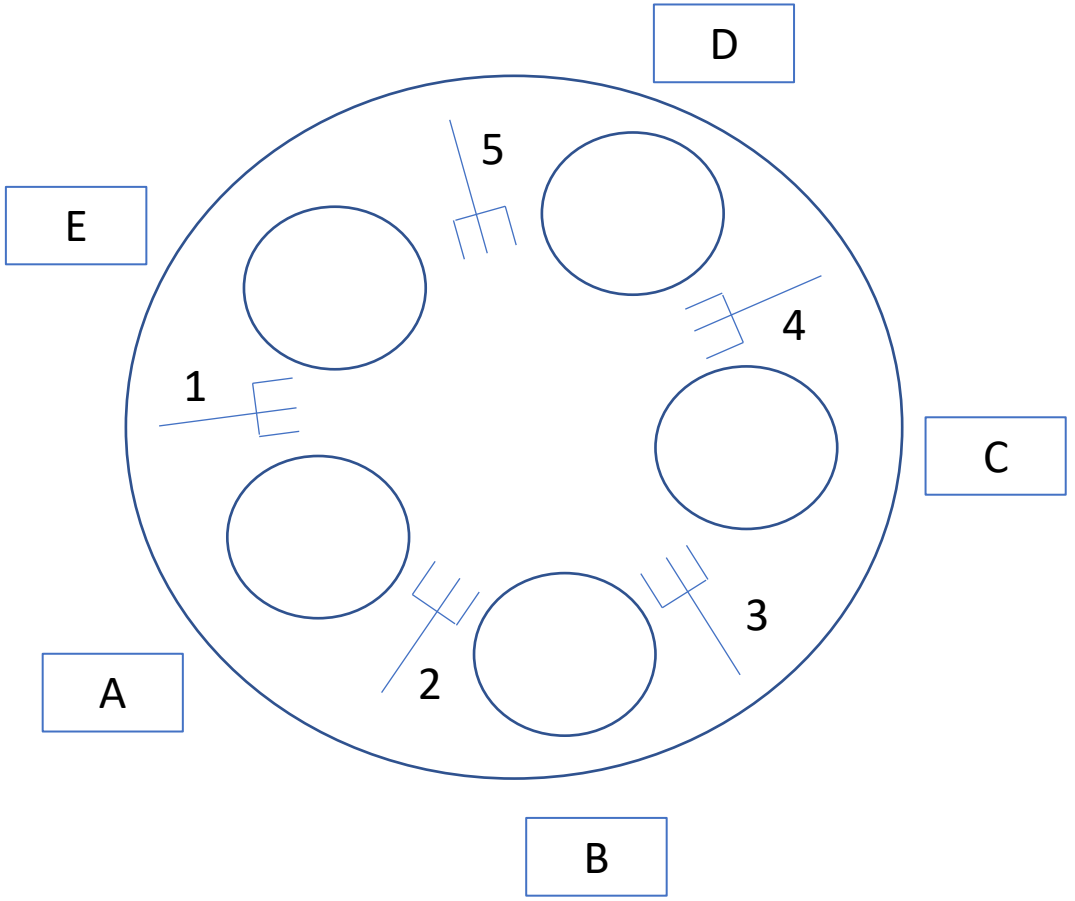
Locking Workshop Solutions

Loop

- Possible scenario
  - Thread 1 tests x == 0. This is true because x has not been modified
  - Thread 2 tests x == 0. This is true because x has not been modified
  - Thread 1 sets x to 1 – x, giving 1
  - Thread 2 sets x to 1 – x, giving 0
  - Thread 1 tests x == 0. This is true
  - Thread 2 tests x == 0. This is true
  - Thread 1 sets x to 1 – x, giving 1
  - Thread 2 sets x to 1 – x, giving 0
  - …
- This is an example of "livelock". The program continues to execute, but makes no progress

Dining Philosophers



Dining Philosophers

- This is a classic problem which demonstrates the problem of deadlock
- Possible implementation
  - Philosopher thinks
  - Philosopher picks up left fork
  - Philosopher thinks
  - Philosopher picks up right fork
  - Philosopher eats
  - Philosopher puts down right fork
  - Philosopher thinks
  - Philosopher puts down left fork
  - Philosopher thinks

Deadlock

- If all the philosophers pick up their left fork, there will be no right forks available (B picks up fork 2, which is A's right fork, preventing A from eating, and so on)
- A's right fork will not become available until B has finished eating, and B cannot start eating because C's left fork is taken
- The philosophers are stuck in the "thinking" state

Livelock

- Trying to avoid deadlock can result in livelock
  - Philosopher picks up left fork
  - Philosopher waits 30 seconds to pick up right fork
  - If unable to pick up right fork, philosopher puts down left fork
  - Philosopher waits 30 seconds
  - Philosopher picks up left fork
- If the philosophers happened to start eating at the same time, they will all pick up their left forks at the same time and prevent each other from picking up their right fork
- The philosophers are changing state, but not able to enter the "eating" state

Solutions

- One solution is to provide a central arbitrator
- A philosopher must ask permission from the waiter before picking up a fork
- The waiter will only allow one philosopher to pick up a fork at a time
  - Philosophers may put down forks at any time
- This is easy to implement, using a mutex, but reduces parallelism
  - If D wants to eat when B is already eating, D cannot start until B has finished, even though both forks 4 and 5 are free
- Another solution is to use a shared lock
  - In effect, a philosopher is allowed to pick up both forks at the same time

Solution without synchronization

- In this case, deadlock can be avoided by introducing a resource hierarchy
  - The forks are numbered from 1 to 5
  - We add a rule that a philosopher must pick up the lowest numbered fork before picking up the highest numbered fork
- If all the philosophers try to eat at the same time, A will pick up fork 1, B will pick up 2, C will pick up 3, D will pick up 4 but E will try to pick up 1
- This leaves fork 5 free for D, who can start eating