CS39002 Operating Systems Laboratory
Spring 2026

Lab Assignment: 3
Date posted: 30-Jan-2026

Inter-Process Communication Using Pipes

In this assignment, you write a multi-process application to simulate the working of a DFA. Each state of the DFA is implemented by a process. The transition function dictates the communication pattern among the state processes. User inputs are handled by a separate process to be called the coordinator.

Storage of a DFA

A DFA $D = (Q, \Sigma, q_0, F, \delta)$ is stored in a text file in a format explained now. Let $s = |\Sigma|$ (the size of the input alphabet), and $|Q| = n$ (the number of states). The input symbols are denoted as a, b, c, ..., so keep s small ($s \leqslant 26$). Also, there will be n state processes, so you do not work with a very large DFA (take $n \leqslant 100$). The states (in Q) are numbered as 0, 1, 2, …, $n-1$. We always take the start state as $q_0 = 0$. The set F of final states is any subset of Q. The transition function is a map $\delta : Q \times \Sigma \rightarrow Q$.

The file storing the DFA D begins with s and n. This is followed by n lines storing the transition function and information about F. Each line is of the following form.

q final/nonfinal $\delta(q,a)$ $\delta(q,b)$ $\delta(q,c)$ ...

Here, q is a state (an integer in the range 0 to $n-1$), and final/nonfinal is F (if q is a final state) or N (if q is not a final state). Finally, each $\delta(q, .)$ is again a state (an integer in $[0, n-1]$).

As an example, the following is a file storing a DFA with $\Sigma = \{a, b, c\}$ (so $s = 3$), and with $n = 8$ states. The DFA accepts the input string if and only if the third last symbol in the string is an a.

```
3
8
0 N 1 0 0
1 N 3 2 2
2 N 5 4 4
3 N 7 6 6
4 F 1 0 0
5 F 3 2 2
6 F 5 4 4
7 F 7 6 6
```

The processes

You write a single file *rundfa.c*(*pp*) that implements the working of all the processes (the coordinator process and the n state processes). When you run your compiled code, the coordinator process (referred to as C) is launched. After some initial task, C creates n child processes. These are the state processes, and will be named as $S_0, S_1, S_2, \ldots, S_{n-1}$. C keeps on reading input strings one by one from the user. For each input string, C initiates the working of the DFA by activating S_0. Subsequently, the state processes are activated in sequence, as dictated by the transfer function. When the end-of-input is reached, the last active state process declares the ACCEPT/REJECT decision.

Initial work by the coordinator process *C*

You run the compiled code with one optional command-line argument: the name of the text file storing the DFA to be simulated. If no file name is supplied at run time, take the file *dfa.txt* as the input file. The coordinator process C first reads the input file, and stores the information for future use.

C then creates $n + 1$ pipes. The pipe P_C is meant for communicating to the coordinator C, whereas the pipe P_q is for communicating to the q-th state process S_q. All these file descriptors must be stored globally, because all the processes need to access the descriptors in future.

Subsequently, C forks the n state processes, and waits for some time (like one second) for these processes to become available.

C now uses the information read from the input DFA file. For each state q, C communicates to S_q using the pipe P_q the following information: whether or not q is a final state, and $\delta(q,\sigma)$ for each input symbol σ. The state process S_q needs to know only its line of the transition function. The alphabet size s may also be shared during this communication round, or stored in a global variable (which is copied during forking). As such, the state processes do not need the number n of states (assuming that the input DFA file has no errors), but this can also be passed in the initial communication round or stored in a global variable.

C enters a user loop (write it in a function), whereas each S_q enters a state loop (write it in another function). After forking and initial bookkeeping (like the first round of communication from C), each state process invokes the state-loop function. In this assignment, no child process `exec`'s any compiled code. In the user loop, C keeps on reading strings from the user. For each input string α, C initiates the working of the DFA on α, by sending the TRANSITION command to S_0 via the pipe P_0. In the state loop, each S_q keeps on waiting for a command (on the read end of its pipe P_q), and as soon as one comes, it takes an appropriate action. Note that S_q never returns to the function which forked it. Upon receiving the QUIT command (see below), S_q does not return from the state-loop function; instead it exits.

The session ends when the user presses Control-C.

Working of the DFA

For each symbol σ of the input string α, the processes behave as follows. Here, σ is either the next symbol in the input string or a special end-of-input marker (not in Σ).

