CSC 2/458: Parallel and Distributed Systems
Spring 2019.
Assignment 2: Drinking Philosophers

Name: Soubhik Ghosh
Email: sghosh13@ur.rochester.edu

Description:

1 source file: philosophers.cpp

1 make file: Makefile

Compilation: make philosophers

Executable: philosophers

The code is commented with enough information, with the README giving a complete overview.

philosophers.cpp

- First the command line arguments are parsed if any.
- Accepts the following 3 kinds of optional arguments in any order:
  -s #session, - (hyphen), #filename
  - -s #session – (Integer) Number of drinking sessions by each philosopher.
  - - (hyphen) – Read configuration from stdin (until Ctrl-d is pressed).
  - #filename – Read configuration from filename
  - Eg. ./philosophers -s 100 config.txt
- If the arguments are malformed or the config file can't be read, then appropriate error messages are displayed, and the program exits.
- The Graph class is the root container class for holding and managing the graph of conflicting processes (philosophers).
- The Fork class is a definition of an edge of the graph.
- The Graph methods can access all the members of the Fork class (friend class).
- Graph contains an adjacency list structure for representing the conflict graph where the nodes are philosophers and the edges are forks, which implies that any 2 philosophers can share at most 1 fork. This graph once initialized will never change shape (static) except the directions of the edges.
- Forks are declared as an edge list in the Graph definition. Hence every philosopher has a list of numbers (shared forks) each "pointing" to an edge (fork) in the edge list.
- Each philosopher is declared as a struct which consists of a thread, the depth (precedence), eating state, accessible list of forks and a condition variable used for waiting to get precedence.
- The fork has state information which are the current owner of the fork, the current holder of the request token, the fork status (dirty/clean) and the orientation of the edge for that fork.
  All this state information is protected by a per fork mutex lock.

- The graph is initialized as an undirected graph with the input consisting of the number of nodes (philosophers) and a list of edges. During this time the graph is checked for correctness such as bounded node id numbers, no self-loops, no duplicate edges and connectivity (no isolated nodes). If any of theses conditions are not met, then the program exits with an appropriate error message.
- This graph is then turned into a DAG (Directed Acyclic Graph) by performing a BFS from philosopher #0 and directing edges away to every new node. The depths are also initialized at this stage, where we can easily see that the #0 philosopher has the highest precedence (and hence can start eating first) according to the definition of depth.
- The drinking sessions start by the main thread creating thread objects for every philosopher while ensuring all the philosophers start (hungry) at the same time by making them spin on a shared flag. The main thread sets this flag after creating all the thread objects.
- Every philosopher starts hungry, but only those with depth = 0 can proceed to eat. Others just wait on their condition variable for the depth to become zero. As per the article these philosophers will eventually eat in finite time as the precedence is propagated through the graph giving fairness to everybody even when the other philosophers eat near infinite times (large number of times).
- After having precedence, the philosopher loops over all its incident edges to get clean forks (with and without requesting). If it doesn't have a clean fork, it sends the request token to its neighbor. This neighbor then immediately transfers the fork. We can assure that the neighbor will always have a dirty fork in this case as the edge is pointing towards the neighbor as she has eaten recently. This also ensures that the recently eaten neighbor can eat again only after the current philosopher eats and releases the fork.
- The philosopher then dirties all the incident forks and starts eating.
- After eating, she immediately starts thinking, while yielding up precedence to its neighbors.
  The yielding action consists of pointing all the edges towards this philosopher atomically and updating the depths in the graph. Lastly all the neighbors who might have just got the highest precedence are notified to stop waiting on their condition variable.
- Finally, when all the philosophers have finished eating, the main thread joins them up and exits the program. (If there is a deadlock, the program will never exit).