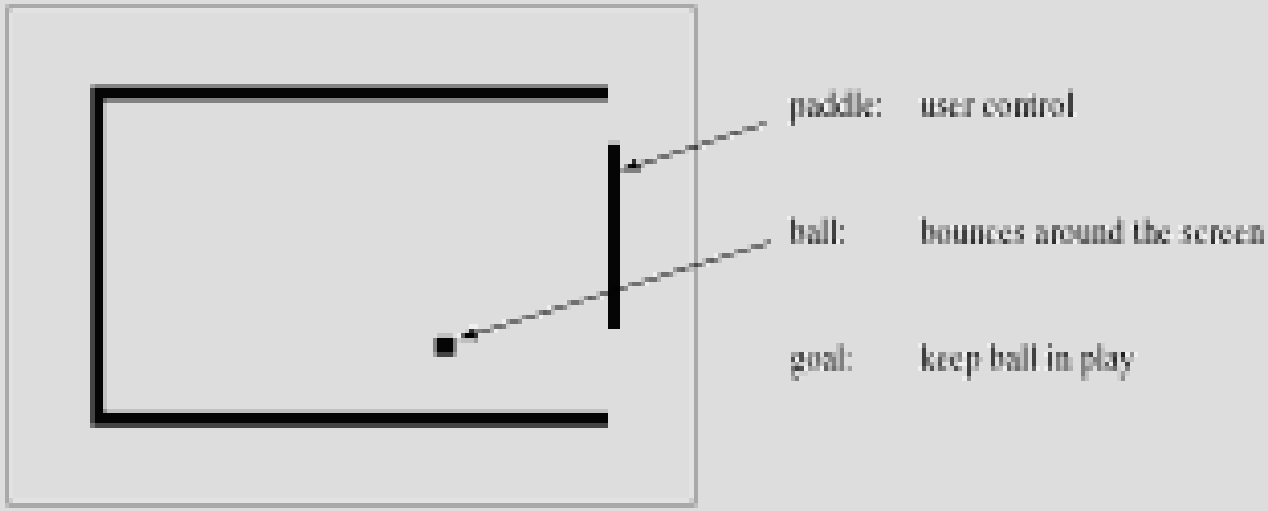
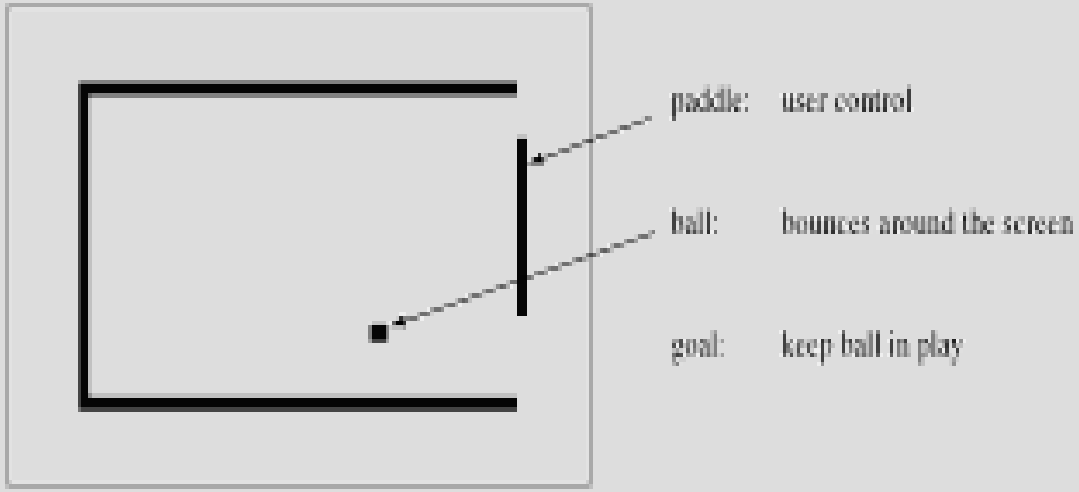
Writing a Pong Game

