Lab 1 Report

1. Design

1.1 Terminal Device Driver

The first function of this Terminal Device Driver (TDD) is to support multiple terminals to echo the characters entered by the user through the X11 window service to the same terminal window as required. The implementation of this feature is as follows:

- Call the API *ReadDataRegister* (RDR) in hardware.h to capture user input, and implement the *ReceiveInterrupt* handler (RIH) that is called after RDR completes to handle the events that occur when the system receives the input.
- RIH, after being called by RDR, will process the received characters accordingly and add the processed characters into the input buffer storage structure respectively.
- The RIH decides whether to cold-start the cycle by determining whether the system is in the middle of an input and output cycle. Specifically, if the system is busy with a cycle, it adds the received character to the echo buffer structure and passes it to the following program to process the output. Otherwise, it uses *WriteDataRegister* (WDR) to output the character directly.
- If the system is busy with a cycle, in other words, the WDR is busy transmitting characters to the terminal, the WDR waits until it has finished transmitting characters before transmitting the characters in the echo buffer to the terminal.
- The *TransmitInterrupt* handler (TIH) is called when the WDR is done and is responsible for continuing to get and transmit the next character from the buffer, if any. This process involves prioritizing the buffer. In addition, TIH is responsible for notifying other waiting procedures that the data transmit has been completed and the WDR is now idle.

Another feature of TDD that supports user thread calls to the API is an entry procedure for the Mesa monitor. It is implemented as follows:

- The user can view the lines entered on the terminal by calling *ReadTerminal* (RT). RT will be determined by the status:
    - Whether there is another RT occupying the monitor, if so, wait.
    - Whether the current line is finished, if not yet, wait.

  If neither of the above two conditions is satisfied, RT will output the characters in the input buffer.
- The user can input data to the terminal at one time by calling WriteTerminal (WT), which will determine whether there is any other WT occupying the monitor by its status, if so, it

will wait. If not, the WT writes the characters provided by the user into the output buffer, which is an actual storage structure instead of an abstraction.

- The user can obtain a snapshot of the current state of all terminals by calling *TerminalDriverStatistics*(TDS). This information is stored in *struct termstat*. TDS requires the user to provide an array of *struct termstat* to store this information. At the same time, there is also an array of *struct termstat* in a monitor to store and update these state data under appropriate conditions. When the user calls TDS, the array of *struct termstat* in the monitor is copied directly to the user's storage structure.

1.2. Buffer Structure and Management

The input/echo/output buffers required in this TDD are implemented based on a queue-like structure. In addition to the traditional queue-in/queue-out operations, they also support *deleteLast*, which deletes the last element in the buffer, and *cutIn*, which replaces the last element in the buffer with a given element, both of which will be described later. The queue structure was chosen because access to data based on these buffers follows a first-in-first-out nature.

In order to avoid frequent data movement and increase the speed of data access, this TDD uses a circular queue structure. In this way, each data access can only rely on moving pointers (head and tail pointers), which optimizes the time complexity and space utilization.

In addition to the three buffers mentioned above, this TDD also adds a priority buffer in order to solve the problem of character processing. Whether for backspace/delete or newline, the TDD is expected to continuously output more than one character to the terminal, which requires a priority buffer above the input/echo/output buffer to continuously occupy the TIH's processing of subsequent string transmits in order to ensure the consistency of special character transmits. coherence.

Because the input buffer must keep track of the current line number to ensure Line-Oriented Input, the buffer structure also has variable *lines* to keep track of the number of *'\n'* in the input buffer.

1.3. Interrupt Handling and Character Processing

The TIH in this TDD is responsible for notifying the other procedures via state variables that the WDR has finished transmitting the last character and is now in an idle state. After that, it will get characters from these buffers and transmit them according to the priority. If these buffers are empty, TIH doesn't get any character and returns with the idle state of the WDR, otherwise it

will transmit the character it got with the busy state of the WDR. The reason for this design is that the call of TIH means the end of a transmission done by WDR, which reflects the state of WDR very well. The state and condition variables of WDR can prevent other procedures from seizing the resources of the output register when WDR is busy.

One of the features of RIH in this TDD is the handling of special characters. For backspace/delete, it transmits *'\b \b'* to the terminal. If the WDR is currently idle, the RIH will immediately call the WDR to transmit the first character, *'\b'*, and then add *' '* and another *'\b'* to the priority buffer. Otherwise it will enter all three into the priority buffer. For newline, the processing logic is similar, but differs in that newline transmits the newline character to echo buffer "as is" and hands over its character processing to TIH if the WDR is busy. backspace/delete does not leave any information in the echo buffer, so they can only resort to the priority buffer when the WDR is busy. Such a design is potentially problematic. The output priority of backspace/delete is higher than that of other characters stored in the echo buffer, which could be inputted before the backspace/delete. The reason for this design is that it ensures continuity of special character output regardless of whether the WDR is idle or not.

