

Periféricos y dispositivos de interfaz humana

Práctica 2



**UNIVERSIDAD
DE GRANADA**

Programas de ejemplo

En este apartado explicaremos el funcionamiento de los programas de ejemplo proporcionados en la documentación de la práctica así como enseñar su ejecución.

Hello.c

Este es un programa muy sencillo que lo que hace es imprimir “holita” en la pantalla. Para ello debemos inicializar el modo curses con *initscr()* e imprimir un mensaje en la pantalla el cual no se verá hasta que refresquemos la pantalla con la función *refresh()*. Para poder ver la ejecución usamos *getch()*, esto lo que hace es esperar al programa hasta que el usuario presiona una tecla. Por último terminamos el modo curses y finaliza la ejecución.

```
#include <ncurses.h>
int main()
{
    initscr();
    printw("Holita");
    refresh();
    getch();
    endwin();
    return 0;
}
```



Ventana.c

La ejecución de este programa sigue el siguiente flujo, lo primero que hace es iniciar el modo curses, después comprueba si la terminal puede mostrar colores con el método *has_colors()*, en caso negativo el programa finaliza. Una vez comprobado que la terminal puede usar colores lo que debemos hacer es inicializar el soporte de colores con el método *start_color()* y creamos 3 pares con la función *init_pair()*.

A continuación limpiamos la pantalla y calculamos el número de caracteres que caben en el terminal en este momento con *getmaxyx()* y creamos una nueva ventana con ese mismo tamaño (*newwin()*). Una vez creada la ventana asignamos el par de colores(fondo y caracteres) a la misma, esto se consigue con el método *wbkgd()* y le asignamos los bordes con *box()*.

Finalmente imprimimos una cadena de caracteres en la ventana y en la posición $x = 10$ $y = 10$ y refrescamos la ventana, para ello usaremos *mvwprintw()* y *wrefresh()* respectivamente. cabe destacar que hasta que el usuario no pulse una tecla no termina el programa.

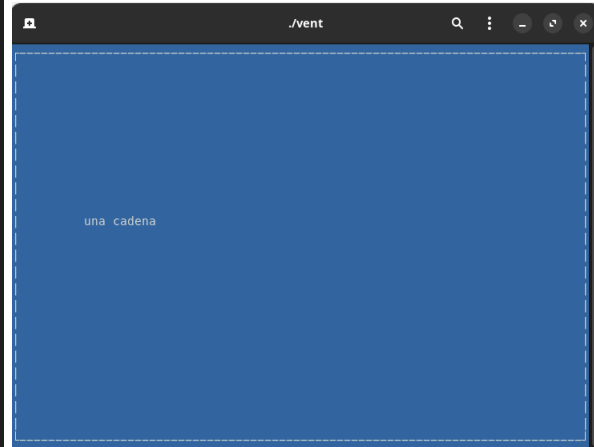
```

#include <stdlib.h>
#include <ncurses.h>

int main(void)
{
    int rows, cols;
    initscr();
    if (has_colors() == FALSE)
    {
        endwin();
        printf("El terminal no tiene soporte de color \n");
        exit(1);
    }
    start_color();
    init_pair(1, COLOR_YELLOW, COLOR_GREEN);
    init_pair(2, COLOR_BLACK, COLOR_WHITE);
    init_pair(3, COLOR_WHITE, COLOR_BLUE); // blanco sobre fondo azul
    clear();
    refresh();
    getmaxyx(stdscr, rows, cols);
    WINDOW *window = newwin(rows, cols, 0, 0);
    wbkgd(window, COLOR_PAIR(3));
    box(window, '|', '-');
    mvwprintw(window, 10, 10, "una cadena");
    wrefresh(window);

    getch();
    endwin();
    return 0;
}

```



Pelotita.c

Este último programa consiste en una “pelota” yendo de un lado a otro de la pantalla haciendo un efecto de rebote.

Lo primero que hacemos es declarar las variables de posición de la pelota, el tamaño de la ventana, la siguiente posición y una variable para la dirección de avance de la pelota.

Acto seguido inicializamos el modo curses y hacemos que no se escriba nada en la terminal con *noecho()* y evitamos que se muestre el cursor con el método *curs_set()* enviando *FALSE* como parámetro.

Lo siguiente que podemos observar es que hay un bucle infinito en que primero se limpia la pantalla, a continuación, se cambia la posición de la pelota imprimiéndola y se refresca la pantalla para que se muestre, después con el método *usleep()* hacemos una espera del tiempo deseado, esto nos va a permitir ver el movimiento de la pelota con una mayor precisión.