> Let q be the current state, that is, the state process S_q has just received the TRANSITION command.
> At this point, the coordinator process C is waiting to read from its pipe P_C.
> C does not keep track of the transitions, and does not know (except initially) what the current state is.
> S_q sends the state number q to C via P_C.
> C sends σ to S_q via P_q.
> Depending upon σ, S_q (and C) work as follows.
>
> > **Case 1:** σ is an invalid symbol (neither in Σ nor the end-of-input marker). S_q writes to the terminal that the symbol is invalid, and does nothing. In this case, C also breaks the loop on sending symbols of α.
> >
> > **Case 2:** σ is the end-of-input marker. In this case, S_q prints ACCEPT/REJECT to the terminal (depending upon whether q is a final state or not), and does nothing else. The loop in C for sending symbols of α also breaks, and C goes to read the next input string from the user.
> >
> > **Case 3:** σ is a valid symbol in Σ. P_q looks at its own transition function $q' = \delta(q,\sigma)$ (it is allowed to have $q' = q$). S_q prints this transition step to the terminal, and then sends the TRANSITION command to $S_{q'}$ via $P_{q'}$. That is, for the next input symbol, q' will be the current state. The coordinator C does not get this information from any process (although C can track the current state from its knowledge of the complete transition table δ, it does not do so).
> >
> > After handling each of these three cases, S_q waits on P_q for receiving the next command.

Using high-level input and output

Although pipes can be accessed by the low-level `read()` and `write()` system calls, you do not do so here. Instead you use the system call `dup()` to duplicate `stdin` and `stdout` as and when needed. Then use the high-level I/O functions like `scanf()` and `printf()` or `cin` and `cout`. This will significantly boost the ease of writing the codes for message transmission.

Notice that these file handles vary with time. For example, the coordinator *C* can read input strings from the terminal. It can also read its pipe P_C to know about state numbers from current states. In all the cases, you use `scanf()` or `cin`. This means that the original file handles for the terminal should be copied beforehand, and reinstated as and when needed. Likewise, *C* can write the prompt to the terminal, or the next input symbol to the pipe of the current state process, or the TRANSITION command to the start process. All these outputs will use `printf()` or `cout`, so the `stdout` of *C* should be appropriately set to the correct targets.

Likewise, each S_q can read and write from/to several file handles.

Every read/write (using low-level or high-level calls) is buffered. Before `stdout` is changed to a new file descriptor, you must `fflush(stdout)`. Without that, the buffer may still contain data meant for the old file descriptor. It is safe to flush `stdout` after every print statement. There is no need to flush `stdin` (it is not illegal, but what will it do?).

Terminating the session

When the user presses control-c, the program should terminate. However, this has to be carried out in a controlled manner. All state processes should ignore SIGINT (use SIG_IGN), whereas the coordinator process *C* registers a wind-up function whenever SIGINT is caught. This signal handler of *C* does the following work.

C sends the QUIT command one by one to all the state processes, and waits for their termination. After all state processes terminate, *C* prints a customized message to the terminal, and itself exits.

When a state process S_q reads the QUIT command from its pipe P_q, it exits after printing a customized departing message to the terminal (in the verbose mode only).

Verbose/Non-verbose printing

Use a compile-time flag _VERBOSE to switch to the verbose printing mode. In the verbose mode, additional messages are printed to the terminal by the processes. In the default (non-verbose) mode, only the coordinator process *C* prints aggregate messages at the beginning and at the end of the session (see the sample output).

A random DFA generator

A file called *gendfa.c* is supplied to you. Compile and run this code with three optional command-like parameters specifying *s*, *n*, and the output file name (to store the DFA). The default values are $s = 4$, $n = 20$, and output file name = *dfa.txt*. The generator guarantees that all the states are reachable (from the start state).

Submit a single C/C++ file implementing both the coordinator- and the state-process functions.