- screen control
- tty control
- time
- asynchronous program



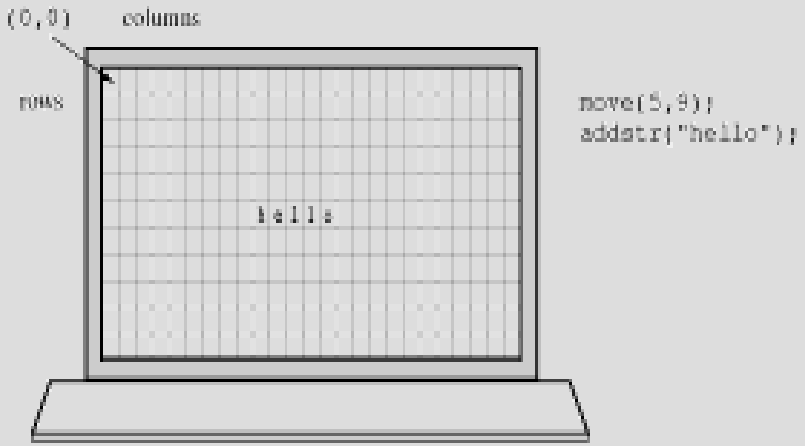
Programming Issues

- Writing to any position on the screen
- Writing at the correct time
- Asynchronous user input and video action



Curses Library

- The curses library is a set of functions that allows a programmer to position the cursor and control the appearance of the text on the terminal screen



Curses Library

- Useful curses functions:
  - initscr()
  - refresh()
  - endwin()
  - move(row,col)
  - addstr(str)
  - addch(c)
  - clear()
  - standout(), standend()

Using Curses

- 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      */
}
```

hello2.c

```
#include  <stdio.h>
#include  <curses.h>
main(){
        int    i;
        initscr();
        /* part2 : do some screen operations      */
                clear();
                for(i=0; i<LINES; i++ ){
                        move( i, i+i );
                        if ( i%2 == 1 ) standout();
                        addstr("Hello, world");
                        if ( i%2 == 1 ) standend();
                }
        /* part3 : push requests to screen    */
                 refresh();  getch();
        /* part4 : wrapup                  */
                endwin();}
```

hello3.c

```
#include  <stdio.h>
#include  <curses.h>
main()
{       int    i;
        initscr();
        /*  do some screen operations */
                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(); }
        /*  wrapup                  */
                endwin();
}
```

Hello4.c

```
#include  <stdio.h>
#include  <curses.h>
main()
[       int   i;
        initscr();
        /*  do some screen operations */
                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();}
```

Hello5.c

```
#include    <curses.h>
#define     LEFTEDGE      10
#define     RIGHTEDGE     30
#define     ROW           10
main()
{       char    msg[] = " Hello ";              /* notice padding spaces */
         int    dir = +1;
         int    pos = LEFTEDGE ;
         initscr();clear();
         while(1){
                 move(ROW,pos);addstr( msg );                   /* draw it */
                 move(LINES-1,COLS-1);          /* park the cursor       */
                 refresh();pos += dir;          /* advance position      */
                 if ( pos >= RIGHTEDGE )        /* check for bounce      */
                         dir = -1;
                 if ( pos <= LEFTEDGE )
                         dir = +1;
                 sleep(1);}}
```

setitimer()

NAME

getitimer, setitimer - get or set value of an interval timer

SYNOPSIS

```
#include <sys/time.h>
int getitimer(int which, struct itimerval *value);
int setitimer(int which, const struct itimerval *value, struct itimer-
    val *ovalue);
```

DESCRIPTION

The system provides each process with three interval timers, each decrementing in a distinct time domain. When any timer expires, a signal is sent to the process, and the timer (potentially) restarts.

ITIMER_REAL decrements in real time, and delivers SIGALRM upon expiration.

ITIMER_VIRTUAL decrements only when the process is executing, and delivers SIGVTALRM upon expiration.

ITIMER_PROF decrements both when the process executes and when the system is executing on behalf of the process. Coupled with ITIMER_VITUAL, this timer is usually used to profile the time spent by the application in user and kernal space. SIGPROF is delivered upon expiration.

Setitimer()

Timer values are defined by the following structures:

```
struct itimerval {
    struct timeval it_interval; /* next value */
    struct timeval it_value;    /* current value */
};
struct timeval {
    long tv_sec;                /* seconds */
    long tv_usec;               /* microseconds */
};
```

setitimer()

Setitimer offer two improvements over alarm:
1) resolution in milliseconds or better
2) pulsing, rather than one shot timers()

The function getitimer fills the structure indicated by value with the current setting for the timer indicated by which (one of ITIMER_REAL, ITIMER_VIRTUAL, or ITIMER_PROF). The element it_value is set to th amount of time remaining on the timer, or zero if the timer is disabled. Similarly, it_interval is set to the reset value. The function setitimer sets the indicated timer to the value in value. If ovalue is nonzero, the old value of the timer is stored there.

setitimer()

Timers decrement from it_value to zero, generate a signal, and reset to it_interval. A timer which is set to zero (it_value is zero or the timer expires and it_interval is zero) stops.

Both tv_sec and tv_usec are significant in determining the duration of a timer.

Timers will never expire before the requested time, instead expiring some short, constant time afterwards, dependent on the system timer resolution (currently 10ms). Upon expiration, a signal will be generated and the timer reset. If the timer expires while the process is active (always true for ITIMER_VIRT) the signal will be delivered immediately when generated. Otherwise the delivery will be offset by a small time dependent on the system loading.

usleep

USLEEP(3) Linux Programmer's Manual USLEEP(3)

NAME

usleep - suspend execution for microsecond intervals

SYNOPSIS

#include <unistd.h>

void usleep(unsigned long usec);

Reentrant Code

A signal handler or and system function that can be called when it is already active without causing a problem is called **reentrant.** Typically functions that use static (ie global) storage locations for routine variables are not reentrant.

curses=ncurses

- No man page for crmode() but it works in program.
- Look in /usr/include/curses.h

```
#define crmode()        cbreak()
#define nocrmode()      nocbreak()
```

man cbreak gives:

curs_inopts(3X) curs_inopts(3X)

NAME

cbreak, nocbreak, echo, noecho, halfdelay, intrflush, keypad, meta, nodelay, notimeout, raw, noraw, noqiflush, qiflush, timeout, wtimeout, typeahead - curses input options

cbreak

Normally, the tty driver buffers typed characters until a newline or carriage return is typed. The cbreak routine disables line buffering and erase/kill character-processing (interrupt and flow control characters are unaffected), making characters typed by the user immediately available to the program. The nocbreak routine returns the terminal to normal (cooked) mode.

Initially the terminal may or may not be in cbreak mode, as the mode is inherited; therefore, a program should call cbreak or nocbreak explicitly. Most interactive programs using curses set the cbreak mode. Note that cbreak overrides raw. [See curs_getch(3X) for a discussion of how these routines interact with echo and noecho.]

Bounce1d.c

- See Bounce1d.c on class web page
  - can multiple signals cause a problem?
  - can we make it reentrant?

Bounce2d.c

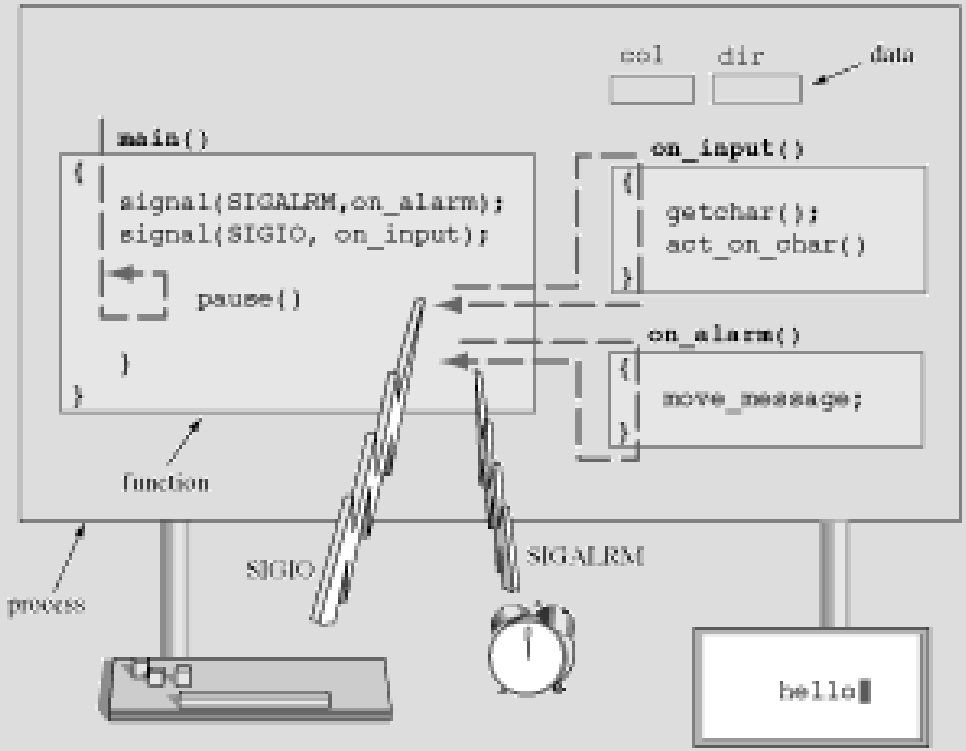
```
struct ppball {
        int     y_pos, x_pos,  /*  ball position*/
                y_ttm, x_ttm,  /*    x,y interrupts per move */
                y_ttg, x_ttg,   /*      x,y interrupt count */
                y_dir, x_dir;   /*      x,y direction to move */
        char    symbol ;

