Ch7. Event-Driven Programming: Writing a Video Game

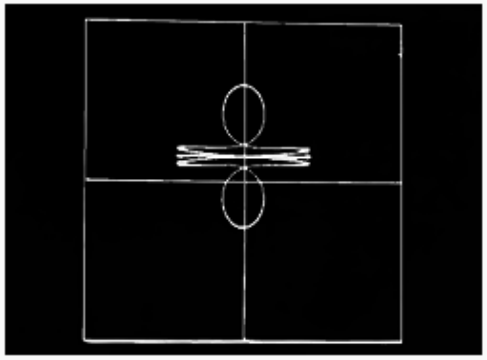
Prof. Seokin Hong

Kyungpook National University

Fall 2019

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.

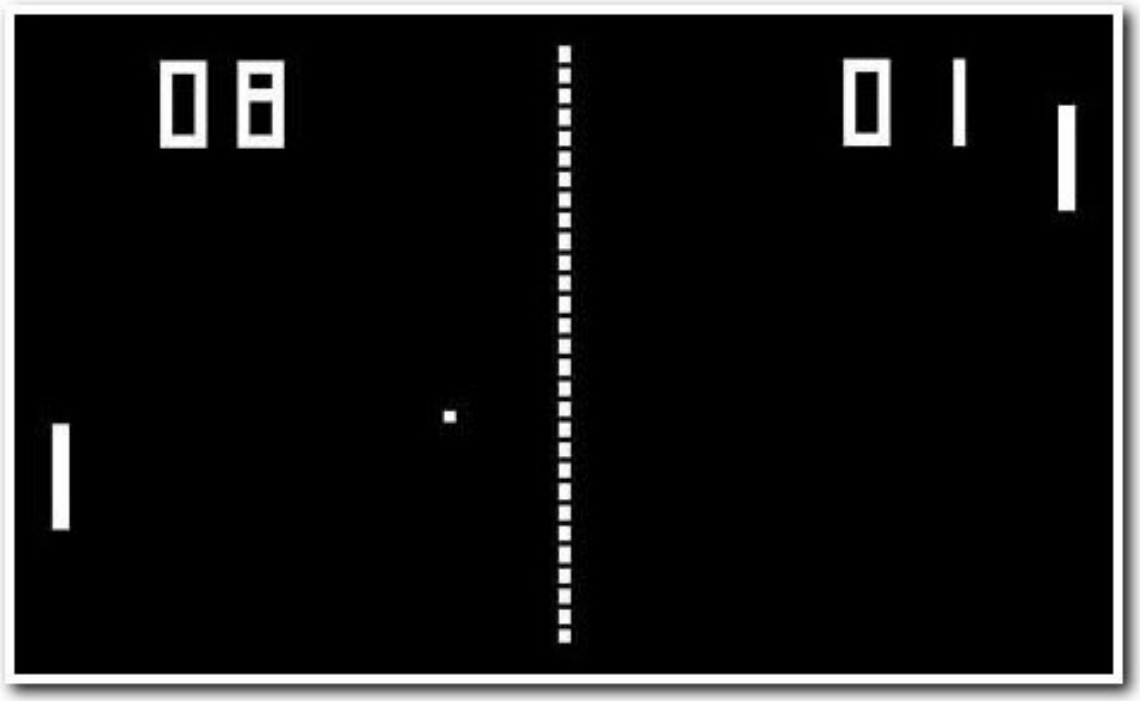


Gameplay image of *Space Travel*

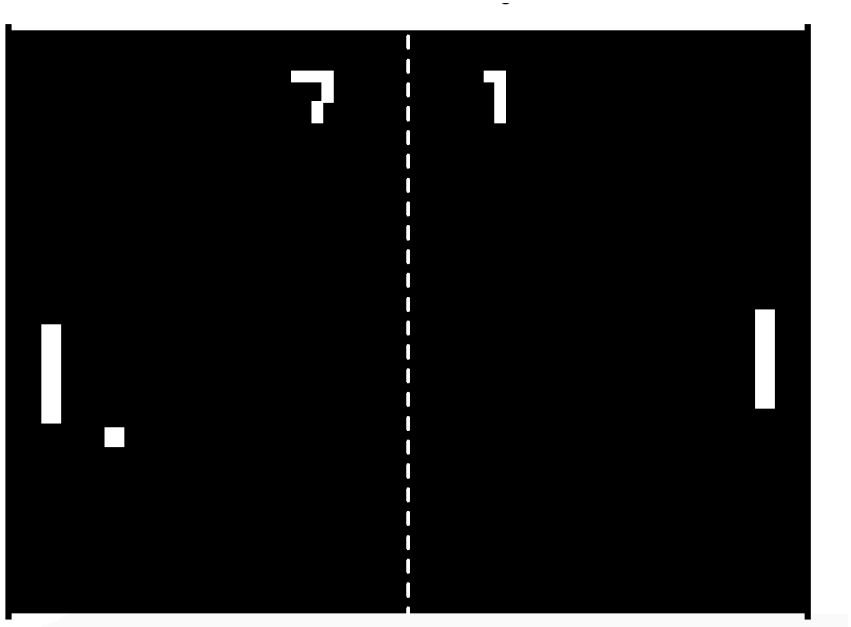
Developer(s)	Ken Thompson
Designer(s)	Ken Thompson
Platform(s)	Multics, GECOS, PDP-7
Release date(s)	1969

PONG (one of the earliest arcade video games)

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



PONG (one of the earliest arcade video games)



http://www.ponggame.org/

- Ball keeps moving at some speed
- Ball bounces off walls and paddle
- User presses keys to move paddle up and down

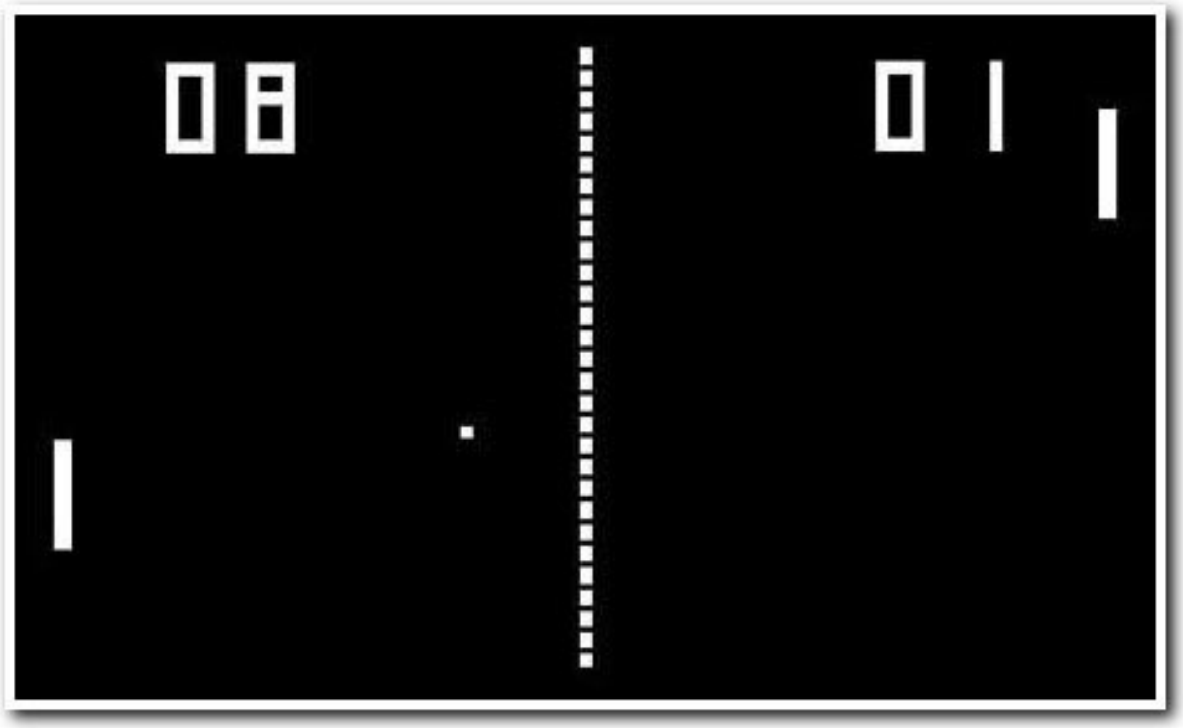
Objectives

- 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

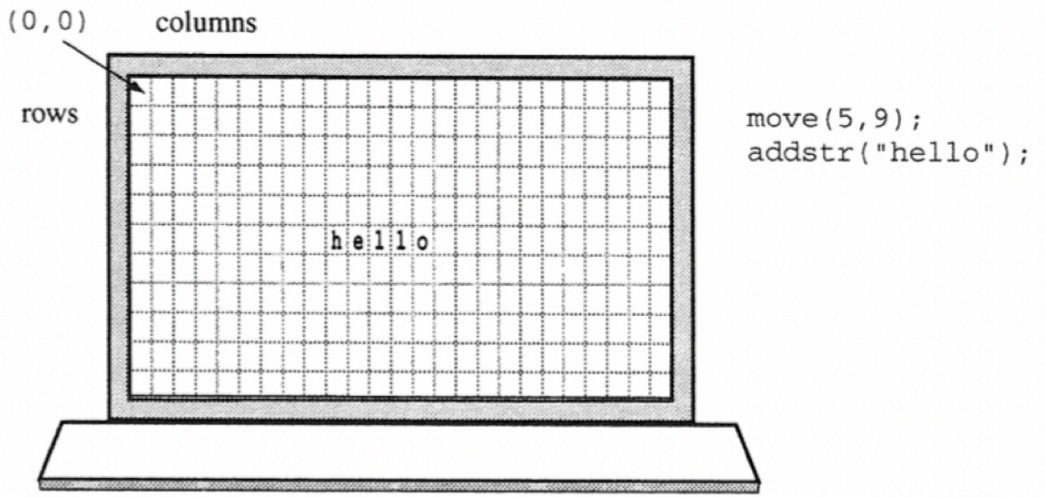
SPACE PROGRAMMING

- 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
  - Set the position of the cursor
  - 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



