Many applications need different co-operating processes to get along with each other so that some overall function is achieved. When these co-operating processes work on the same data (e.g., a word processor where different subsets of the program all work on the user's text) the argument can be made that use of threads should offer improved efficiency. That is, instead of each of the co-operating processes being implemented as a heavyweight process (hwp) and shared memory or message passing being used to make data available to the hwp's, the processes could be implemented as individual threads in a single task and the sharing is thus implicitly provided since the threads all run in the same task.

Of course, multiple concurrently-active threads working on the same instance of data will require some mechanism to ensure correct synchronization of their activity.

In this assignment you will implement, as a multi-threaded application, a subset of a train ticket selling machine.

The real machine has a display (an LCD panel) a printer to print the customer's ticket, and a hex keypad (0..F). To buy a ticket, a passenger uses the hex keypad to select their destination in a dialogue with messages appearing on the display. Once the 'to' city is selected, the passenger is told how much the ticket costs and, after payment, is issued a printed paper ticket. The machine assumes the trip originates from its location, which it knows. The ticket shows: from-station, to-station, time and date of issue, and the fare.

The subset you are to implement for this assignment will do some of this. You don't have two separate output devices (LCD panel + printer) so will use the display you do have for both (one at a time). Note, too, that your program will be selective about what it accepts from the keyboard when (see below). The only keys your program should ever respond to are 'enter', the digits 0..9 (we'll cheap out and only do decimal), 'D to end input and 'C as the 'cancel' button a user can use to cancel a not-completed transaction. Note that 'C should not kill the program. Users should be told that they can press 'C to cancel their transaction at any time until the transaction is completed (which, in reality, would be after they've paid).

When started, your multi-threaded program enters a loop, which it exits only on EOF when in state 1 (see below). In the loop, your program:

<table>
<thead>
<tr>
<th>State</th>
<th>Behaviour</th>
<th>Keyboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>displays a welcome message, including simple instruction (e.g., &quot;Press enter to begin&quot;) and waits for the user to press enter to start a new transaction.</td>
<td>accept only enter, 'C or 'D; 'C returns to state 1, 'D ends input to the program</td>
</tr>
<tr>
<td>2.</td>
<td>once a new transaction starts, you display a list of available destinations (up to 10; not less than 6) and ask the user to press the digit key corresponding to the destination.</td>
<td>accept only enter, 'C, digits 0..9; 'C returns to state 1, a digit advances to state 3, anything else is ignored</td>
</tr>
<tr>
<td>3.</td>
<td>display the user's selected destination and the fare, and ask the user if this is what they want to buy: press 0 for no, 1 for yes. If the user indicates 'no', then take them back to state 2. If the user indicates 'yes', then go to state 4.</td>
<td>accept only 'C, 0, 1; 'C returns to state 1</td>
</tr>
<tr>
<td>4.</td>
<td>Print the ticket. This ends the transaction, so the program goes back to state 1</td>
<td>accept nothing: pressing keys should have no effect on the display at all</td>
</tr>
</tbody>
</table>

The format for the ticket is particular. A ticket is no more than 30 characters wide, so you must ensure you don't print anything wider than this. It should have at most two lines at the top identifying the railway and service, then a blank line, then the 'from station', line, then the 'to station' line, then the 'fare line', a blank line, and then the date and time printed as 'hh:mm dd MM YYYY' using 24-hour time. All two-digit fields have leading zeroes, and the month is the standard two-letter abbreviation. For example, 15:12 26 AU 2002.

Your program should use three threads:
(1) a keyboard-reading thread,
(2) an 'item-handling' thread,  
(3) a thread to generate the ticket

The first thread gets characters from the keyboard. As seen above, the characters it will pass on to the rest of the program depend on the state of the program; all other characters are to be ignored.

The second thread manages the transaction. This means it controls the dialogue with the user, is responsible for issuing the messages the user sees, and for figuring out the fare.

The third thread prints the tickets, based on what it is told by the second thread.

All three threads should be created when your process starts and run concurrently, using synchronization mechanisms to control when they should and should not do work. The entire program must loop and be ready to sell another ticket after completing a first transaction, and continue until an end-of-file occurs on stdin. Thus, your threads are created once when the process starts and stay active through however many transactions are conducted — they are not destroyed and created anew with each new sale.

This assignment must be written in on of C or C++. It requires you to do some work in controlling how a UNIX tty works, so as to get the desired keyboard action. For instance, in state 2, when the user has to select a city, your program should respond immediately when a digit is pressed, and not require the user to press a digit then the enter key. You also have to arrange for the actual character typed in not to appear as itself. That is, if a user selects the second destination on the list (number 2, say), then when they press the ‘2’ key, the character ‘2’ should not appear on the screen. Instead, your program should advance to state 3, and show the correct destination city, reflecting the ‘2’ the user pressed.

Now for some notes on how to go about this. This assignment is really straightforward except for figuring out how to actually use the threads. Make sure you spend enough time thinking about the design of the program, what the threads are going to do and when they can do it, before you ever start worrying about how to write code to do any of it. You will have much less difficulty if you start coding with a clear idea of what you want the program to do...don’t try designing it in C from the outset. For the threads, begin with a helpful command like man -k pthread. This should produce a long list of different functions that you can use to create, synchronize, manipulate and destroy threads. Pthreads should be available on any of the linux systems you have access to, as well as any of the Solaris systems.

Example of a display showing destinations:

```
SELECT YOUR DESTINATION

To select, press the key with the number appearing beside your destination (or press \e[C to cancel):

1. Atlanta   5. New York
2. Baltimore 6. Philadelphia
```

**Bonus option:** For up to 15% more points, make this ticket selling machine multilingual. This means introducing a change in the greeting screen to show the start message in each language. After the user presses enter, a new state is introduced offering the user a choice of language using the numeric keys to select. The rest of the states follow as above, with messages to the user continuing in the language they chose. The ticket is always printed in English, however.

One of the language choices must be English; the others can be any other language that works with the ISO-8859-1 character set — your choice, e.g., French, Spanish, German, Italian, Portugese, Swedish ... You can get 5% for each additional language you support, up to a maximum of 15% more.