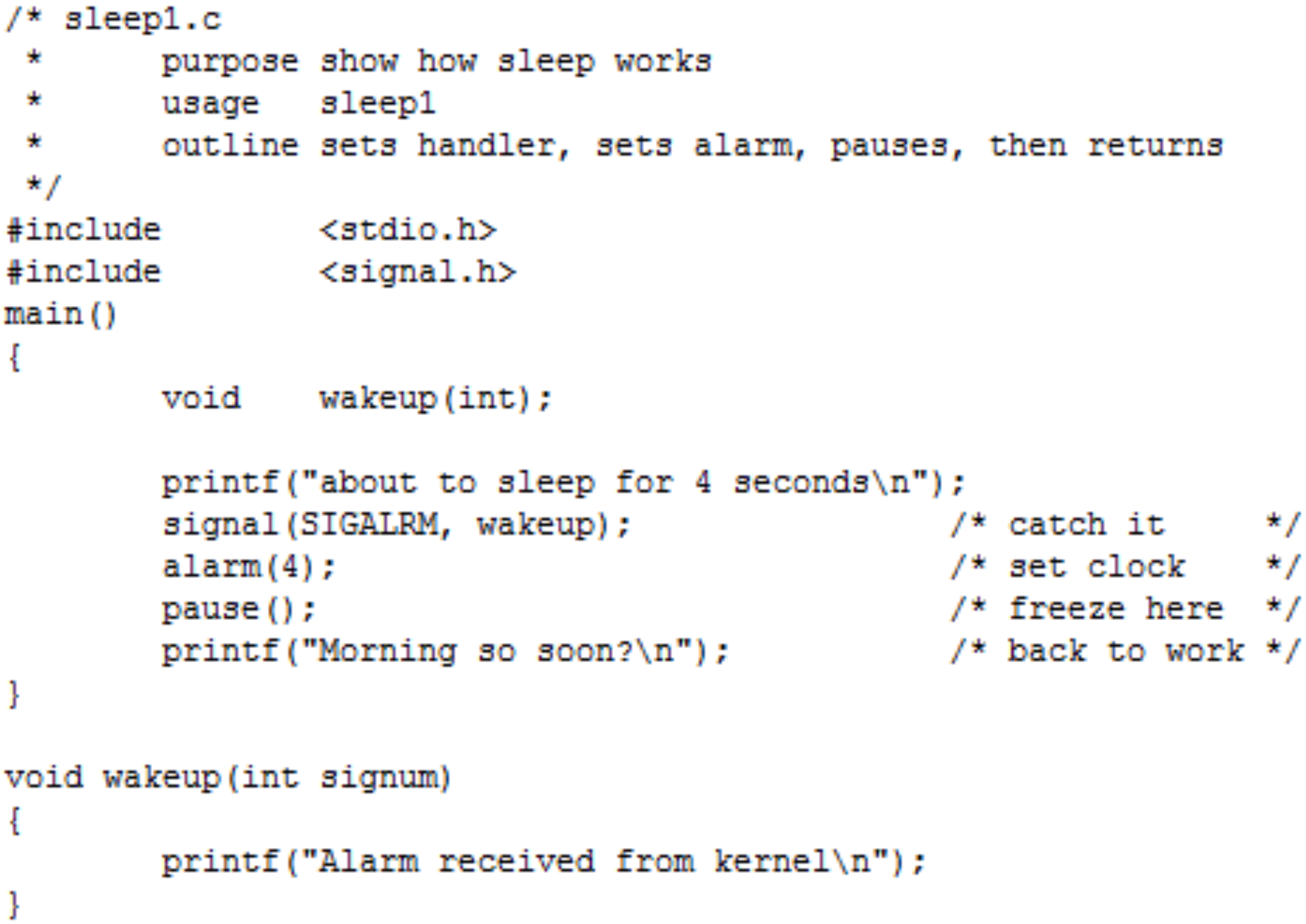
FIGURE 7.7

A process sets an alarm then suspends execution.