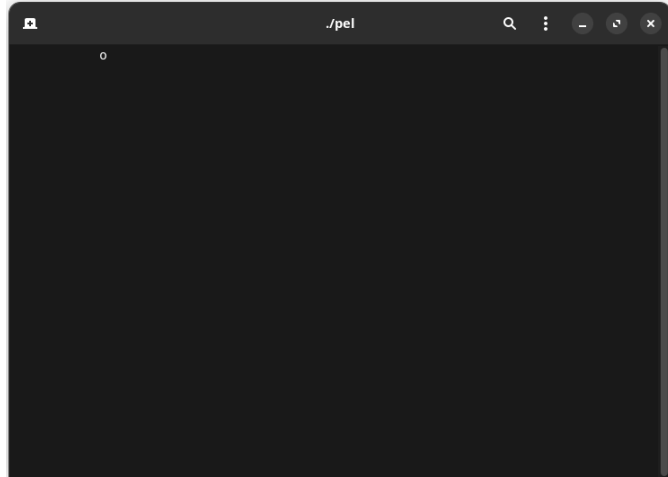
Por último en ese bucle se calcula la siguiente posición de la pelota, en caso de llegar a los límites puestos anteriormente se cambia la dirección de la pelota, creando ese efecto de rebote.

```

#include <ncurses.h>
#include <unistd.h>
#define DELAY 30000
int main(int argc, char *argv[])
{
    int x = 0, y = 0;
    int max_y = 50, max_x = 50;
    int next_x = 0;
    int direction = 1;
    initscr();
    noecho();
    curs_set(FALSE);
    while (1)
    {
        clear();
        mvprintw(y, x, "o");
        refresh();
        usleep(DELAY);
        next_x = x + direction;

        if (next_x >= max_x || next_x < 0)
        {
            direction *= -1;
        }
        else
        {
            x += direction;
        }
    }
    endwin();
}

```



Programa Pong

Para este ejercicio hemos creado un juego tipo “pong” en el que podemos controlar a 2 jugadores y hacemos rebotar una pelota de un lado a otro de la pantalla.

Al igual que en el ejercicio anterior lo primero que hacemos es crear las variables que necesitaremos para nuestro juego como las posiciones iniciales de los jugadores, el tamaño de la ventana, la puntuación, etc.

A continuación se inicializa el modo curses y establecemos una serie de parámetros:

- Evitar que aparezcan en la pantalla los caracteres que tecleamos → *noecho()*
- Hacer que los caracteres obtenidos estén inmediatamente disponibles para el programa, esto nos permitirá mover a los jugadores con una mayor eficiencia → *cbreak()*
- Hacer el cursor invisible → *cursor_set()*

Una vez determinado todo esto limpiamos la ventana y la refrescamos, además de activar la opción que permite que *getch()* pause la ejecución del programa, ésta es *nodelay(stdscr, FALSE)*. A continuación creamos la ventana del tamaño deseado y con un borde compuesto por símbolos “+” creamos la ventana de inicio donde explicamos los controles y las reglas generales.


Una vez el usuario pulse una tecla iniciará el juego, para ello debemos establecer que *getch()* no nos detenga la ejecución mientras obtiene las pulsaciones de los jugadores, esto lo hacemos igual que antes con *nodelay(stdscr, TRUE)*.

Cuando ya hayamos configurado todo debemos crear el bucle del juego, este solo se detendrá cuando pulsemos "y" o uno de los jugadores llegue a 3 puntos. Dentro del bucle lo primero que hacemos es imprimir los "jugadores" que no son más que tres "|" puestos uno encima del otro a modo de portería. Habrá uno a cada lado de la pantalla.

A continuación se refrescará la pantalla y se pondrá un delay deseado. Una vez terminada la espera se pasará a recoger el carácter pulsado y con una serie de condiciones se decidirá que portería se mueve y en que dirección o en última instancia si se finaliza el juego.

Lo siguiente que se calcula es la dirección de la pelota cuando llega al límite de la pantalla. Por una parte si ésta choca contra alguna de las porterías cambiará de dirección, en caso contrario se reseteará la posición de las pelotas y la portería iniciando un nuevo punto, haciendo siempre que la pelota se dirija siempre hacia el jugador que marcó el punto.

Una vez llegados al final del partido, independientemente de la razón, borraremos la ventana creada para el juego con *werase()* e imprimiremos la puntuación de partido, esperando a que el usuario pulse una tecla para finalizar la partida.

Como el código del programa es muy largo no se han puesto capturas, pero se puede encontrar en [pong.c](#) al igual que un video de la ejecución del programa en  [ejecucion pong.mp4](#) .