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(행), c(열))	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

Ex1: hello1.c

```
 1 /* hello1.c
 2  *  purpose show the minimal calls needed to use curses
 3  *  outline initialize, draw stuff, wait for input, quit
 4  */
 5
 6 #include <stdio.h>
 7 #include <curses.h>
 8
 9 int main(void)
10 {
11     initscr();
12     clear();
13
14     move(10, 20);
15     addstr("Hello, World");
16     move(LINES-1, 0);
17
18     refresh();
19     getch();
20
21     endwin();
22
23     return 0;
24 }
```

Compile

$ gcc hello1.c –o hello1 **–lcurses**

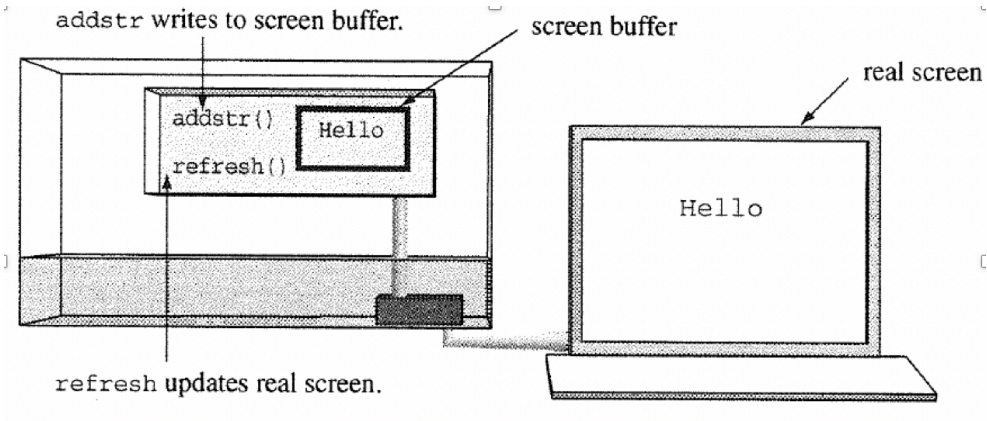
$ ./hello1

Ex2: 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>
#include <unistd.h>

int main(void)
{
    int i;

    initscr();          /* turn on curses*/
    clear();
    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(3);
    endwin();           /* reset the tty etc*/

    return 0;
}
```

Curses Internals: Virtual and Real Screens

- Curses minimizes data flow by working with virtual screens.
- Curses keeps two internal versions of the screen.
  - **a copy of the real screen (curscr)**
  - **a workspace screen (stdscr)**
    - records changes to the screen

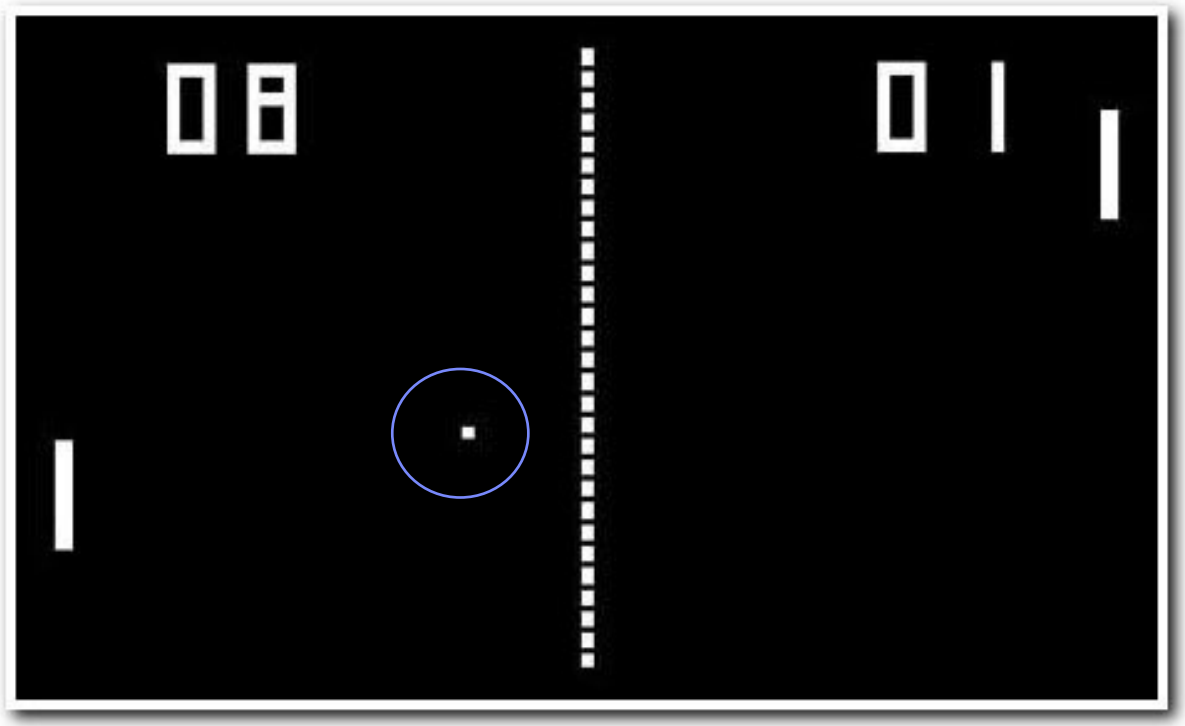


  - Most functions in the curses library affect only the workspace screen
  - The **refresh** function compares the workspace screen to the copy of the real screen,
  - then sends out through the terminal driver the characters to make the real screen match the working screen.

Contents

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

Ex3: hello3.c

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

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

int main(void)
{
    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();
    return 0;
}
```

Ex4: hello4.c

```c
/* hello4.c
 *  purpose show how to use erase, time, and draw for animation
 */

#include    <stdio.h>
#include    <curses.h>
#include "unistd.h"

int main(void)
{
    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();

    return 0;
}
```

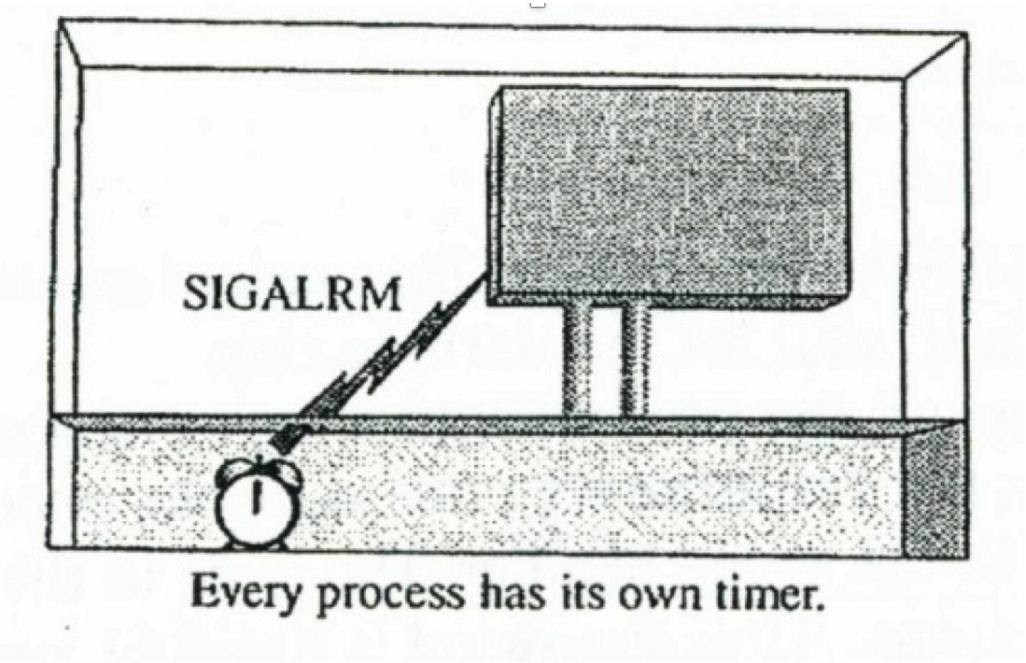
Ex5: hello5.c

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


#include <curses.h>
#include "unistd.h"
#define  LEFTEDGE  10
#define  RIGHTEDGE  30
#define  ROW  10

int main(void)
{
    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;
     }