How the sleep function works:

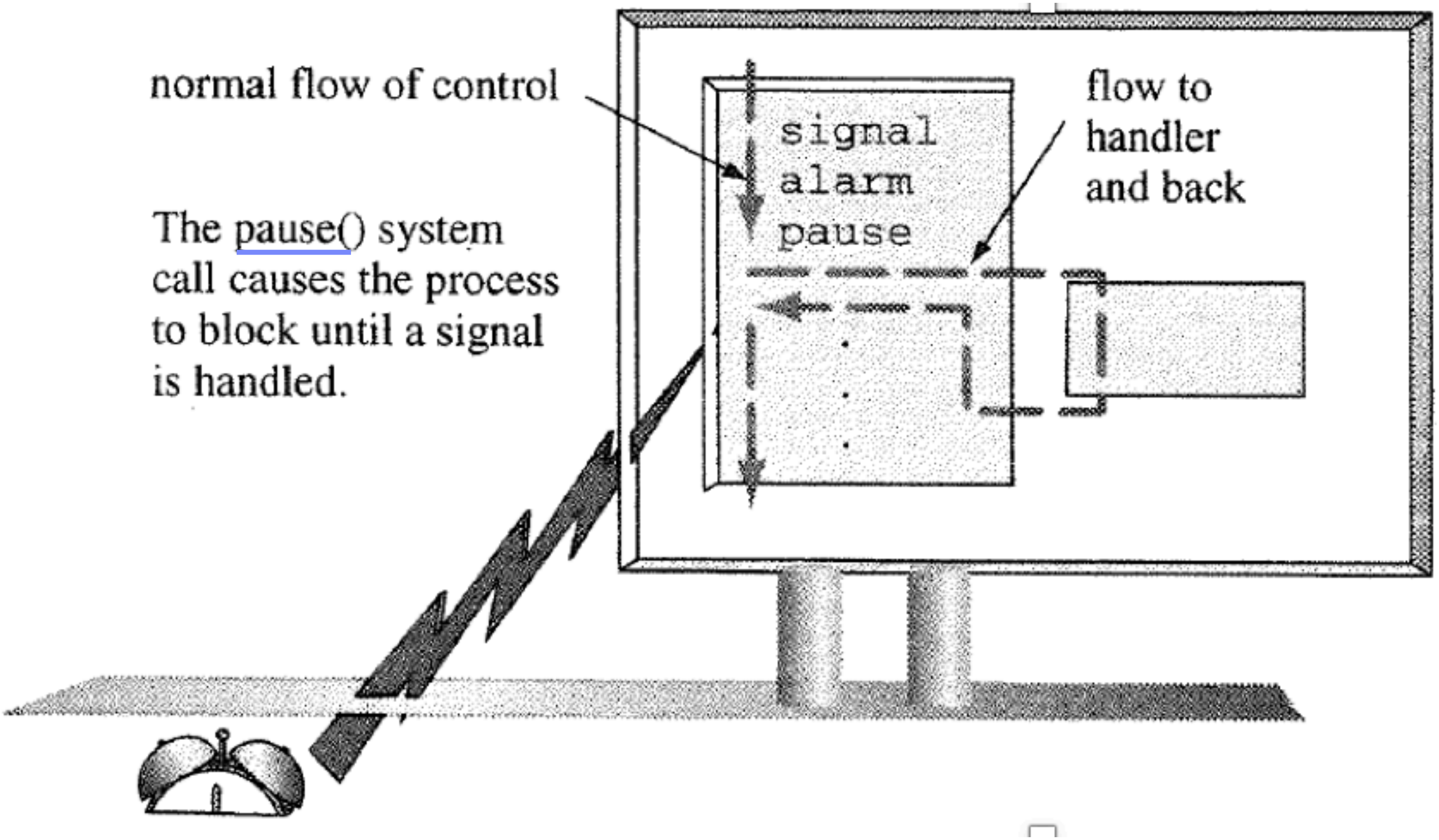
- signal(SIGALRM, handler);
- alarm(n);
- pause();

PROGAMMING WITH TIME I : ALARMS



```
/* 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);                               /* 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



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

Scheduling a Future Action

- The other way to use time is
  - to schedule an action for some future time and do something else in the meantime.

- Scheduling an action in the future:
  - Set the **timer** by calling alarm then proceed to do something else.
  - When the timer reaches zero, the signal will be sent, and the handler will be invoked.

