Justin Bahr
NUID: 001592707

High Performance Computing
Homework 2 - Question 2

a. If there are five philosophers at the table but only three forks, only one philosopher will be able to eat at a time. Additionally, if philosophers are allowed to grab one fork at a time while still waiting on the other fork, three different philosophers could grab each fork, which could lead to deadlock and starvation (unless the solution used requires both the left and right fork to be available before picking either up).

b. If one philosopher is given priority over the other's it will always be the first to grab available forks, meaning he/she will eat more often than other philosophers. This leads to an unequal distribution of resources. When applied to a multicore processor, this would mean that other cores would be more likely to be idle (thinking), wasting resource power and reducing concurrency. One potential strategy to reduce the severity of this unequal distribution would be introducing a required period of thinking after eating, so other philosophers get a chance to eat after the high priority philosopher puts down his/her forks.

c. Varying the order in which philosophers pick up forks prevents the chance of a deadlock where all philosophers have picked up their respective left fork only or right fork only due to shared competition for the first fork. However, this pattern requires a more complicated branching to assign the first and second fork to pick up.

My Solution with Pthreads:
I implemented two solutions using Pthreads. The first prevents deadlock by requiring both the left and right fork to be available before a philosopher picks up either. The second prevents deadlock by alternating the order in which philosophers pick up forks (left vs right) around the table. This works by breaking the symmetry and creating competition for the first fork as described in part c. Below is a screenshot with five philosophers demonstrating that they are all able to eat, and no deadlock occurs:

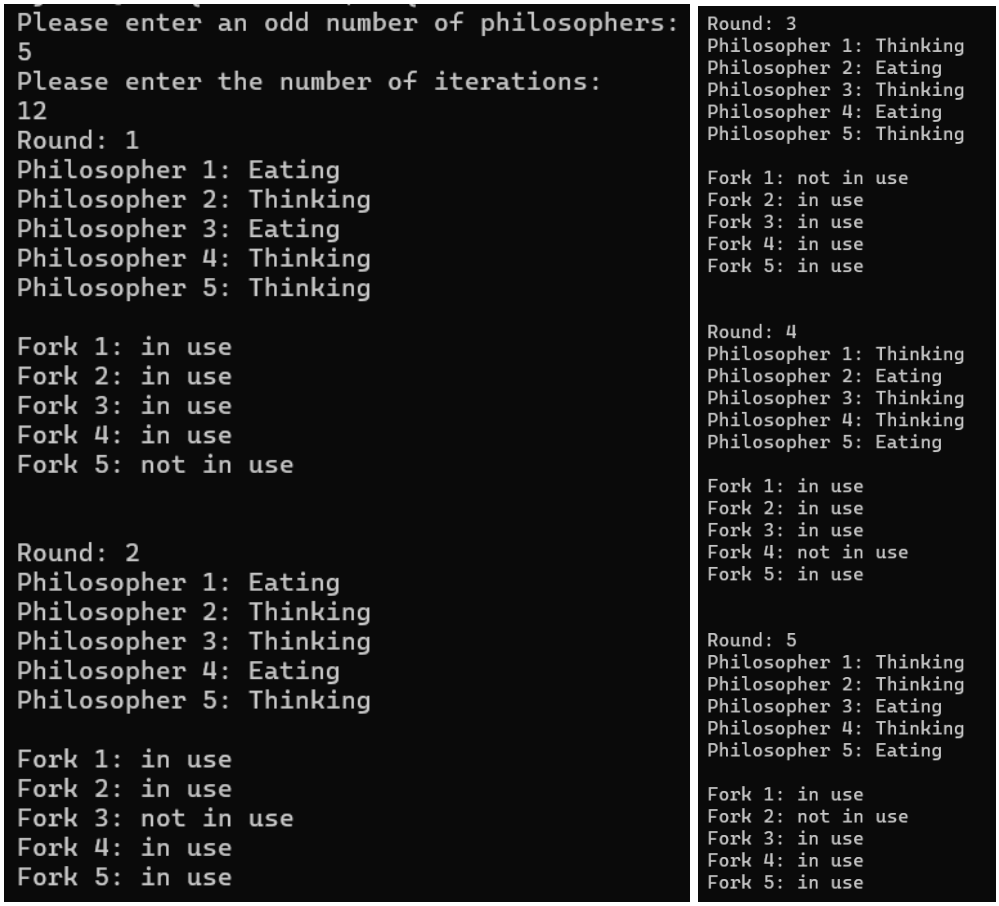

Figure 1. Pthread Demonstration

Edsger Dijkstra:
Edsger Dijkstra was a computer scientist from the Netherlands. He is well known for his algorithm to find the shortest path(typically in a weighted graph. He designed this algorithm in 1956. He also contributed to Reverse Polish Notation, a mathematical representation where oparands come before operators (ie 3+4 -> 3 4 +), as well as the related Shunting Yard Algorithm which uses a stack to convert expressions into RPN. Additionally, he developed the multiprogramming system and semaphores. In 1974, he won the Harry H. Goode Memorial Award for his contributions to computer programming.

The dining philosophers problem has multiple solutions, each with the goal of preventing deadlocks and data races, while promoting fairness among the philosophers. This problem was posed to illustrate the challenges of parallel programming, synchronization, resource allocation, which is why our solution is coded using Pthreads and mutex locks or semaphores. The philosophers represent processes (threads), and the forks represent hardware resources such as cores, memory, and I/O. The solutions to this problem are important for multithreaded applications, operating systems, database systems, networks, and cloud services.

Dijkstra's shortest path algorithm is a greedy algorithm that keeps track of the "cost" (shortest distance from the start) of each node. It begins by setting the cost of each node (except for the starting point) to infinity (practically, a very large number). Next, it marks all other vertices as unvisited. Then, all the neighbors of the first vertex are relaxed (cost is updated if it is lower than the previously marked value. After this, the algorithm proceeds to the unvisited neighbor with the shortest distance, which is marked as visited, and all its neighbors get relaxed. This continues for until all vertices have been visited. By the end, the shortest distance from the start to each vertex (cost) is known, so you can trace back the shortest path by moving backwards from the end vertex to the neighbor with the lowest cost repeatedly until the start is reached. This algorithm has time complexity O(V^2) or O([V+E]logV) if a priority queue is used to keep track of the costs of each vertex and select the next vertex to visit. Here, V is the number of vertices and E is the number of edges.

Sources:

https://www.computer.org/profiles/edsger-dijkstra

https://adacomputerscience.org/concepts/path_dijkstra

https://web.eecs.utk.edu/~jplank/plank/classes/cs560/560/notes/CBThread_Dphil/