    return 0;
}
```

Contents

PROGAMMING WITH TIME I : ALARMS

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



Every process has its own timer.

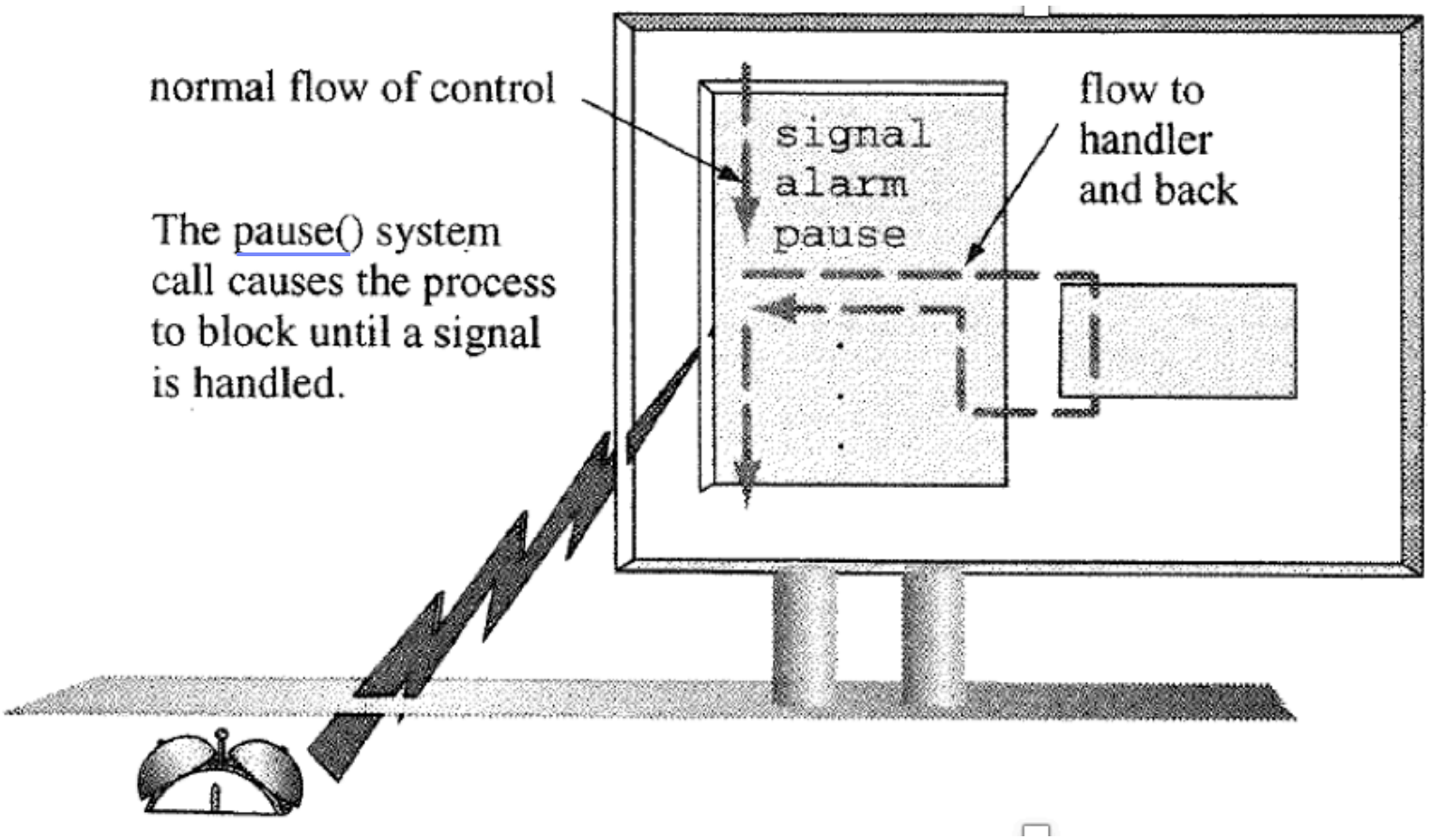
How the sleep function works:

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

Ex6: sleep1.c

```c
/* sleep1.c
 *  purpose show how sleep works
 *  usage   sleep1
 *  outline sets handler, sets alarm, pauses, then returns
 */

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

int main(void)
{
    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*/
    return 0;
}


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

PROGAMMING WITH TIME I : ALARMS



PROGAMMING WITH TIME I : ALARMS

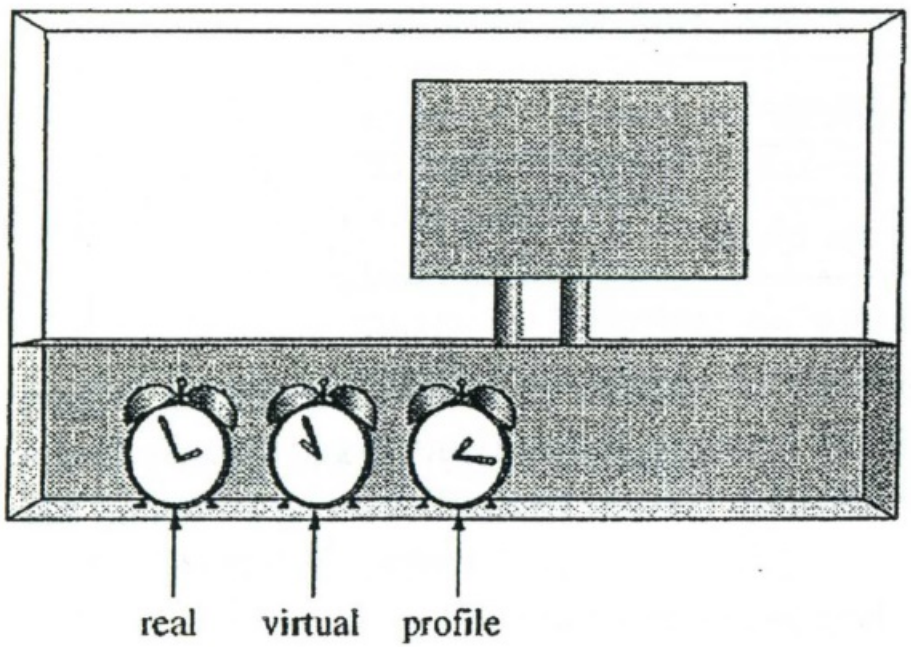
alarm	
PURPOSE	Set an alarm timer for delivery of a signal
INCLUDE	#include<unistd.h>
USAGE	unsigned old = alarm(unsigned seconds)
ARGS	seconds - how long to wait
RETURNS	-1 if error old time left on timer

alarm(0) : stop. alarm

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

PROGAMMING WITH TIME 2: INTERVAL TIMERS

- Each process has three interval timers
  - Real timer, Virtual timer, Profile timer
  - Each timer has two settings
    - The time until the first alarm
    - The interval between repeating alarms

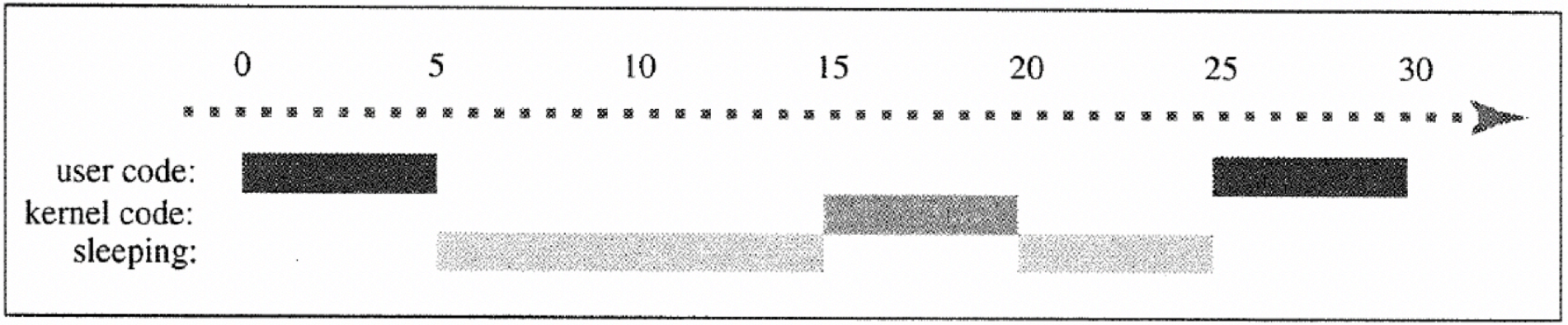


Every process has three timers.

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

PROGAMMING WITH TIME 2: INTERVAL TIMERS

- Each process has three interval timers
  - **Real timer :** counts elapsed time --> CPU time + IO time + waiting time
  - **Virtual timer :** counts elapsed time used by the process. --> CPU time
  - **Profile timer :** counts both elapsed time used by the process and by system calls on behalf of the process.



times:	real	virtual (user mode)	prof (user+kernel mode)

PROGAMMING WITH TIME 2: INTERVAL TIMERS

- Each process has three interval timers
  - **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

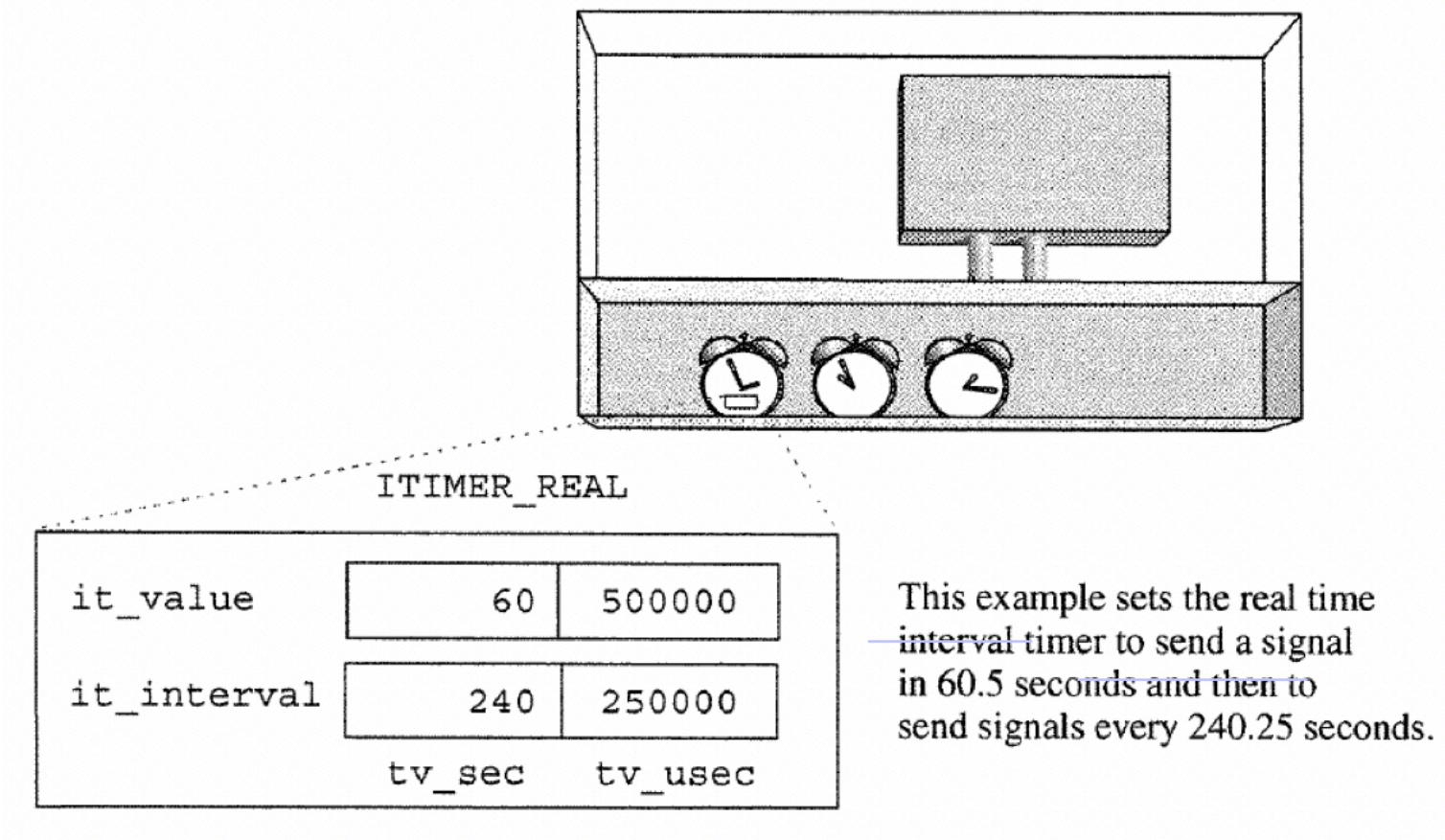
- Programming with the Interval Timers
  - Decide on an **initial interval** and a **repeating interval**
  - Set values in a **struct itimerval**
    - Initial interval and repeating interval
  - Pass the structure to the timer by calling **setitimer**

