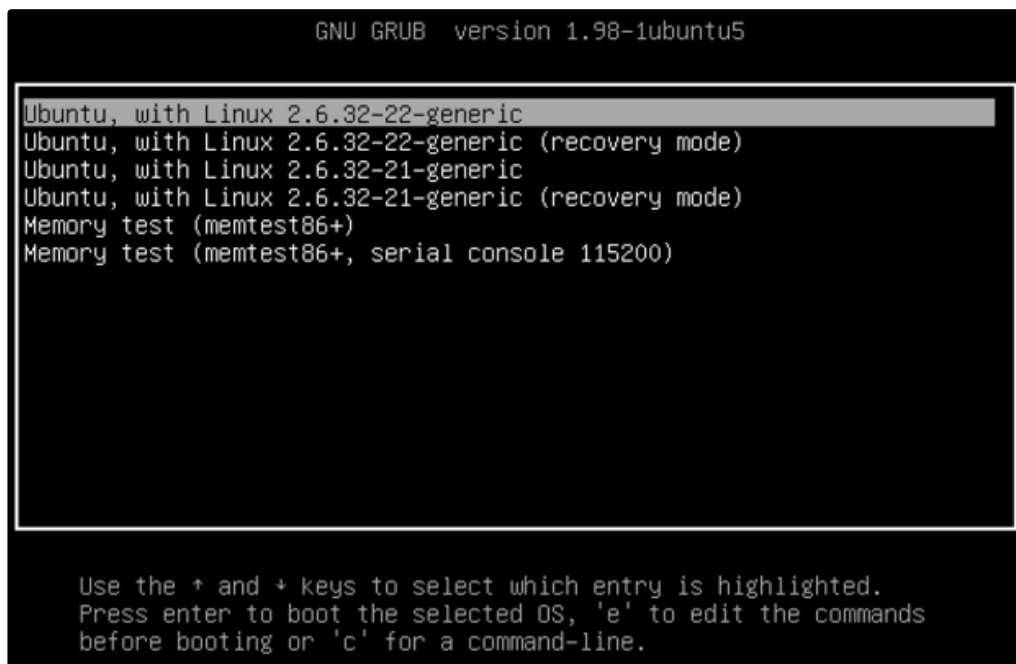


2. Text User Interface (2023 OS Project)

- [A. Install ncurses development package](#)
- [B. Using ncurses](#)
- [C. Printing a colored string at a chosen location on the screen](#)
- [C. Handling Asynchronous Keyboard Events](#)
- [<Exercise and Submission Checklist>](#)
- [Reference](#)

The text user interface (TUI) is the most basic form of the user interface. In C programming, you can use [getchar\(\)](#) to get a character from the input (via stdin) and use [putchar\(\)](#) to output a character to the screen (via stdout). Most likely, you should have learned about `scanf()` and `printf()` in your C programming course for handling strings with your C console program. A limitation of these TUI functions is that the interaction between the user and the computer is linear (sequential). You output a string with `print`, and the output from the next `printf` will follow behind the previous output on the screen.

In fact, the interactions on TUI can be nonlinear. For instance, when you boot up your Linux installation, the GRUB bootloader window allows to use Up/Down key to select the version of kernel you would like to you (assuming you have multiple kernels installed on the system).



To create a TUI similar to the GRUB example, we can use the [ncurses](#) library.

A. Install ncurses development package

```
1 sudo apt-get install ncurses-dev
```

B. Using ncurses

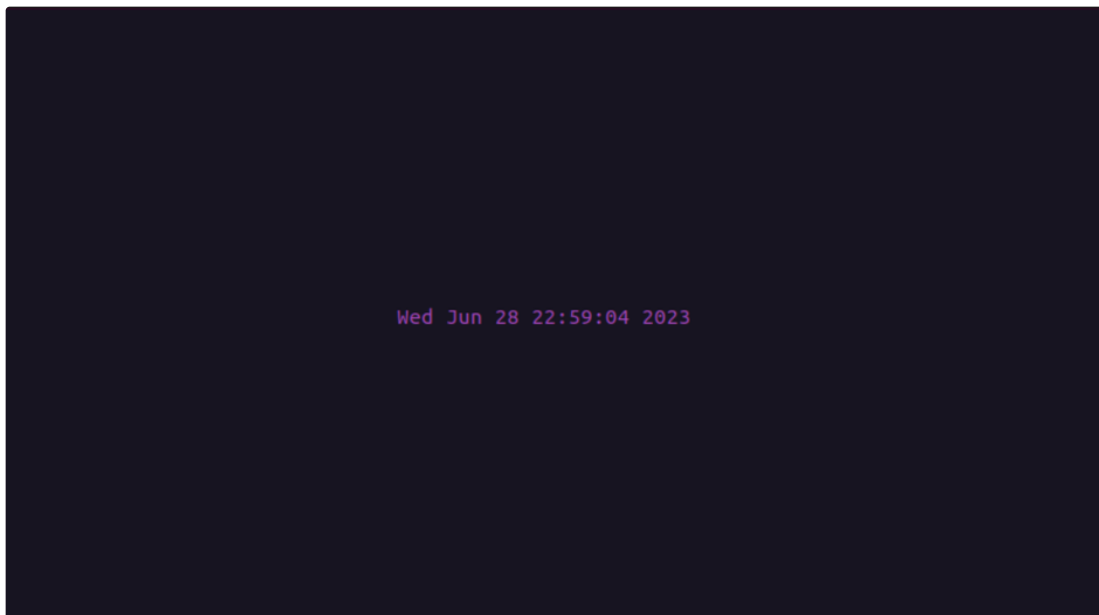
To compile your C/C++ programs using ncurses library, you need to include the curses header file `<ncurses.h>`.