Contents

- 7.1 Video Games and Operating Systems
- 7.2 The Project : Write Single-Player pong
- 7.3 Space Programming : The curses Library
- 7.4 Time Programming : sleep
- 7.5 Programming with Time I : Alarams
- **7.6 Programming with Time II : Interval Timers**
- 7.7 Signal Handling I : Using signal
- 7.8 Signal Handling II : Using sigaction
- 7.9 Protecting Data from Corruption
- 7.10 kill: Sending Signals from a Process
- 7.11 Using Timers and Signals: Video Games
- 7.12 Signals on Input : Asynchronous I/O

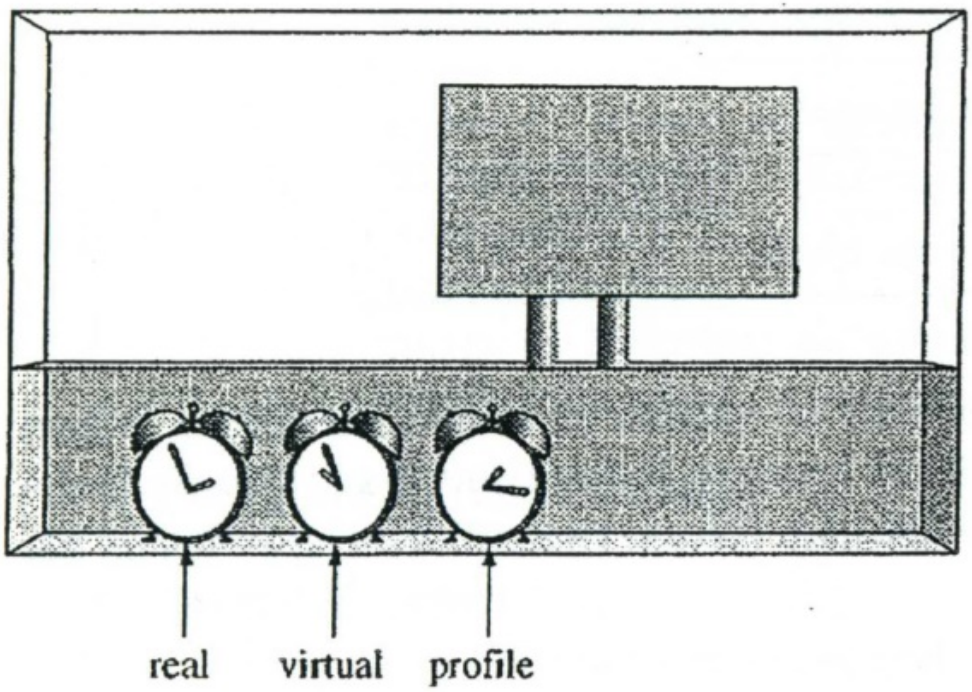
PROGAMMING WITH TIME 2: INTERVAL TIMERS

- 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.
  - Each timer has two settings
    - The time until the first alarm
    - The interval between repeating alarms



Every process has three timers.

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

Three Kinds of Timer: Real, Virtual, Profile

- Processes can measure three kinds of time:
  - o **Real timer :** counts elapsed time --> CPU time + IO time + waiting time
  - o **Virtual timer :** counts elapsed time used by the process. --> CPU time
  - o **Profile timer :** counts both elapsed time used by the process and by system calls on behalf of the process.