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 */
};
```

PROGAMMING WITH TIME 2: INTERVAL TIMERS



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

Ex7: ticker_demo.c (1/2)

```c
#include    <unistd.h>  //for pause()
#include        <stdio.h>
#include        <sys/time.h>
#include        <signal.h>
#include        <stdlib.h>  // for exit()

int set_ticker( int );
void  countdown(int);

int main(void)
{
    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);
    }
}
```

Ex7: ticker_demo.c (2/2)

```
36 /*
37  * set_ticker( number_of_milliseconds )
38  *      arranges for interval timer to issue SIGALRM's at regular intervals
39  *      returns -1 on error, 0 for ok
40  *      arg in milliseconds, converted into whole seconds and microseconds
41  *      note: set_ticker(0) turns off ticker
42  */
43 int set_ticker( int n_msecs )
44 {
45     struct itimerval new_timeset;
46     long    n_sec, n_usecs;
47
48     n_sec = n_msecs / 1000 ;            /* int part*/
49     n_usecs = ( n_msecs % 1000 ) * 1000L ;      /* remainder*/
50
51     new_timeset.it_interval.tv_sec  = n_sec;        /* set reload       */
52     new_timeset.it_interval.tv_usec = n_usecs;      /* new ticker value */
53     new_timeset.it_value.tv_sec     = n_sec  ;      /* store this       */
54     new_timeset.it_value.tv_usec    = n_usecs ;     /* and this         */
55
56     return setitimer(ITIMER_REAL, &new_timeset, NULL);
57 }
```

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.

Contents

Signal

- *Handling interruptions* is an essential part of an operating system and of system programs.
  - Unix refers to software interruptions as signals.
- Kernel sends signals to a process *in response to a variety of events*.
  - certain user keystrokes,
  - Illegal process behavior, and
  - Elapsed timers\
- A process calls `signal` to select one of three responses to a signal:
  - Default action (usually termination), signal (SIGALRM, SIG_DFL)
  - Ignore the signal, signal (SIGALRM, SIG_IGN)
  - Invoke a function, signal (SIGALRM, handler)

Handling Multiple Signals

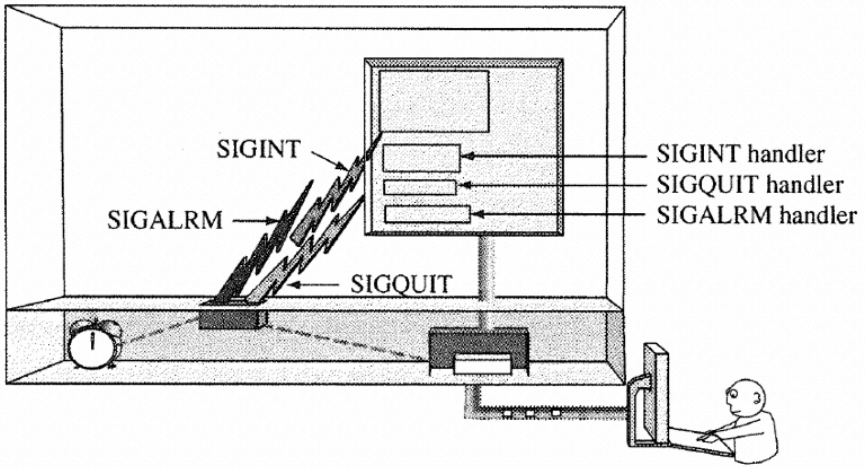
- Original signal model works fine if only *one signal* arrives.
- What happens when *multiple signals* arrive?
  - For *termination* and *ignore* responses, the outcome is clear.
  - For the *invoke a function* response, the answer is neither clear nor consistent.
- The Mousetrap Problem
  - A signal handler is like a mousetrap.
  - Ex) a `SIGINT` handler


```
void handler(int s)
{
    /* process is vulnerable here */

    signal(SIGINT, handler);     /* reset handler */
    ...                          /* do work here  */
}
```

  - The original signal handling is unreliable.

Handling Multiple Signals

- Multiple Signals for a Process
  - How does a process respond to multiple signals?



- Is the handler disabled after each use? (Mousetrap model)
- What happens if a SIGY arrives while the process is in the SIGX handler?
- What happens if a second SIGX arrives while the process is still in the SIGX handler? Or a third SIGX?
- What happens if a signal arrives while the program is blocking on input in getchar or read?

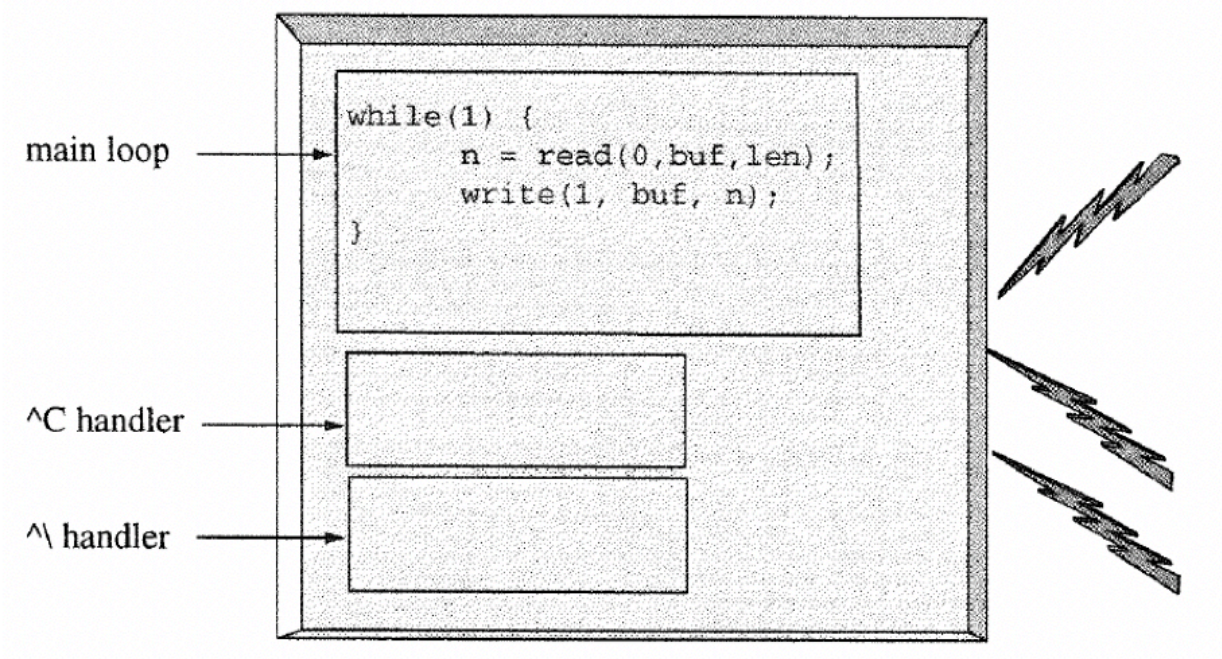
Ex8: Testing Multiple Signals

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

#include <string.h> //gsjung
#include <unistd.h> //gsjung

#define   INPUTLEN 100

int 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 );

    return 0;
}
```

Ex8: Testing Multiple Signals

```
33 void inthandler(int s)
34 {
35 // gsjung
36 //  static int num = 0;
37 //  signal(SIGINT, inthandler);
38 //  printf("inthandler %d\n", ++num);
39
40      printf(" Received signal %d .. waiting\n", s );
41      sleep(2);
42      printf("  Leaving inthandler \n");
43 }
44
45 void quithandler(int s)
46 {
47      printf(" Received signal %d .. waiting\n", s );
48      sleep(3);
49      printf("  Leaving quithandler \n");
50 }
```

Ex8: Testing Multiple Signals



- sigdemo3.c
- Compile and execution

(a) ^C^C^C^C

(b) ^\^C^\^C

(c) hello^C *Return*

(d) hello *Return* ^C

(e) ^\^\hello^C

Ex8: Testing Multiple Signals

- Results of these experiments show how your system handles combinations of signals:
  - Unreliable signals (mousetrap)
    - If two `SIGINT`s kill the process, you have unreliable signals
    - If multiple `SIGINT`s do not kill the process, handlers stay in effect after being invoked.
  - `SIGY` interrupts `SIGX` handler
    - The program first jump to `inthandler`, then to `quithandler`, then back to `inthandler`, then back to the main loop
  - `SIGX` interrupts `SIGX` handler
    - Recursively, call the same handler.
    - Ignore the second signal
    - Block the second signal until done handling the first
  - Interrupted System Calls
    - Programs often receive signals while waiting for input.
    - hel^Clo Return

More Signal Weaknesses

- You Do Not Know Why the Signal Was Sent
  - **The original model** tells handlers which signal invoked it, but **does not tell** the handler **why** the signal was generated
  - Ex) **Floating-point exception** can be generated for several types of arithmetic errors:
    - dividing by zero, integer overflow, and *floating-point underflow*.
  - A handler needs to know about the cause of the problem.