To link the programs you need to use the `-lncurses` option, like

```
gcc prog.c -lncurses
```

C. Printing a colored string at a chosen location on the screen

We will use the following example to walk you through the basic usage of ncurses. The following program will print the current time and date on the screen. You may use Up/Down/Left/Right button to move the time and date message around on the screen. To leave the program, you may press 'q'.



Simple Clock

[Code for Simple Clock]

```
1 #include <ncurses.h>
2 #include <time.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <poll.h>
6 #include <sys/eventfd.h>
7
8 int main()
9 {
10     int color_index;
11     bool running = true;
12     int h, w, ch, y, x, ny, nx;
13
14     time_t t;
```

```

15 char s[64];
16 const char* space = " ";
17 int len = strlen(space);
18
19 initscr();
20 raw(); //enter raw mode
21 keypad(stdscr, TRUE); //enable capturing special keystrokes
22 noecho(); // don't show the typed character on the screen
23 curs_set(0); // hide the cursor
24
25 start_color();
26 init_pair(1, COLOR_RED, COLOR_BLACK);
27 init_pair(2, COLOR_GREEN, COLOR_BLACK);
28 init_pair(3, COLOR_YELLOW, COLOR_BLACK);
29 init_pair(4, COLOR_BLUE, COLOR_BLACK);
30 init_pair(5, COLOR_MAGENTA, COLOR_BLACK);
31 init_pair(6, COLOR_CYAN, COLOR_BLACK);
32 init_pair(7, COLOR_WHITE, COLOR_BLACK);
33 color_index = 0;
34
35
36 nx=ny=x=y=0;
37
38 while (running) {
39
40     wmove(stdscr, y, x);
41     wprintw (stdscr, "%.*s", len, space);
42
43     t = time(0);
44     strftime(s, sizeof(s), "%c", localtime(&t));
45     len = strlen(s);
46
47     getmaxyx(stdscr,h,w);
48     if (ny<0) ny = 0;
49     if (ny>=h) ny = h-1;
50     if (nx+len>=w) nx = w - len;
51     if (nx<0) nx = 0;
52
53
54     attron(COLOR_PAIR(color_index+1) );
55     wmove(stdscr, ny, nx);
56     waddstr (stdscr, s);
57     wrefresh(stdscr);
58     attroff(COLOR_PAIR(color_index+1) );
59
60     color_index = (color_index+1)%7;
61
62     y = ny;
63     x = nx;
64
65     ch = getch();
66
67     switch(ch) {
68         case KEY_UP: ny = y-1;
69             break;
70         case KEY_DOWN: ny = y+1;
71             break;
72         case KEY_LEFT: nx = x-1;

```

```

73         break;
74     case KEY_RIGHT: nx = x+1;
75         break;
76     case 'q':
77     case 'Q':
78         running = false;
79         break;
80
81     default: ;
82 }
83
84 }
85
86 wrefresh(stdscr);
87 endwin();
88
89 return 0;
90 }

```

At Line 19~23, we perform the initialization.

At Line 25~32, we set up the indices of the color pairs (foreground/background) to be used in the program.

At Line 38, we use a while loop to repeat the following steps

(Line 40~41) Print a blank string with length `strlen(space)` at screen location `(y, x)`

(Line 43-45) Get the current date and time. Format the date and time into string `s`. The length of `s` is calculated and stored in `len`.

(Line 47) Get the width `w` and height `h` of the screen

(Line 48-51) We are going to print the time and date string at the screen location `(ny, nx)`. For that, we check if the whole string will be within the boundary of the screen. If not, we will adjust the value of `ny` and `nx` accordingly.

(Line 54) Set the font color to the color indexed by `color_index` (check Line 26-32)

(Line 55) Move the screen cursor to location `(ny, nx)`

(Line 56) Write the string to the screen at the current cursor location.