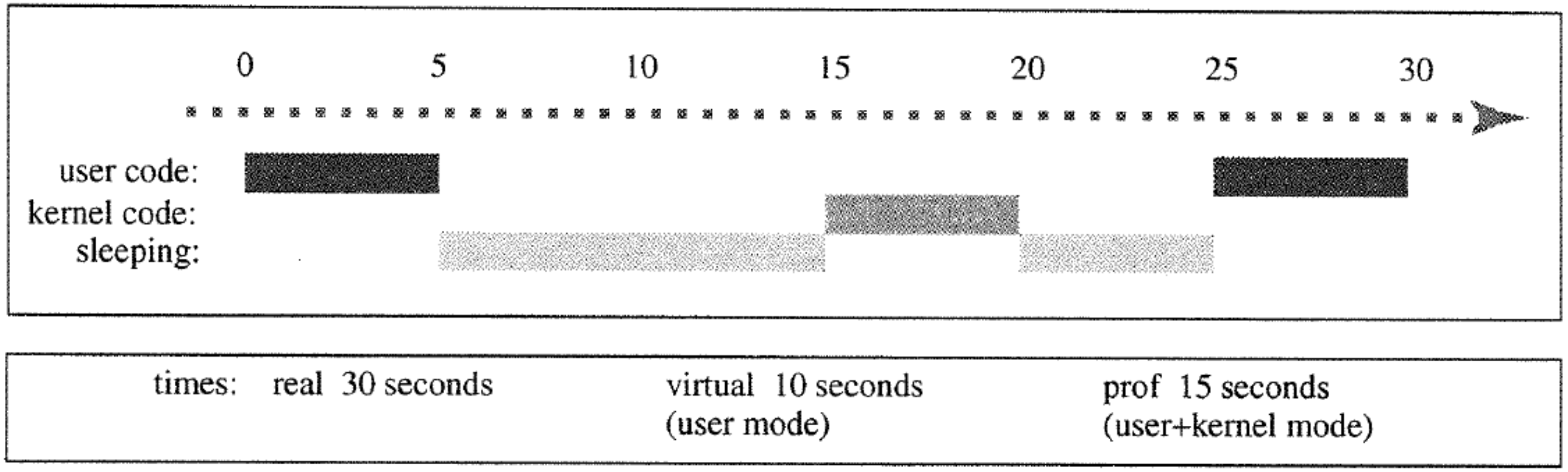


FIGURE 7.10

Where does the time go?

Three Kinds of Timer: Real, Virtual, Profile

- 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
    - 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

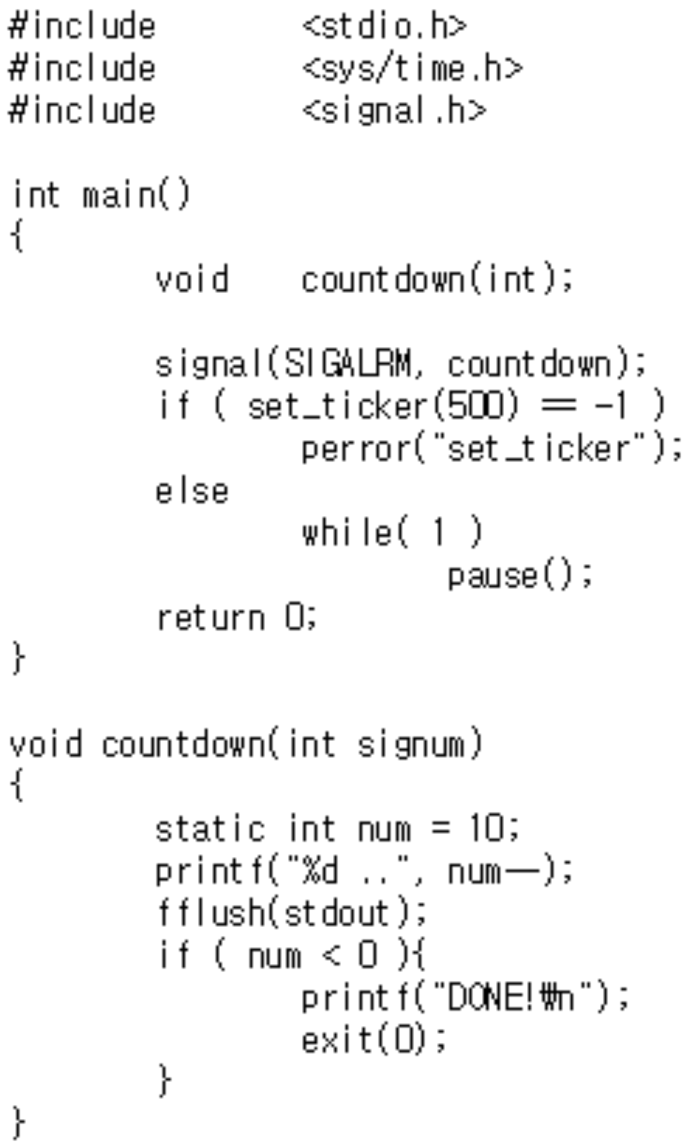
- Programming with the Interval Timers
  - Decide on an **initial interval** and a **repeating interval**
  - Set values in a **struct itimerval**
    - Initial interval and repeating interval
  - Pass the structure to the timer by calling **setitimer**