More Signal Weaknesses

- You cannot safely block other signals while in a handler
  - Ex) A program to ignore `SIGQUIT` when it responds to `SIGINT`.

```
void inthandler(int s)
{
    int rv ;
    void (*prev_qhandler)();                    /* holds prev handler */

    prev_qhandler = signal(SIGQUIT, SIG_IGN); /* ignore QUIT's */
    ...
    signal( SIGQUIT, prev_qhandler );          /* restore handler */
}
```

  - Two problems
    - There is *a window of vulnerability* between the call to `inthandler` and the call to `signal`.
    - We do not want to ignore `SIGQUIT` , *we just want to block* it until the fire alarm is over.

Contents

Handling Signals: sigaction

- The POSIX replacement for signal

sigaction	
PURPOSE	Specify handling for a signal
INCLUDE	#include<signal.h>
USAGE	res = sigaction(int signum, const struct sigaction *action, struct sigaction *prevaction);
ARGS	signum signal to handle action pointer to struct describing action preaction pointer to struct to receive old action (if not null)
RETURNS	-1 on error 0 on success

Handling Signals: sigaction

- Customized Signal Handling

```
struct sigaction {
 /* use only one of these two */

 void     (*sa_handler)( int );  /* SIG_DFL, SIG_IGN, or function   */
 void (*sa_sigaction)(int, siginfo_t *, void *);  /* NEW handler   */

 sigset_t sa_mask;               /* signals to block while handling */
 int      sa_flags;              /* enable various behaviors        */
}
```

Handling Signals: sigaction

- Customized Signal Handling
  - **`sa_flags`**

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`. If the `sa_sigaction` value is used, that handler function is passed not only the signal number, but also pointers to structs containing information about why and how the signal was generated.

  - **`sa_mask`**
    - This value contains ***a set of signals to block*** while in the handler.
    - Blocking signals is *an important technique for preventing data corruption*. We examine this topic in detail in the next section.

Ex9: sigactdemo.c

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

#define   INPUTLEN   100

int main(void)
{
    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);
        }

    return 0;
}
```

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

Ex9: sigactdemo.c

- Execution
  - Ctrl-C Ctrl-\
    - *the quit signal will be blocked*

      until the handler for the interrupt signal completes.
  - Ctrl-C Ctrl-C
    - *the program will be killed by the second one.*
  - *If you prefer to catch all Ctrl-C's*,
    omit the `SA_RESETHAND` mask from `sa_flags`.

Contents

Critical Sections

- A section of code to which interruption can produce incomplete or damaged data.

- The simplest way to protect critical sections is *to block* or *ignore signals that call handlers that use or change the data.*

Blocking Signals: `sigprocmask()` and `sigsetops`

- You can block signals at the *signal-handler level* and the *process level*.

- Blocking Signals in a Signal Handler
  - Set the `sa_mask` member of the `struct sigaction` you pass to `sigaction` system call, when you install the handler.
  - `sa_mask` is of type `sigset_t`, a set of signals.

- Blocking Signals for a Process
  - *A process has, at all times, a set of signals it is blocking, signal mask*. Not ignoring, but blocking.
  - To modify that set of blocked signals, use `sigprocmask`.

Blocking Signals: `sigprocmask()` and `sigsetops`

sigprocmask	
PURPOSE	Modify current signal mask
INCLUDE	#include<signal.h>
USAGE	res = sigprocmask(int how, const sigset_t *sigs, sigset_t *prev);
ARGS	how how to modify the signal mask sigs pointer to list of signals to use prev pointer to list of previous signal mask (if not null)
RETURNS	-1 on error 0 on success

how : SIG_BLOCK, SIG_UNBLOCK, or SIG_SET
(adding to, removing from, or replacing it with the signals in **sigs*)

If *prev* is not null, the previous signal mask is copied to **prev*.

Blocking Signals: `sigprocmask()` and `sigsetops`

- Building Signal Sets with `sigsetops`

```
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 to by *setp*.

```
sigdelset(sigset_t *setp, int signum)
```

Remove *signum* from the set pointed to by *setp*.

Blocking Signals: `sigprocmask()` and `sigsetops`

- Example: Temporarily Blocking User Signals
  - Blocking `SIGINT` and `SIGQUIT` temporarily

```
sigset_t  sigs, prevsigs;                    /* define two signal sets */

sigemptyset( &sigs );                        /* turn off all bits      */
sigaddset( &sigs, SIGINT );                  /* turn on SIGINT bit     */
sigaddset( &sigs, SIGQUIT );                 /* turn on SIGQUIT bit    */
sigprocmask( SIG_BLOCK, &sigs, &prevsigs);   /* add that to proc mask  */
// .. modify data structure here ..
sigprocmask( SIG_SET, &prevsigs, NULL);      /* restore previous mask  */
```

Reentrant Code: Dangers of Recursion

- *A signal handler*, or *any function* for that matter, that can be called when it is already active and not cause any problems *is said to be reentrant*.
- `sigaction` allows you
  - to turn on *recursive handling*
    - by *setting the* `SA_NODEFER` *flag* and
  - to turn on *blocking*
    - by *clearing the* `SA_NODEFER` *flag*.
- How do you choose?
  - If the handler is not reentrant, you must use blocking.
  - But if you block signals, you can lose signals.
    - Those signals may be significant; is it safe to miss some?

Contents

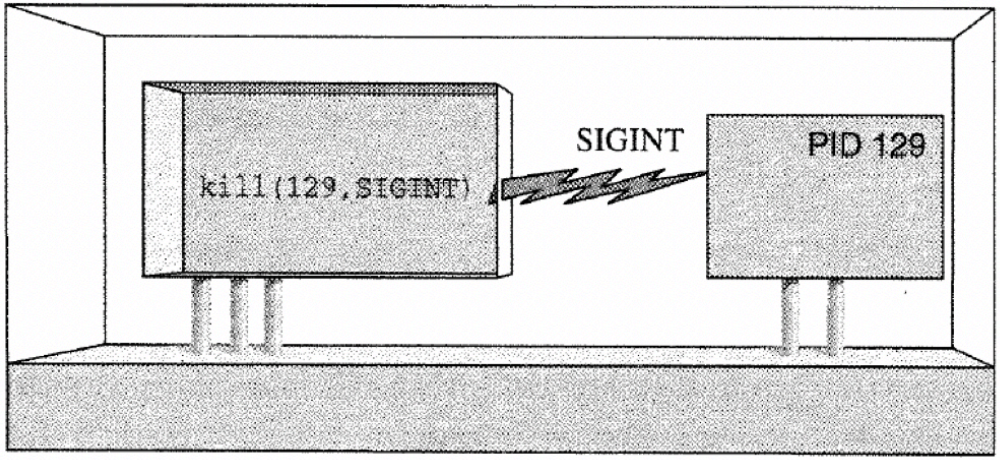
Sending Signals from a Process

- A process sends a signal to another process by using the `kill` system call:

kill	
PURPOSE	Send a signal to a process
INCLUDE	#include <sys/types.h> #include <signal.h>
USAGE	int kill(pid_t pid, int sig)
ARGS	pid process id of target sig signal to throw
RETURNS	-1 on error 0 on success

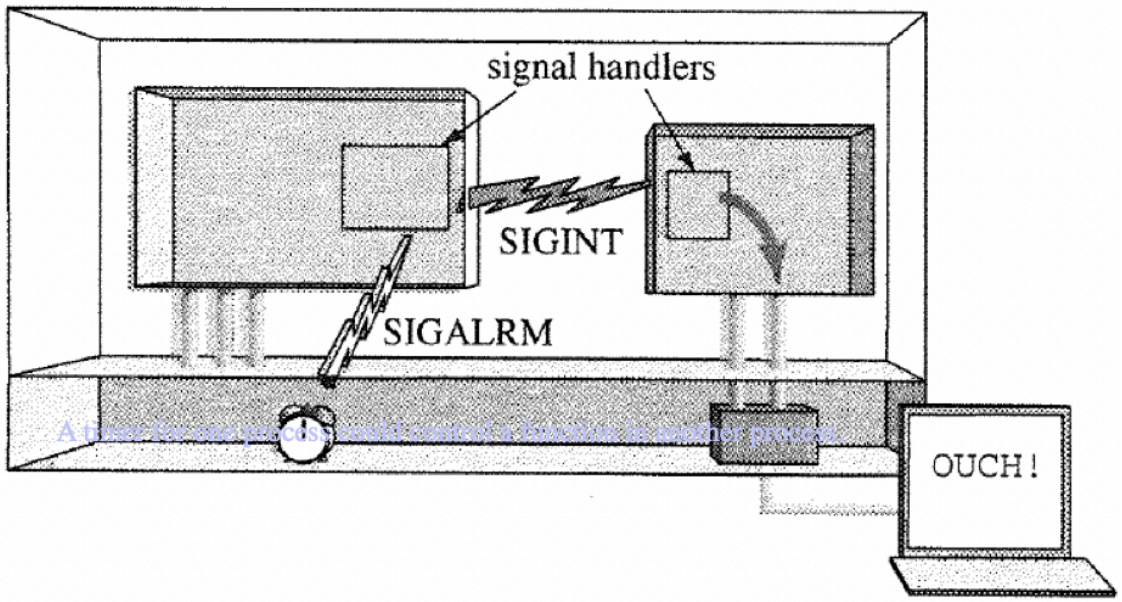
Sending Signals from a Process

- 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*.
- A process may send signals *to itself*.
- A process can send *any signal* to another process, including signals that usually come
  - from the keyboard,
  - from timers, or
  - from the kernel.
- The `kill` command uses the `kill` system call.



