Operating System
CS 370
Fall 2016

Project: 6
Dijkstra's K-State Mutual Exclusion

Due Date: Thursday,1 December 2016
Must be submitted before midnight

Total Points: 100

Teaching Assistant: Pradip Singh Maharjan
Office Hours:
MW: 2:00 pm to 4:00 pm
TuTh: 10:00 am to 12:00 pm

Course Web Page : `http://osserver.cs.unlv.edu/moodle/`

Requirements:

- Must be implemented in C (not C++)
- Must compile on cardiac.cs.unlv.edu
- First submittal of this project must compile as "gcc project6a.c". Similarly, second or third submittals must compile as "gcc project6b.c" or "gcc project6c.c" respectively.
- Project must be submitted through website ONLY.
- Input must be from redirection

Purpose:

This project has been designed to provide insight into socket programming. The project will also provide insight into mutual exclusion algorithms and self-stabilizing algorithms.

Objective:

The main objective of this project is to enable processes to communicate with each other using sockets. When processes are connected to each other, they create a ring network. The system keeps running the mutual exclusion algorithm and whenever a fault occurs, it recovers from the fault automatically. You are to implement Dijkstra's k-state mutual exclusion algorithm.

Dijkstra's K State Mutual Exclusion Algorithm:

The system consists of n processors, P_0, P_1, ..., P_{n-1} that are connected in a uni-directional ring. Each processor has a left and a right neighbor. The left neighbor of every processor P_i, 0< i <= n-1, is P_{i-1} and the left neighbor of P_0 is P_{n-1}. Similarly, the right neighbor of every processor P_i, 0 =< i < n-1, is P_{i+1} and the right neighbor of P_{n-1} is P_0. We call P_0 the bottom node and P_1 to P_{n-1} the other nodes. They have a distinct function.

Each processor P_i has a variable S_i that stores an integer value that is no smaller than 0 and no larger than K (K = n). The variable S indicates the node's state. The bottom node (P_0) receives the S_{n-1} value from the left neighbor (P_{n-1}) and uses the value obtained together with the value of S_{bottom} to compute a new value for S_{bottom}. Similarly, the other nodes receive the S_{i-1} value from the left neighbor, and use the value obtained together with the value of S_i to compute a new value for S_i.

Here is the algorithm:

```
Bottom :   if S = L, then S = (S+1) mod K
Others :   if S <> L, then S = L
* Range of value is 0 to K.
(L = State variable of left neighbor, S = state variable of itself)
```

When "if statement" of the node is true, that node is said to have a privilege. That is, the node can use the critical section. When the system starts, the S variable of each node can vary or an unexpected fault can change the value of S. When this occurs, then more than one node might have a privilege to use the critical section. By running this algorithm, however, the system will stabilize automatically in finite time and there will be only one process which has privilege at any given time. In a correct execution of the algorithm, the privilege will move in the ring from P_0, P_1, ..., P_{n-1}, P_0, P_1, ... It will continue until the user terminates it.

Input Specification:

You will have one compiled program, and that program will be run once for every node in the network. Each program will take in two integer values, the first is the index of the node in the ring, and the second is the size (number of nodes) of the network. This algorithm requires that each node is connected its left and right neighbor so the order that the nodes are started is important. So for this assignment the nodes will be started in order (0, 1, 2, ..., n-1). In connecting sockets there are two types of connections, server and client. A server connection is where the node will create the connection and wait for another node to connect to it. A client connection is where the node will connect to a node that has already created a server connection. Here is a diagram to illustrate how to get the network sockets connected.

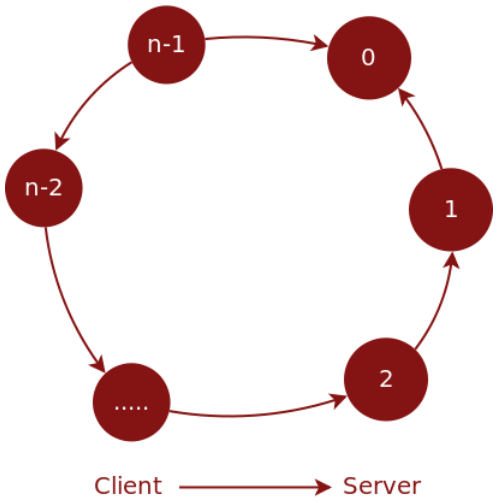

Figure 1: client-server connection

So there will be three different types of nodes for connecting the network.

- Bottom (node 0)
  - Create a server connection to connect to its right neighbor

  - Create a server connection to connect to its left neighbor
  - This order is very important

- Middle (node 1, n-2, . . .)
  - Client connection to its left neighbor
  - Create a server connection to connect to its right neighbor

- Top (node n-1)
  - Client connection to its left neighbor
  - Client connection to its right neighbor

Once the network has made all the connections it can start the algorithm. You are to use AF_UNIX sockets.

Output Specification:

Each node must notify that it is in the critical section with a print statement, here is an example.

```
####################################
     In Critical Section
####################################
```

When the node is not in the critical section it should clear the console. To do this you can simply print a bunch of empty print statements.

Handling Faults:

Process's value (S from the algorithm above) should be set to a random number (between 0 and n) intially. To get the random number use int value = rand() % n; which is in the stdlib.h library.
When a fault occurs you will see that multiple nodes are in the critical section at one time. If you implemented you program correctly, it should stabilize to having only one process in the critical session at a time.

Point Breakdown:

- Connecting the sockets in a ring network [30 pts]
- Correctly implementing the algorithm [30 pts]
- Correctly simulating faults [25 pts]
- Coding style
  - Proper and informative comments [5 pts]
  - Informative variable names [5 pts]
  - Code organization (methods are not too long) [5 pts]

References:

- http://theory.lcs.mit.edu/classes/6.852/05/papers/p643-Dijkstra.pdf