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 timeval
{
    time_t          tv_sec;         /* seconds */
    suseconds_t     tv_usec;        /* and microseconds */
};
```

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);
        }
}
```

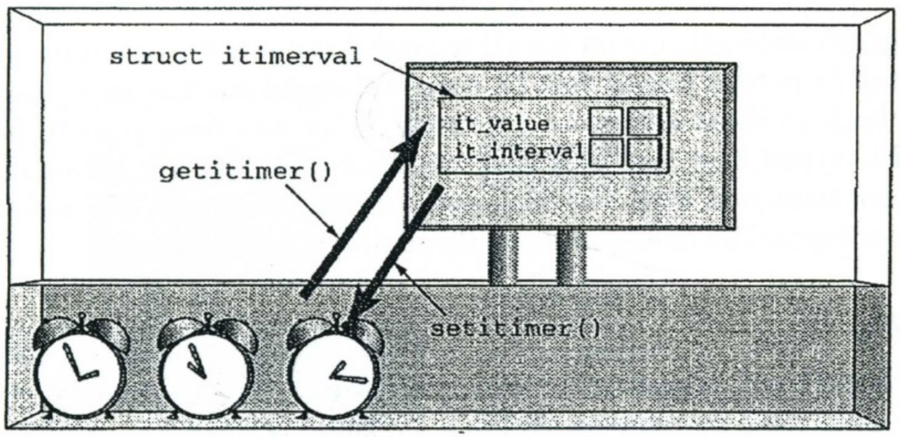
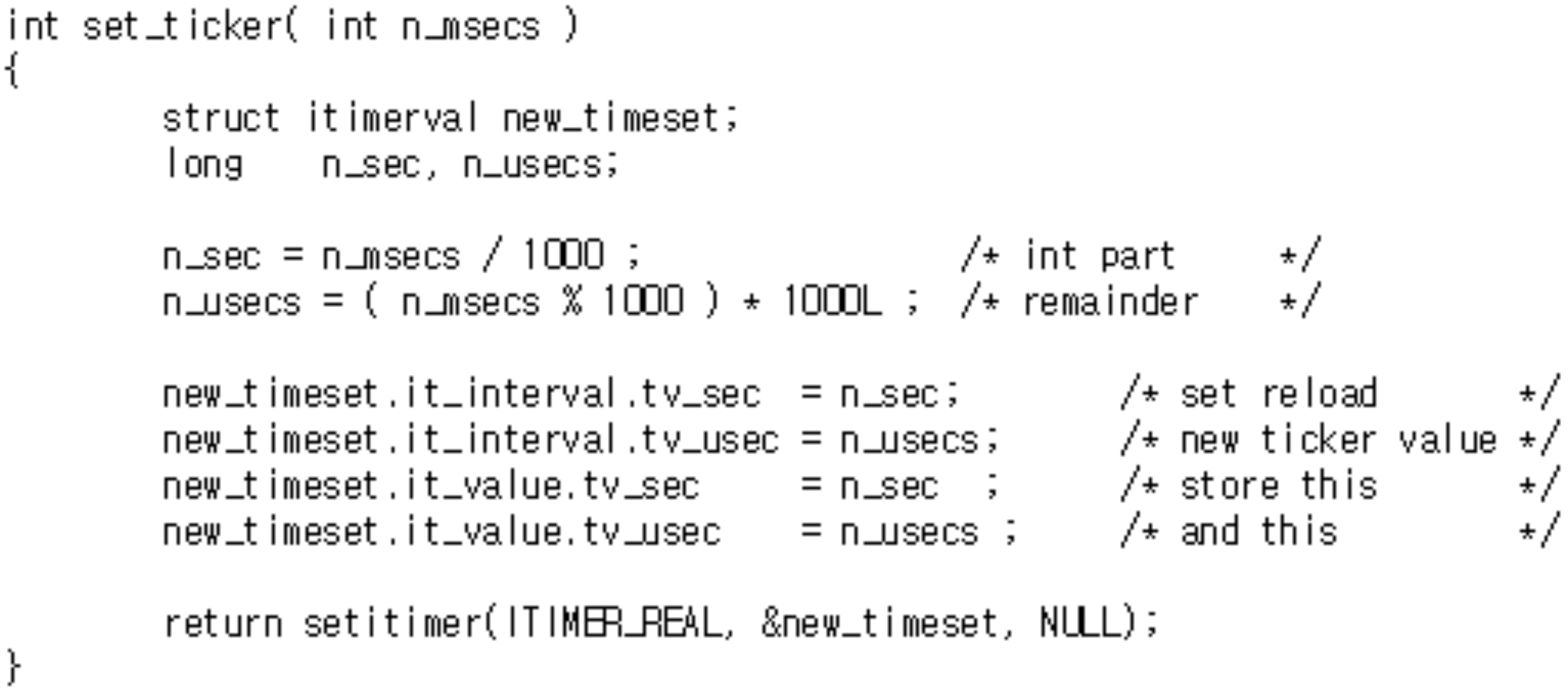


FIGURE 7.11

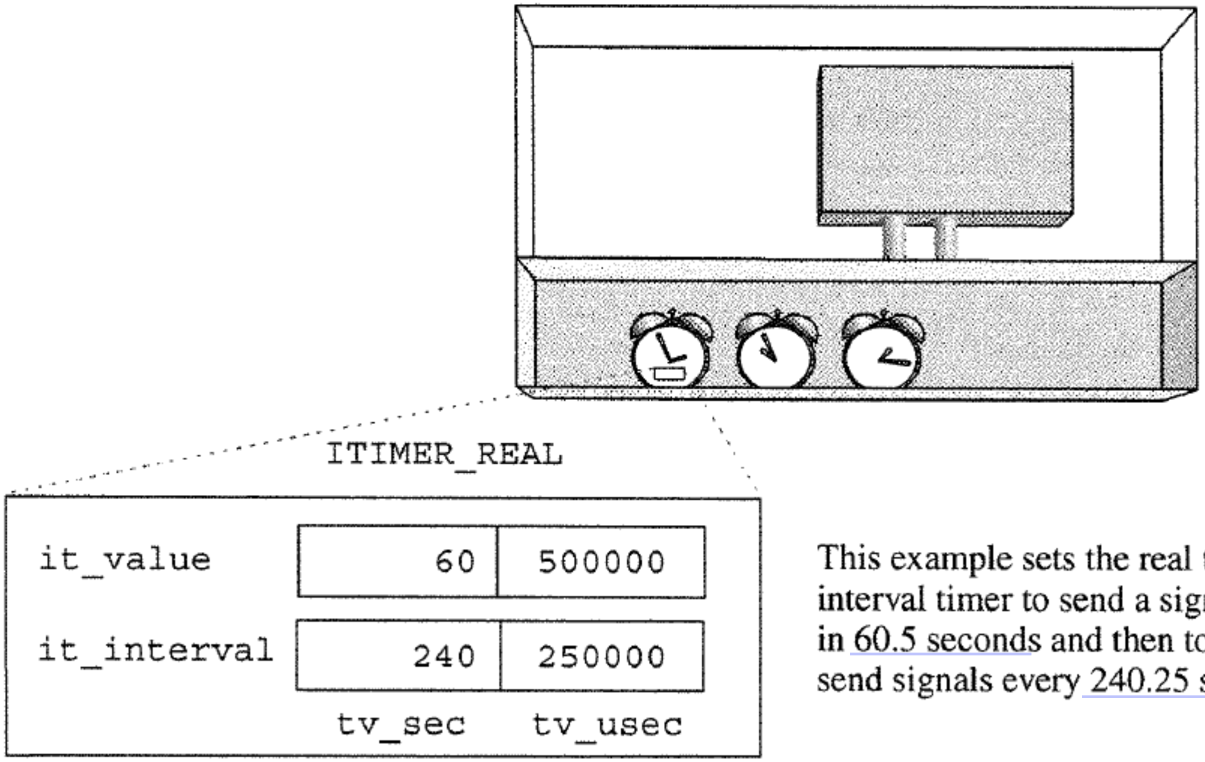
Reading and writing timer settings.

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



	tv_sec	tv_usec
it_value	60	500000
it_interval	240	250000

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.

FIGURE 7.13

Seconds and microseconds.

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

Summary of Timers

- A Unix program uses timers
  - to suspend execution and
  - to schedule future actions.

- A timer is a mechanism in the kernel that sends a signal to the process after a specified interval.
  - alarm system call arranges to send SIGALRM to the process after a specified number of seconds of real time.
  - setitimer system call controls timers with high resolution and the ability to send signals at regular intervals.