Sending Signals from a Process

- Implications for Interprocess Communication



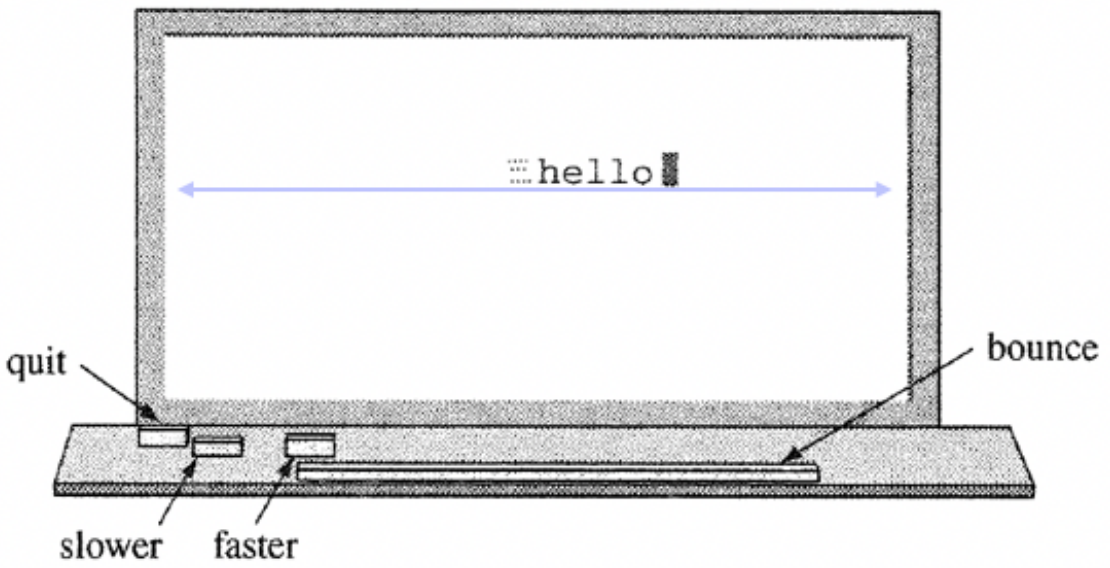
A timer for one process could control a function in another process.

- Signals Designed for IPC: `SIGUSR1`, `SIGUSR2`
  - Two signals for custom applications
  - have no predefined role
  - We look at techniques for interprocess communication in later chapters.

Contents

USING TIMERS AND SIGNALS: VIDEO GAMES

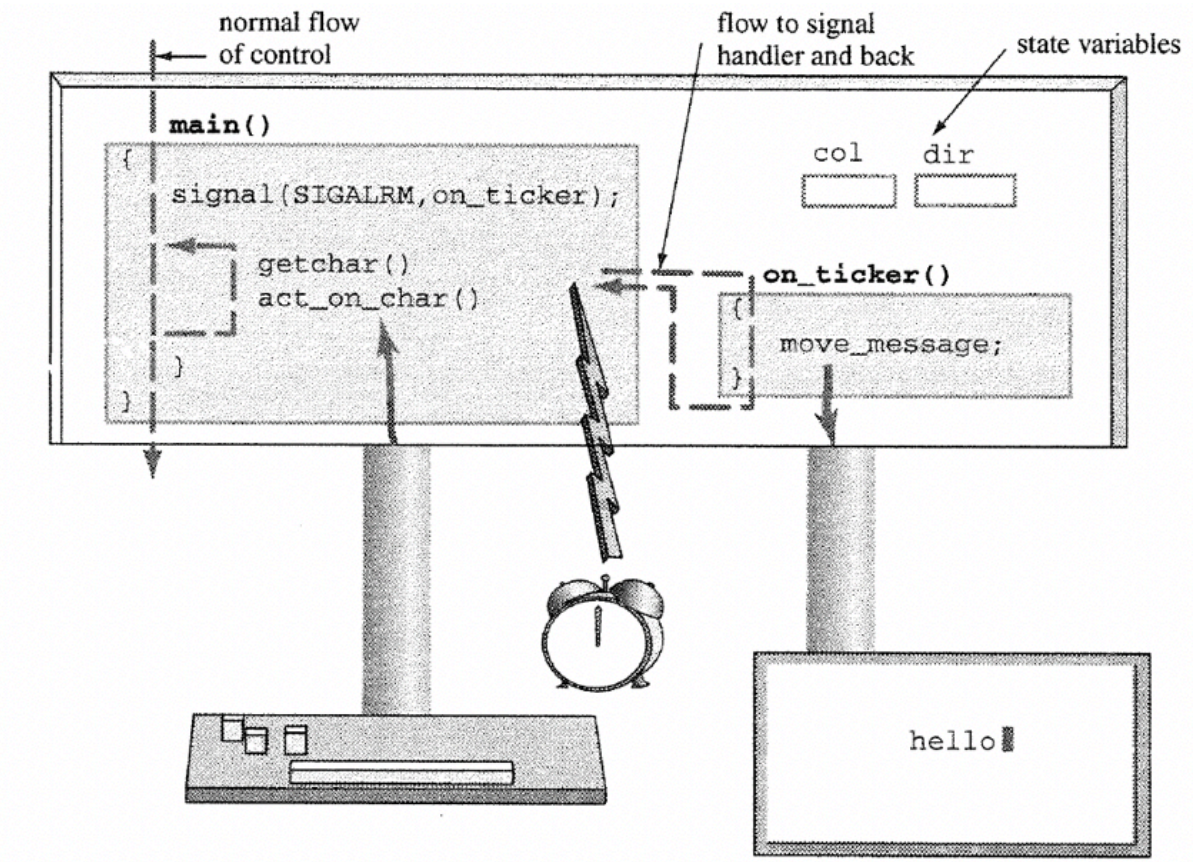
- The game has two main elements: *animation* and *user input*.
  - The animation has to continue smoothly, and
  - the user input has to modify the motion.



space bar : the message reverses direction
s and f : make the message move slower and faster
Q : quits the game

`bounce1d`: Controlled Animation on a Line

- `bounce1d` contains two important ideas: *state variables* and *event handling*.
  - the state variables
    - *position*, *direction*, *delay(speed)*
  - events that modify these state variables
    - *timer ticks*, *user input*



Ex10: bounce1d.c

```c
#include  <stdio.h>
#include  <curses.h>
#include  <signal.h>
#include  <string.h>
#include  <termios.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   */

void set_cr_noecho_mode(void);
int set_ticker(int);
```

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

    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 );

    while(1)
    {
        ndelay = 0;
        c = getchar();
        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;
}
```

Ex10: bounce1d.c

Compile

$gcc bounce1d.c set_ticker.c -lcurses -o bounce1d

```
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;
}

void set_cr_noecho_mode(void)
/*
 * purpose: put file descriptor 0 into chr-by-chr mode and noecho mode
 *  method: use bits in termios
 */
{
    struct termios ttystate;

    tcgetattr( 0, &ttystate);       /* read curr. setting   */
    ttystate.c_lflag    &= ~ICANON; /* no buffering     */
    ttystate.c_lflag    &= ~ECHO;   /* no echo either   */
    ttystate.c_cc[VMIN]     = 1;        /* get 1 char at a time */
    tcsetattr( 0, TCSANOW, &ttystate);  /* install settings */
}
```

bounce2d: Animation in Two Dimensions

- bounce2d uses the same three-part design of bounce1d.
  - Timer Driven
  - Keyboard Blocked
  - State Variables

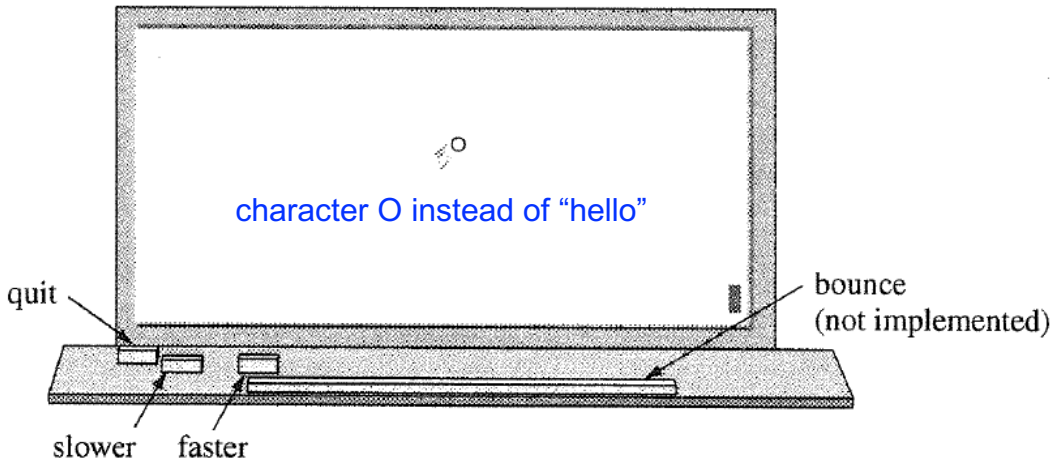
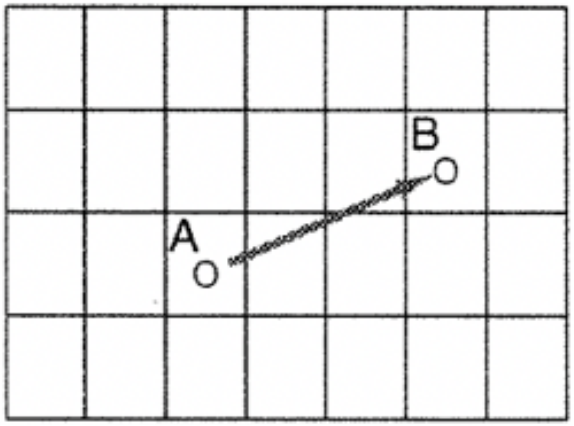


FIGURE 7.21

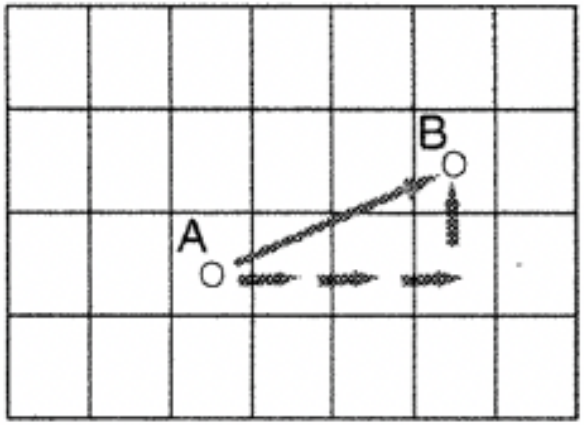
Animation in two directions.

bounce2d: Animation in Two Dimensions

- How Does the Ball Move along a Diagonal?



Question: How to move 'O' from cell A to cell B smoothly?



To approximate diagonal motion:

move right every two timer ticks, and
move up every six timer ticks.

This technique requires two counters, one to count timer ticks for horizontal motion and one to count timer ticks for vertical motion.

bounce2d: Animation in Two Dimensions

- This Sounds like Two Timers
  - A program only has one real-time interval ticker, so we need to build two timers of our own and use the interval timer to count down each of our timers.

- The Code
  - **two counters** to serve as timers
  - Each of those counters has ttg and ttm.
    - the number of ticks to go before the next redraw
    - the number of ticks to move, that is, the interval between each redraw

Ex11 : bounce.h