Another feature of WDR is its ability to determine whether a cold start is required. The logic is very simple, i.e., it determines whether the state of the WDR is idle or not. Does the idle state of WDR represent the idle state of the cycle? Is it possible that such a WDR idle state could be in the middle of processing the previous and the next character? This is not possible, because according to the implementation logic in TIH, if all buffers are empty, TIH will return with the idle state of WDR. Then according to the definition of Mesa monitor, TIH can't enter RIH before returning. Therefore, when RIH determines that WDR idle is when the whole cycle is idle, it needs to be cold-started, i.e., call WDR directly.

1.4. ReadTerminal API and Line-Oriented Input

ReadTerminal execution logic is "wait before reading", i.e., it waits for the user to type a newline (on a conditional variable that determines whether or not there is a newline currently), then reads the input buffer. The reason for this design instead of reading every character typed by the user is that the characters in the input buffer are variable and cannot be read by the user until the newline is received. The problem with this design is that if the user does not enter a newline until the input buffer is full, the driver will not perform a line feed because the newline is discarded instead of entering the input buffer. This TDD is designed so that if the input buffer is full and has not yet received a newline, the other new characters are discarded while the newline is added to the input buffer to replace the last element.

The *lines* variable within the input buffer structure is used to determine the occurrence of the newline. After receiving *'\n'* again, this variable is incremented by one. This variable is decremented by one when the RT finishes reading a line.

The ReadTerminal also uses a state variable and a condition variable to ensure that only one ReadTerminal reads the input buffer from the same terminal at the same time. The reason for this design is to prevent interleaved reads.

1.5. WriteTerminal and Condition Variables

The logic of *WriteTerminal* is to store the characters given by the user into the output buffer one by one and wait until the output buffer is full or there are no more characters written by the user before transmitting the data in the output buffer. The reason for this design instead of writing and transmitting at the same time is that the TIH is called after the WDR is called, and the TIH may find that the output buffer is empty after the WDR has finished transmitting (because it has not yet returned to the WriteTerminal for the next character to be written to the output buffer), and then the character-writing terminal cycle is interrupted and a cold start is required. Second, the TIH can handle special characters and monitor the status of the WDR as well as handling the output priority. However, TIH has no access to the user's buffer, only to the output buffer, so the logic here is that after the *WriteTerminal* has finished storing the characters in the output buffer, it calls WDR and TIH to transmit from the output buffer. This is also why the output buffer is physically present in this example.

In addition to similar logic to *ReadTerminal*, where *WriteTerminal* utilizes conditional and state variables to ensure that only one *WriteTerminal* is running per terminal without causing interleaved output, *WriteTerminal* requires another conditional variable to determine if the output buffer is full, and *WriteTerminal* also needs another condition variable to determine if the output buffer is full, to decide whether to start transmitting to the output buffer. In addition, it requires a condition variable to determine if all of the user's characters have been transmitted. The condition variable corresponding to the completion of the transmit of all characters is signaling in the TIH when the TIH doesn’t get any character from any buffer, which means that all buffers are empty and the TIH is called after the last character has been transmitted. This effectively prevents another character from entering the WDR until the WDR is completely idle.

2. Potential improvements

The system currently doesn't output a warning or *'\a'* when needed. For example, discarding newly incoming characters when the input buffer is full, or substituting newline for the

last character of a full input buffer are the cases when the TDD needs to output something in *stderr*. Also, the return values of some of my functions do not represent the execution state of the function. For example, *InitTerminal* or *InitTerminalDriver*, both of which may only return 1. And when an error is encountered nothing is returned. In short, my monitor needs to be strengthened in terms of error and exception handling. This enhancement would improve the strength of the code by allowing us to quickly find and fix problems when the system encounters an exception.

In *WriteTerminal*, it waits until the output buffer is full before transferring. This results in a waste of WDR resources, because there may be a period after the WDR finishes transmitting where it is idle because of the input of characters to the output buffer by writeTerminal. Therefore, I can improve on how *WriteTerminal* outputs characters to the output buffer to ensure that the WDR is always busy and maximize resource utilization.

Also, the ReadTerminal in this system waits for a line of data to be written to the input buffer before reading it immediately. So the variable lines in the input buffer to record the newline is at most 1 because after reading the input buffer, the value of lines will be decremented by one. So the possible improvement is to make this variable boolean to record whether there is a newline in the current input buffer or not, the variable will be True when accepting a newline, and False after the reading is finished.

3. Potential issues in large systems

My system is having problems with special characters. This is because backspace/delete has a higher priority than characters that go into the echo buffer before it. In small test cases nowadays, this problem is not obvious, because TDD hardly ever echoes characters through the echo buffer, but in a large systems, when there are a lot of users typing into the same terminal, the characters that didn't have time to be echoed will be backlogged in the echo buffer. When a user types backspace/delete, the echo buffer is skipped because it is entered into the priority buffer, and it is immediately echoed to the terminal with the highest priority. This is obviously not the correct order. Second, if a large number of users type a large number of special characters, it may cause the priority buffer to run out of space and then fail to echo those special characters.

A possible improvement would be to process special characters not through the priority buffer, but by processing special characters in TransmitInterrupt, i.e., when TransmitInterrupt reads a new character as backspace/delete, block the other procedures until it finishes

processing the *'\b \b'* echo. such a solution removes the limitation on system correctness imposed by the priority buffer and its size.