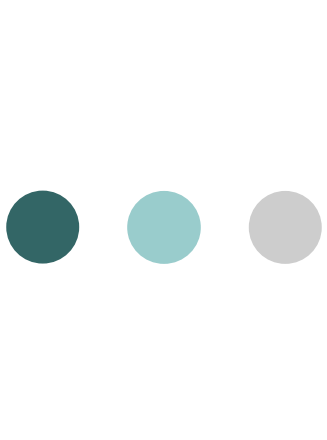
*CSCI3180 Principles of Programming Languages*

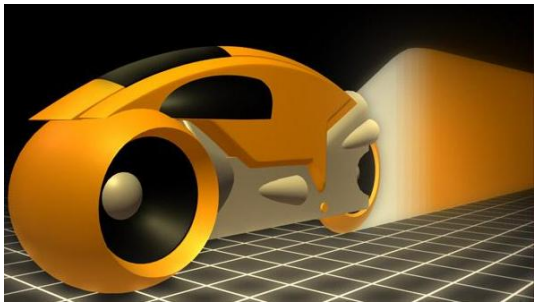
# Assignment 2 and Ncurses Library

Tutorial 4



# Assignment 2

# Assignment 2



- Game simulation (*Light Cycle* in *Tron*)
  - Two light cycles competing in an arena
  - Concurrent programming exercise (Administrator-and-Worker)
- Implementation: the C language
  - GNU/Linux with the GCC compiler
  - User Interface: Ncurses Library
  - Concurrent programming: Synchronous Interprocess Messaging Project for LINUX (SIMPL)

# Light Cycle

- Two kinds of players:
  - Computer  (required)
  - Human (optional, bonus points)
- Initially, light cycles are placed in fixed positions facing each other.
- Walls of light are created behind the cycles as they move

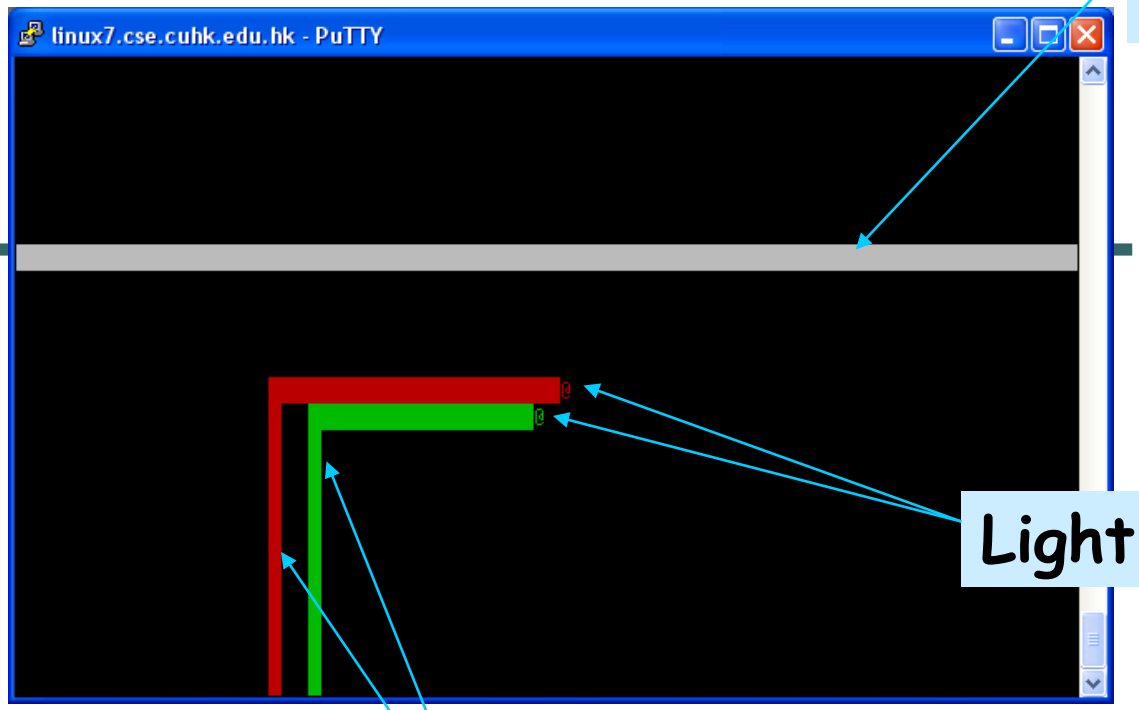
# More Details

- Controls:
  - Turn to East, South, West or North
  - Use turbo boost
- A cycle loses immediately if it collides with the boundary of the arena, the wall created by other cycles or itself.
- All cycles advance simultaneously at a constant speed.
  - Turbo boost can greatly increase the speed of a cycle
- The last cycle remains in the arena wins the game.