```
/* bounce.h          */
/* some settings for the game   */

#define BLANK       ' '
#define DFL_SYMBOL  'o'
#define TOP_ROW     5
#define BOT_ROW     20
#define LEFT_EDGE   10
#define RIGHT_EDGE  70

#define X_INIT      10  /* starting col     */
#define Y_INIT      10  /* starting row     */
#define TICKS_PER_SEC   50  /* affects speed    */

#define X_TTM       5
#define Y_TTM       8


/** the ping pong ball **/

struct ppball {
    int y_pos, x_pos,
        y_ttm, x_ttm,
        y_ttg, x_ttg,
        y_dir, x_dir;
    char    symbol ;
} ;
```

Ex11 : bounce2d.c

```c
#include    <stdio.h>
#include    <curses.h>
#include    <signal.h>
#include    <string.h>
#include    <termios.h>

#include    "bounce.h"

void set_cr_noecho_mode();

int set_ticker(int);
int bounce_or_lose(struct ppball*);


struct ppball the_ball ;

/**  the main loop  **/
void set_up();
void wrap_up();

int main(void)
{
    int c;
    set_up();
    while ( ( c = getchar()) != 'Q' ){
        if ( c == 'f' )      the_ball.x_ttm--;
        else if ( c == 's' ) the_ball.x_ttm++;
        else if ( c == 'F' ) the_ball.y_ttm--;
        else if ( c == 'S' ) the_ball.y_ttm++;
    }

    wrap_up();

    return 0;
}
```

Ex11 : bounce2d.c

```
void set_up(void)
/*
 *  init structure and other stuff
 */
{   void    ball_move(int);

    the_ball.y_pos = Y_INIT;
    the_ball.x_pos = X_INIT;
    the_ball.y_ttg = the_ball.y_ttm = Y_TTM ;
    the_ball.x_ttg = the_ball.x_ttm = X_TTM ;
    the_ball.y_dir = 1  ;
    the_ball.x_dir = 1  ;
    the_ball.symbol = DFL_SYMBOL ;
    initscr();
    noecho();
    crmode();

    signal( SIGINT , SIG_IGN );
    mvaddch( the_ball.y_pos, the_ball.x_pos, the_ball.symbol  );
    refresh();

    signal( SIGALRM, ball_move );
    set_ticker( 1000 / TICKS_PER_SEC ); /* send millisecs per tick */
}

void wrap_up(void)
{

    set_ticker( 0 );
    endwin();       /* put back to normal   */
}
```

Ex11 : bounce2d.c

```
void ball_move(int signum)
{
    int y_cur, x_cur, moved;

    signal( SIGALRM , SIG_IGN );    /* do not get caught now */
    y_cur = the_ball.y_pos ;    /* old spot      */
    x_cur = the_ball.x_pos ;
    moved = 0 ;

    if ( the_ball.y_ttm > 0 && the_ball.y_ttg-- == 1 ){
        the_ball.y_pos += the_ball.y_dir ;  /* move */
        the_ball.y_ttg = the_ball.y_ttm  ;  /* reset*/
        moved = 1;
    }

    if ( the_ball.x_ttm > 0 && the_ball.x_ttg-- == 1 ){
        the_ball.x_pos += the_ball.x_dir ;  /* move */
        the_ball.x_ttg = the_ball.x_ttm  ;  /* reset*/
        moved = 1;
    }

    if ( moved ){mvaddch( y_cur, x_cur, BLANK );
        mvaddch( y_cur, x_cur, BLANK );
        mvaddch( the_ball.y_pos, the_ball.x_pos, the_ball.symbol );
        bounce_or_lose( &the_ball );
        move(LINES-1,COLS-1);
        refresh();
    }

    signal( SIGALRM, ball_move);    /* for unreliable systems */
}
```

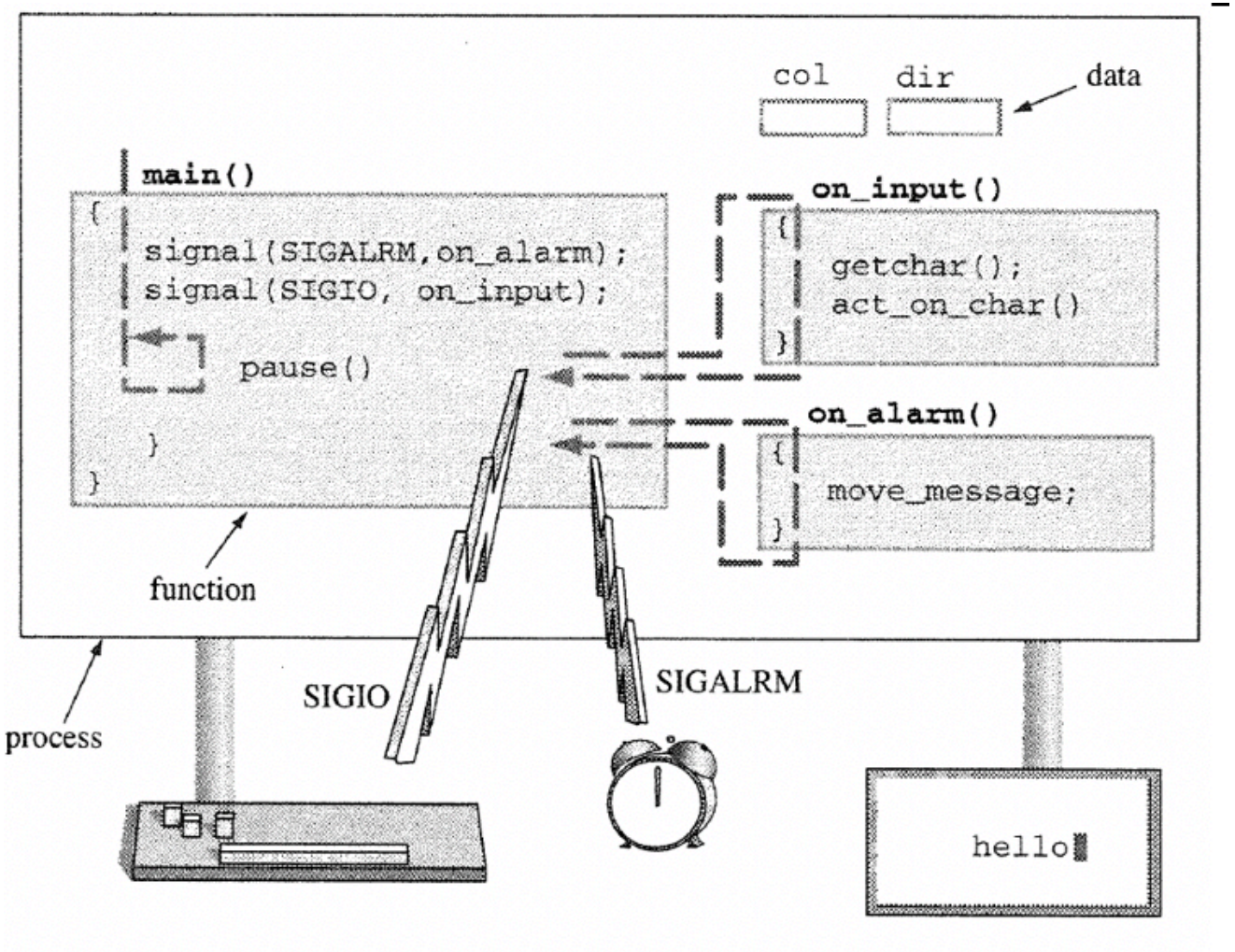
Ex11 : bounce2d.c