Sample Output

```
$ gcc -Wall -o rundfa -D_VERBOSE rundfa.c
$ ./rundfa thirdlast.txt
                                                +++ Non-final state 0 created
                                                +++ Non-final state 1 created
                                                +++ Non-final state 2 created
                                                +++ Non-final state 3 created
                                                    +++ Final state 4 created
                                                    +++ Final state 5 created
                                                    +++ Final state 6 created
                                                    +++ Final state 7 created
                                    +++ Coordinator: 8 state processes are created
                                            +++ Coordinator: Going to user loop

Enter next string: abcabcabcabcabcabcabcabcabc
0 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2
  -- c --> 4 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2
  -- c --> 4 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2
  -- c --> 4 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2
  -- c --> 4 -- a --> 1 -- b --> 2 -- c --> 4 ACCEPT
Enter next string: abcabcabcabcabcabcabcabcabca
0 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2
  -- c --> 4 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2
  -- c --> 4 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2
  -- c --> 4 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2
  -- c --> 4 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 REJECT
Enter next string: abcabcabcabcabcabcabcabcabcab
0 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2
  -- c --> 4 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2
  -- c --> 4 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2
  -- c --> 4 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2
  -- c --> 4 -- a --> 1 -- b --> 2 -- c --> 4 -- a --> 1 -- b --> 2 REJECT
Enter next string:
0 REJECT
Enter next string: a
0 -- a --> 1 REJECT
Enter next string: aa
0 -- a --> 1 -- a --> 3 REJECT
Enter next string: aaa
0 -- a --> 1 -- a --> 3 -- a --> 7 ACCEPT
Enter next string: ^C

                            +++ Coordinator going to terminate all state processes
                                                    +++ State 0 going to quit
                                                    +++ State 1 going to quit
                                                    +++ State 2 going to quit
                                                    +++ State 3 going to quit
                                                    +++ State 4 going to quit
                                                    +++ State 5 going to quit
                                                    +++ State 6 going to quit
                                                    +++ State 7 going to quit
                                                         +++ Coordinator: Bye
$ gcc -Wall -o gendfa gendfa.c
$ ./gendfa 4 16 demodfa.txt
$ cat demodfa.txt
4
16
0 N 1 6 9 9
1 N 2 2 4 3
2 N 7 5 6 15
3 F 2 9 10 8
4 F 12 13 11 14
5 N 1 15 3 0
6 F 5 4 13 12
7 N 0 3 11 0
8 N 7 10 14 11
9 N 7 1 11 0
10 N 14 10 4 15
11 F 2 6 13 6
12 N 9 11 11 6
13 F 1 6 11 12
14 N 0 4 0 14
15 F 3 5 0 14
$ gcc -Wall -o rundfa rundfa.c
$ ./rundfa demodfa.txt
                                    +++ Coordinator: 16 state processes are created
Enter next string: adcb
0 -- a --> 1 -- d --> 3 -- c --> 10 -- b --> 10 REJECT
Enter next string: adcbbbbbbbbbbbbbbbbbbbbbb
0 -- a --> 1 -- d --> 3 -- c --> 10 -- b --> 10 -- b --> 10
  -- b --> 10 -- b --> 10 -- b --> 10 -- b --> 10 -- b --> 10 -- b --> 10
  -- b --> 10 -- b --> 10 -- b --> 10 -- b --> 10 -- b --> 10 -- b --> 10
  -- b --> 10 -- b --> 10 -- b --> 10 -- b --> 10 -- b --> 10 -- b --> 10
  -- b --> 10 -- b --> 10 -- b --> 10 REJECT
Enter next string: ddddddddddddddddddddddd
0 -- d --> 9 -- d --> 0 -- d --> 9 -- d --> 0 -- d --> 9
  -- d --> 0 -- d --> 9 -- d --> 0 -- d --> 9 -- d --> 0 -- d --> 9
  -- d --> 0 -- d --> 9 -- d --> 0 -- d --> 9 -- d --> 0 -- d --> 9
  -- d --> 0 -- d --> 9 -- d --> 0 -- d --> 9 -- d --> 0 -- d --> 9 REJECT
Enter next string: acdb
0 -- a --> 1 -- c --> 4 -- d --> 14 -- b --> 4 ACCEPT
Enter next string: acdbacdbacdb
0 -- a --> 1 -- c --> 4 -- d --> 14 -- b --> 4 -- a --> 12
  -- c --> 11 -- d --> 6 -- b --> 4 -- a --> 12 -- c --> 11 -- d --> 6
  -- b --> 4 ACCEPT
Enter next string: acdbacdbbdca
0 -- a --> 1 -- c --> 4 -- d --> 14 -- b --> 4 -- a --> 12
  -- c --> 11 -- d --> 6 -- b --> 4 -- b --> 13 -- d --> 12 -- c --> 11
  -- a --> 2 REJECT
Enter next string: abcdedcba
0 -- a --> 1 -- b --> 2 -- c --> 6 -- d --> 12 INVALID INPUT SYMBOL: e
Enter next string: ^C
                              +++ Coordinator: All state processes terminated. Bye.
$
```