# More Details

- The arena
  - A 200x100 rectangle
  - The arena can be much larger than the size of the window
  - Only a part of the arena is visible in the window

# Screen Output



# Ncurses (new curses) Library

- Free software emulation of the original curses library in UNIX
- Provide an efficient API to terminal IO operations: move cursor, create windows, produce colors, etc.
- Need not worry about the underlying terminal capabilities
- Header file: `<curses.h>`
- To compile:

```
gcc prog.c –o prog -lcurses
```

# Initialization and Termination

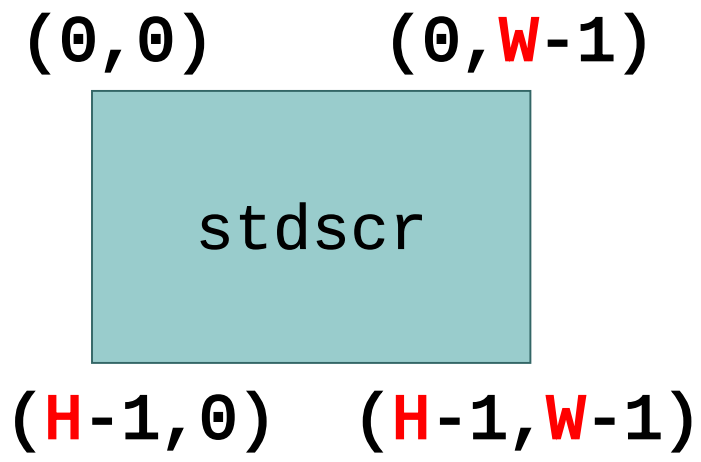
```
#include <curses.h>
int main() {
   initscr();    /*start curses mode*/
   cbreak();     /*disable input buffering*/
   noecho();     /*disable echoing input*/

      /* ... screen handling ... */

   endwin();     /*end curses mode*/
   return 0;
}
```

# Window – The Imaginary Screen

- 2D arrays of characters representing all or parts of screen
- I/O should pertain to specific window
- Coordinates: (y, x)
- Default window: `stdscr`
  - To get the dimension:
    ```
    int H,W;
    getmaxyx(stdscr,H,W);
    ```
  - Changes to window are not reflected until: `refresh();` or `wrefresh(stdscr);`



# Moving Cursor and Output

- Example: say “Hi” to a user in (5,10) of `stdscr`

```
char* user = “Bob”; int y = 5, x = 10;
move(y,x); printw(“Hi %s”,user);
```

- Syntactic sugar:

```
mvprintw(y,x,“Hi %s”,user);
```

- Other output functions:
  - Print a character: `addch` and `mvaddch`
- Remember to `refresh` to see the changes!

# Input

- To read a character from `stdscr`:

  ```
  int ch = getch();
  ```

  和 getchar()不同，getchar()输入完要按回车才输入，而getch()则马上输入

- To capture special keys such as arrow keys:

  ```
  keypad(stdscr,TRUE);
  ```

- Definitions of special keys' integer value:

  `KEY_LEFT`, `KEY_RIGHT`, `KEY_UP`, etc...

- To disable `getch` of waiting for a key hit:

  ```
  nodelay(stdscr,TRUE);
  ```

  - Then, `getch` return `ERR` if no key is hit.

