CENG 448/548 — Real-Time Operating Systems

South Dakota School of Mines & Technology

Laboratory Assignment Four

The objective of this assignment is to port ninvaders to FreeRTOS. Ninvaders is based on ncurses (New Curses), a library for manipulating text terminals. In order to port ninvaders, you you must first port a compatible curses library. For several reasons, ncurses would be very difficult to port. However, there is another version of curses, called PDCurses that can be ported with minimal effort.

Figure 1 shows the application program and the software layers that support it. The UART driver layer was written in the previous lab. The PDCurses layer will be downloaded and used without modification. However, we must provide an interface between the PDCurses library and our UART Driver. This is accomplished by writing a set of functions for driving ANSI terminals, and using that library to provide an ANSI platform for PDCurses. The ANSI Platform layer provides 34 functions that are used by the PDCurses library. The application should not call those functions directly.

The ANSI Terminal Driver layer provides 26 functions that are used by the platform layer, and can also be used by the application. The application must not use these functions to directly access the UART that is being used by PDCurses, but is allowed to use them to access any of the other UARTs on the system.

The UART Driver layer provides 12 functions (plus ISRs) to manage the UART hardware. This layer was written in the previous lab assignemnt. The application can call these functions directly to access the UARTs on the system. Other than initialization and configuration, the the application should not directly access the UART that is being used by PDCurses.

Part 1: Port PDCurses and Run the Demo Programs

1. PDCurses will require two additional functions from your UART driver. Add the function prototypes shown in Listing 1 to `UART_16550.h` and implement them in `UART_16550.c`.

2. Copy your Lab 3 directory into a new directory, or download the instructor's code from D2L.

3. Download the source code for PDCurses (find it on GitHub) and unpack it. You should move it into your main project directory or make a link to it.

4. After running cmake, your main project directory should look like this:

```
bin             CMakeFiles      CMSIS     include   PDCurses  src
CMakeCache.txt  CMakeLists.txt  FreeRTOS  Makefile  scripts
```

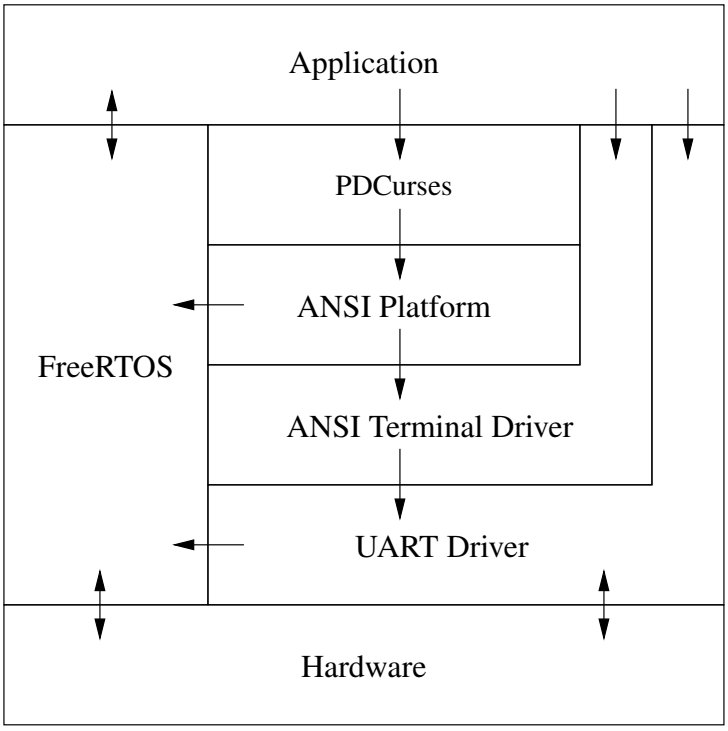

Figure 1: Software/Hardware Layers.