(Line 57) Refresh the screen (this ensures the content on the screen will become visible on the monitor)

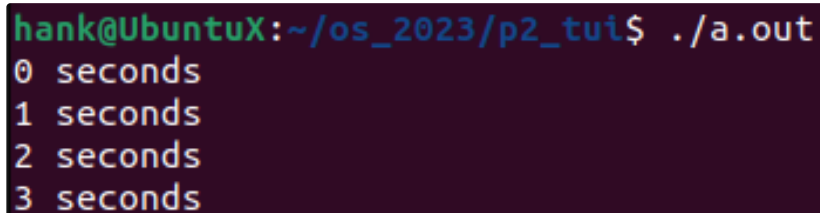
(Line 65) Wait for a keystroke from the keyboard

(Line 67) Perform the actions for the corresponding keystrokes

At Line 86, free up the resources hold by ncurses runtime.

C. Handling Asynchronous Keyboard Events

You may notice that the execution of the Simple Clock program will get blocked on the call to `getch()` at Line 65, waiting for keyboard input. As a result, the clock time on the screen will not be updated (Line 43) until we hit a keystroke. To avoid the wait for keyboard inputs from blocking the execution of the main program, we can leverage asynchronous I/O. Here we will use an example program to demonstrate asynchronous I/O with `poll()`.



```

hank@UbuntuX:~/os_2023/p2_tui$ ./a.out
0 seconds
1 seconds
2 seconds
3 seconds

```

Example of `poll()` usage

The program will output the number of seconds since the start of the program. The output will be updated every 1 sec. The execution continues until we hit a 'q' or 'Q' on the keyboard. The code of the example is given in the following.

[Code for example of poll() usage]

```
1  #include <stdio.h>
2  #include <ncurses.h>
3  #include <unistd.h>
4  #include <poll.h>
5  #include <sys/eventfd.h>
6
7  int main()
8  {
9      int ch;
10     bool running;
11     int seconds;
12     struct pollfd poll_fds[1];
13
14     poll_fds[0].fd = STDIN_FILENO;
15     poll_fds[0].events = POLLIN;
16
17     running = true;
18     seconds = 0;
19
20     initscr();
21
22     while (running) {
23
24         printf("%d seconds\n\r", seconds++);
25
26         if ( poll(poll_fds, 1, 1000) < 0 ) {
27             perror("Error \n");
28             continue;
29         }
30
31         if (poll_fds[0].revents & POLLIN) {
32             ch = getch();
33             switch(ch) {
34                 case 'q':
35                 case 'Q':
36                     running = false;
37                     break;
38                 default: ;
39             }
40         }
41     }
42
43     endwin();
44
45     return 0;
46 }
```

As you can see, we still use the `getch()` (Line 32) to get the keystroke of 'q' or 'Q'. To prevent the `getch()` from waiting for keystrokes, we use `poll(poll_fds, 1, 1000)` (Line 26) to check if a keystroke has been pressed. The call to `poll()` carries three parameters. The `poll_fds` array include the descriptors of the files we would like to check. For the keyboard input, the corresponding file descriptor is `STDIN_FILENO`, which is set at Line 14. The second parameter indicates the number of file descriptors in the `poll_fds` array. Here, we have only the `STDIN_FILENO` descriptor to check, so the value of the second parameter is set to 1. In general, a `poll()` call can check the

input status of multiple files. The third parameter specifies the upper limit on the time for which `poll()` will block, in milliseconds. Here, we put 1000 milliseconds. As a result, the execution of the program will be blocked at Line 26 for 1 second if no keystroke is pressed. If a keystroke is pressed, the call to `poll()` at Line 26 will return immediately. We will check if the return corresponds to the file descriptor (the keyboard) we are interested in at Line 31. And, if that's the case, we will use `getch()` to retrieve the keystroke (so the call to `getch()` would not block). If the keystroke is a 'q' or 'Q', we will end the program by breaking the while loop (Line 36).

<Exercise and Submission Checklist>

For this homework, you need to extend the code of the simple clock so that the clock time on the screen will be updated every second, showing the current time even when no keystroke has been pressed. The program would terminate when the user pressed 'q' or 'Q'.

For your homework submission, please include the following items:

- A1. (80%) Your clock program's source code.
- A2. (20%) A concise report in PDF format that outlines the steps to compile and run your code.
- A3. (Optional) If your code is written in a language other than C/C++, ensure it's straightforward for the TA to test. Consider simplifying the process, for instance, by preparing a Docker pull.
- A4. (Optional) If you believe there may be potential misunderstandings from your written report, you can provide a step-by-step video guide. Please share the video link in your submission.

Reference

- 1. [Position text on your screen in Linux with ncurses | Opensource.com](#)
- 2. [Ncurses programming guide](#)