    } ;
```

O_ASYNC

O_ASYNC

Generate a signal (SIGIO by default, but this can be changed via fcntl(2)) when input or output becomes possible on this file descriptor. This feature is only available for terminals, pseudo-terminals, and sockets. See fcntl(2) for further details.

Bounce_async.c



The main loop here is while (1) pause. Is there another way to set this up?

Exercise

- Reaction-Time Tester.
  Write a program that measures how quickly a user can respond. The program waits for a random interval of time then prints a single digit on the screen. The user has to type that digit as quickly as possible. The program records how long it takes the user to respond. Your program should perform 10 such tests and report the minimum, maximum and average response time. (Hint, read the manual page for gettimeofday).

aio_read()

NAME

aio_read - asynchronous read

SYNOPSIS

```
#include <aio.h>
int aio_read(struct aiocb *aiocbp);
```

DESCRIPTION

The aio_read function requests an asynchronous "n = read(fd, bu count)" with fd, buf, count given by aiocbp->aio_filde aiocbp->aio_buf, aiocbp->aio_nbytes, respectively. The return status can be retrieved upon completion using aio_return(3).

The data is read starting at the absolute file offset aiocbp->aio_of set, regardless of the current file position. After this request, t value of the current file position is unspecified.

The "asynchronous" means that this call returns as soon as the request has been enqueued; the read may or may not have completed when the call returns. One tests for completion using aio_error(3).

aio.h

Struct aiocb

```
{int aio_fildes;              /* File desriptor.  */
  int aio_lio_opcode;          /* Operation to be performed.  */
  int aio_reqprio;             /* Request priority offset.  */
  volatile void *aio_buf;      /* Location of buffer.  */
  size_t aio_nbytes;           /* Length of transfer.  */
  struct sigevent aio_sigevent; /* Signal number and value.  */
  /* Internal members.  */
  struct aiocb *__next_prio;
  int __abs_prio;
  int __policy;
  int __error_code;
  __ssize_t __return_value;
#ifndef __USE_FILE_OFFSET64
  __off_t aio_offset;          /* File offset.  */
  char __pad[sizeof (__off64_t) - sizeof (__off_t)];
#else
  __off64_t aio_offset;        /* File offset.  */
#endif
  char __unused[32];
};
```