Listing 1: functions to be added to your UART driver.

```
// Return the number of characters available in the receiver stream buffer
int UART_16550_chars_available(int UART_number);

// Flush the UART receiver FIFO and receiver stream buffer
void UART_16550_flush_rx(int UART_number);
```

5. The header file for PDCurses is `PDCurses/curses.h`. Add the the path for the header to your `CMakeLists.txt` file.

6. Most of the C source code is in the the directory named `PDCurses/common`. There are other subdirectories in the PDCurses directory which provide drivers for specific platforms. Add all of the C files in `PDCurses/common` to your `CMakeLists.txt` file. Do not include any code from any of the driver subdirectories. An easy way to include all of the C source files is:

   ```
   file(GLOB PDCURSES_SOURCES
     "${CMAKE_SOURCE_DIR}/PDCurses/pdcurses/*.c"
     )
   ```

   You can then use the variable `PDCURSES_SOURCES` wherever you need it.

7. Copy `firework.c` from the PDCurses `demo` directory into your project `src` directory, and convert its `main` into a FreeRTOS task. You will need to create a header file for it in your `include` directory, similar to the other tasks. Disable the stats and hello world tasks in your `main.c` file, and run the firework task instead. You will need to make several other minor changes to get it to compile without errors or warnings.

   For example, the firework code contains the following line:
   `seed = time((time_t *)0);`
   which requires the `_gettimeofday` function. Replace that line with a constant seed value.

8. During the link step, there will be some unresolved symbols. Those are functions that must be implemented in order to port PDCurses to your FreeRTOS project.

   Read the PDCurses Implementor's Guide, which outlines the steps required in creating a port. The instructor has done most of the work already, and you just need to finish the port. You can download `PDCurses_ANSI_driver.tgz` from D2L and unpack it inside your PDCurses directory.

   - Add the C files in the `ANSI` directory to your `CMakeLists.txt`, and add a path to the directory for the header files.
   - The `ANSI` directory has a subdirectory named `drivers`. Add the C file in that directory to your `CMakeLists.txt`, and add a path to the directory for the header files.
   - The `ANSI/drivers` directory contains a subdirectory called `FreeRTOS`. That directory contains the instructor's implementation of the previous lab assignment, and a driver table that maps those functions to a device-independent driver table. Add the `driver_table.c` file to your CMake project. You will **use your own implementation of the `UART_16550.c` file**, so do not add it to your CMake project.
   - There are a few functions in the ANSI terminal platform code and drivers that have not been completed. Search all of the C files in the ANSI directory for "Insert code here" and complete the platform driver.

9. Once you have the firework task running, modify the stats and hello world tasks so that they use UART1 instead of UART0, and enable them to run. Use a USB/TTL adapter (available in the lab cabinet) to connect to UART1 in a separate terminal window.

10. Make note of The CPU time used by the three tasks, the maximum and minimum times for the `hello world` task, the amount of jitter, and whether or not you detect any corruption of the stats task or firework task output. You will include those observations in your lab report.

11. Replace `firework.c` with `testcurs.c` and test it thoroughly. Note that you may be running low on memory. When running an NCurses program, you should set your terminal to have 80 columns and 25 lines. The amount of RAM used by NCurses depends on how many rows and colems there are in the terminal. You may also have to enable compiler optimization.

12. Try running some of the other test programs that came with PDCurses..

Part 2: Port ninvaders

1. Find and download ninvaders. Unpack it in your main project directory.

2. Edit your `CMakeLists.txt` to add the ninvaders source file(s) and header path(s).

3. Edit `nInvaders.c` to convert it to a FreeRTOS task. There are only three major changes that you must make:

   - Disable the `evaluateCommandLine` function (comment it out).
   - Disable the `finish` function (comment it out).
   - Re-write the `setupTimer` function so that is uses a FreeRTOS software timer instead of a POSIX software timer.

4. Add a header file for the ninvaders task, and edit your `main` function to run it. Try to compile.

5. The nInvaders code is a bit amateurish. You will get lots of "multiply defined symbols". Most of these are generated because the author did not mark global variables as **`extern`** in the header files. For example, the variable `weite` in `nInvaders.h` should have been declared extern. Edit all of the ninvaders header files and mark any global variables as extern. This may cause some variables to no longer exist. Define those variables in the appropriate C files.

6. Provide any missing functions and make modifications until it compiles and runs.

7. You may need to move the heap from the tightly coupled RAM to the DRAM.

8. Modify hello world and stats so that they use some functions from the ANSI device layer to move the cursor around and make their output stay in the same place rather than scrolling down the screen.

9. Record CPU usage and any other information you can gather and include it in your report.