# Other Topics

- Drawing border
- Clear the screen
- Create and destroy new windows
- Colors Handlings
- Etc…
- Read from the online resources

http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/

# Tips

- Drawing the walls
  - The following attribute can be useful in writing the character ▮ :

    `A_REVERSE` `Reverse video`
  - To turn on an attribute:

    ```
    attron(A_REVERSE);
    ```
  - To turn off an attribute:

    ```
    attroff(A_REVERSE);
    ```

- More details can be found on:

http://tldp.org/HOWTO/NCURSES-Programming-HOWTO/attrib.html

# Running the Sample Programs

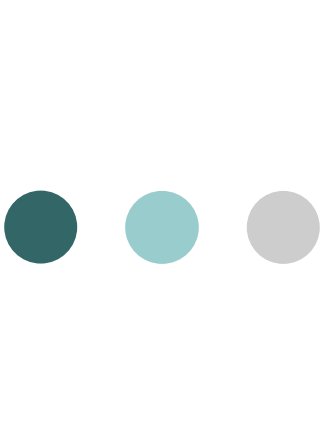
- Download the package from the course webpage
- Follow the instruction on the **README file**
- Important points:
  - Create a new directory where the FIFOs to live e.g. `mkdir $HOME/fifo` (1)
  - Set up the `FIFO_PATH` and `SIMPL_HOME` environment variables vim .cshrc
  - If you forced termination (i.e. pressed Ctrl-c), remember to KILL all your useless processes
  - TIPS: If you got strange problem, make sure the directory pointed by `FIFO_PATH` is clean

# Homework

- Download the sample implementation from the course web page and play around
  - Note: sample programs (both part A and B) can only be run on linux6 – linux9
  - **Part B can only be run** on Linux with **Putty**
- Read the assignment specification
- Do Part A of the assignment using Ncurses
  - Animate a single cycle moving within the border

Note: Part A can be done on either Linux or Sparc machines while Part B must be done on Linux machines

Sparc machines: sparc1 – sparc59
Linux machines: linux6 – linux9

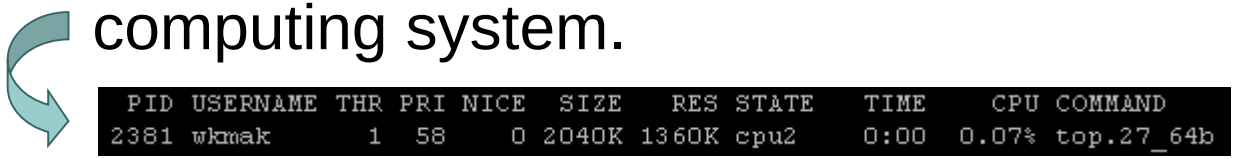


# Concurrent Programming

# Concurrent Programming

- Sequential Process
  - It is a totally ordered set of events
  - Each event being a change of state in a computing system.

In Unix, each process has a Process ID (PID)



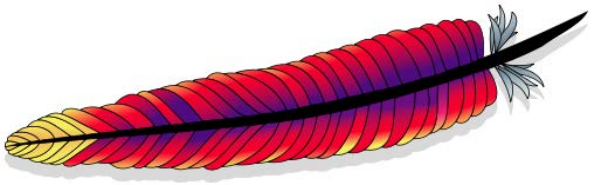

- Sequential Program
  - It is a text that specifies the possible state changes of a sequential process
  - E.g. The FORTRAN / COBOL program in assignment 1

Can we write a program with more than one sequential process?

# Concurrent Programming

- A concurrent program specifies the possible state changes of two or more sequential processes
  - The sequential processes are often cooperating
- Examples of concurrent program


Operating Systems



Web Servers

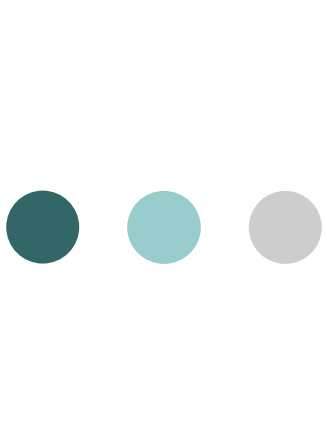
What happens if they are sequential programs?

# Concurrent Programming

- If the set of concurrent processes are to cooperate to achieve a common goal, they need to:
  - Communicate in order to exchange information
    - Processes need to talk to each other
  - Synchronize at certain critical points to ensure proper merging of control
    - A process may need to wait until other processes finish their jobs

Improper synchronization may result in deadlock or starvation!

For more information about deadlock/starvation -> CSCI3100,CSCI3150



# END