```
int bounce_or_lose(struct ppball *bp)
{
    int return_val = 0 ;

    if ( bp->y_pos == TOP_ROW ){
        bp->y_dir = 1 ;
        return_val = 1 ;
    } else if ( bp->y_pos == BOT_ROW ){
        bp->y_dir = -1 ;
            return_val = 1;
    }

    if ( bp->x_pos == LEFT_EDGE ){
        bp->x_dir = 1 ;
            return_val = 1 ;
    } else if ( bp->x_pos == RIGHT_EDGE ){
        bp->x_dir = -1;
            return_val = 1;
    }

    return return_val;
}
```

Contents

Bouncing with Asynchronous I/O

- Instead of blocking, could we be notified of user input with a signal?
  - **Yes! SIGIO**



Method 1: Using O_ASYNC

- Create and install a handler
- Use the F_SETOWN of fcntl
  - Tell the kernel to send input notification signals to our process
- Turn on input signals by calling fcntl to set the O_ASYNC attribute in file descriptor 0
- Execute a simple loop calling pause to wait for signals

Ex12: bounce_async.c

```c
#include    <stdio.h>
#include    <curses.h>
#include    <signal.h>
#include    <fcntl.h>
#include    <string.h>

#include    <unistd.h>
#include    <termio.h>

#define MESSAGE "hello"
#define BLANK   "     "

int row   = 10; /* current row      */
int col   =  0; /* current column   */
int dir   =  1; /* where we are going   *
int delay = 200;    /* how long to wait *
int done  = 0;

void set_cr_noecho_mode(void);
int set_ticker(int);
```

```c
int main(void)
{   void    on_alarm(int);  /* handler for alarm    */
    void    on_input(int);  /* handler for keybd    */
    void    enable_kbd_signals();

    initscr();      /* set up screen */

    set_cr_noecho_mode();

    clear();

    signal(SIGIO, on_input);          /* install a handler        */
    enable_kbd_signals();             /* turn on kbd signals      */
    signal(SIGALRM, on_alarm);        /* install alarm handler    */
    set_ticker(delay);        /* start ticking     */

    move(row,col);                /* get into position      */
    addstr( MESSAGE );            /* draw initial image       */

    while( !done )            /* the main loop */
        pause();

    endwin();

    return 0;
}
```

Ex12: bounce_async.c

```
void on_input(int signum)
{
    int c = getchar();

    if ( c == 'Q' || c == EOF )
        done = 1;
    else if ( c == ' ' )
        dir = -dir;
}

void on_alarm(int signum)
{
    signal(SIGALRM, on_alarm);  /* reset, just in case  */
    mvaddstr( row, col, BLANK );    /* note mvaddstr()  */
    col += dir;             /* move to new column   */
    mvaddstr( row, col, 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;
}
```

Ex12: bounce_async.c

```
/*
 * install a handler, tell kernel who to notify on input, enable signals
 */
void enable_kbd_signals()
{
    int  fd_flags;

    fcntl(0, F_SETOWN, getpid());          /*set io signal to current pid*/
    fd_flags = fcntl(0, F_GETFL);          /*get status of the file*/
    fcntl(0, F_SETFL, (fd_flags|O_ASYNC));/*set status to tty with O_ASYNC*/
}

void set_cr_noecho_mode(void)
{
    struct termios ttystate;

    tcgetattr( 0, &ttystate );
    ttystate.c_lflag    &= ~ICANON;
    ttystate.c_lflag    &= ~ECHO;
    ttystate.c_cc[VMIN] = 1;
    tcsetattr( 0, TCSANOW, &ttystate);
}
```

- **Compile**
  - $gcc bounce_async.c set_ticker.c -lcurses –o bounce_async

Method 2: Using `aio_read`

1. Install `on_input`, the handler we want called when input is read.
2. Set values in a `struct aiocb` to describe
   - what input to wait for and
   - what signal to send when that input is read.
3. Place *a read request* by passing that struct to `aio_read`.
4. We write the handler
   - to get the input character, by calling `aio_return`
   - and then process that character.

Ex13: bounce_aio.c

```c
#include    <unistd.h>
#include    <stdio.h>
#include    <curses.h>
#include    <signal.h>
#include    <aio.h>
#include    <string.h>
#include    <termios.h>

/* The state of the game            */
#define MESSAGE "hello"
#define BLANK   "     "

int row   = 10; /* current row      */
int col   =  0; /* current column   */
int dir   =  1; /* where we are going   */
int delay = 200;    /* how long to wait */
int done  = 0;
struct aiocb kbcbuf;    /* an aio control buf   */

void set_cr_noecho_mode(void);
int set_ticker(int);
```

Ex13: bounce_aio.c

```
void main(void)
{
    void    on_alarm(int);  /* handler for alarm    */
    void    on_input(int);  /* handler for keybd    */
    void    setup_aio_buffer();
    initscr();      /* set up screen */

    set_cr_noecho_mode(); //gsjung

    clear();
    signal(SIGIO, on_input);          /* install a handler        */
    setup_aio_buffer();       /* initialize aio ctrl buff */
    aio_read(&kbcbuf);        /* place a read request     */
    signal(SIGALRM, on_alarm);        /* install alarm handler    */
    set_ticker(delay);        /* start ticking     */
    mvaddstr( row, col, MESSAGE );    /* draw initial image       */
    while( !done )            /* the main loop */
        pause();
    endwin();
}
```

Ex13: bounce_aio.c

```
/*
 *  handler called when aio_read() has stuff to read
 *  First check for any error codes, and if ok, then get the return code
 */
void on_input(int snum)
{
    int  c;
    char *cp = (char *) kbcbuf.aio_buf; /* cast to char * */
    /* check for errors */
    if ( aio_error(&kbcbuf) != 0 )
        perror("reading failed");
    else
        /* get number of chars read */
        if ( aio_return(&kbcbuf) == 1 )
        {
            c = *cp;
            if ( c == 'Q' || c == EOF )
                done = 1;
            else if ( c == ' ' )
                dir = -dir;
        }
    /* place a new request */
    aio_read(&kbcbuf);
}

void on_alarm(int snum)
{
    signal(SIGALRM, on_alarm);  /* reset, just in case  */
    mvaddstr( row, col, BLANK );    /* clear old string */
    col += dir;         /* move to new column   */
    mvaddstr( row, col, MESSAGE );  /* draw new string  */
    refresh();          /* and show it      */
    /*
     * now handle borders
     */
    if ( dir == -1 && col <= 0 )
        dir = 1;
    else if ( dir == 1 && col+strlen(MESSAGE) >= COLS )
        dir = -1;
}
```

Ex13: bounce_aio.c

```
 /*
 * set members of struct.
 *   First specify args like those for read(fd, buf, num)  and offset
 *   Then  specify what to do (send signal) and what signal (SIGIO)
 */
void setup_aio_buffer()
{
    static char input[1];              /* 1 char of input */

    /* describe what to read */
    kbcbuf.aio_fildes     = 0;         /* standard intput */
    kbcbuf.aio_buf        = input;         /* buffer          */
    kbcbuf.aio_nbytes     = 1;              /* number to read  */
    kbcbuf.aio_offset     = 0;              /* offset in file  */

    /* describe what to do when read is ready */
    kbcbuf.aio_sigevent.sigev_notify = SIGEV_SIGNAL;
    kbcbuf.aio_sigevent.sigev_signo  = SIGIO;  /* send sIGIO   */
}

void set_cr_noecho_mode(void)
{
    struct termios ttystate;

    tcgetattr( 0, &ttystate);
    ttystate.c_lflag    &= ~ICANON;
    ttystate.c_lflag    &= ~ECHO;
    ttystate.c_cc[VMIN]     = 1;
    tcsetattr( 1, TCSANOW, &ttystate);
}
```

Do We Need Asynchronous reads for Bounce?

- No.
  - The bounce programs works fine blocking on user input and moving in response to interval timer ticks.

- *The advantage of asynchronous reads* is that
  - *the program does not have to block on input* and can instead *spend its time doing something else (paly music, generate sound effects, compute a complex background image, etc.)*.

Before

```
while( !done )
  pause();
endwin();
```

After

```
compute_pi();
endwin();
```

Term-Project Announcement

- Goal : developing any type of Linux application
- The deadline for your Term Project submission will be 9th December.
- We will have the presentations of the term project.

- **Team members : less than or equal to 3**
- Build your team
  - Let me know the team name and members
  - Let me know what your topic by 30 October.

- Maintain source code with Github

Term-Project Announcement

- Here's list of example program that you could get ideas from.
  - **lftp** - A nice little command-line ftp client.
  - **nmap** - The Network Mapper. A portscanner. Fun.
  - **nethack** - A fun little "storm the dungeon, kill some monsters, collect treasure" game. Warning: adictive.
  - **angband** - Another dungeon game, similar to nethack.
  - **slrn** - A newsreader.
  - **mutt** - An email program.
  - **telnet** - A remote login program.
  - **mysql** - A command-line database front-end to the MySQL database
  - **pico / nano / emacs / vim / jed / joe / ed** - A text editor.
  - **crafty / phalanx / gnuchess** - A chess program.
  - **wget** - A command-line network file retrieval program
  - **lynx / w3m / links** - A text-based web browser.

Term-Project Announcement

- Here's list of example program that you could get ideas from.
  - **finger** - A program that looks up user information over the 'Net.
  - **ispell / aspell** - A spell-checking program.
  - **junkbuster** - An ad-blocking proxy daemon.
  - **tetris-bsd** - A text-based tetris game.
  - **snipes** - Everybody's played Snipes before, right?
  - **nc** - NetCat - The TCP/IP swiss army knife.
  - **top** - System process monitor.
  - **pdmenu** - A text-based, configurable menu program.
  - **queso** - Guess the operating system of a remote machine.
  - **strace** - Traces system calls and signals
  - **talk** - A chat program that allows two users to type messages to each other.
  - **traceroute** - Prints the route packets take to network host
  - **whois** - Client for the whois directory service