1 Introduction

In this lecture we discuss the Dining Philosopher’s Problem. We show that the Dining Philosopher’s algorithm cannot be symmetric or anonymous (defined below). We then introduce the Right-Left algorithm.

Definition 1 (Symmetry) An algorithm is symmetric if it is the same for all processes.

Definition 2 (Anonymity) An algorithm is anonymous if a processor does not know its index. Anonymity implies symmetry.

- Technical note: The queue’s for the chopsticks below can be implemented using the Read-Modify-Write (RMW) objects discussed in Lecture 7.

2 Dining Philosophers Problem

The dining philosophers (DP) problem is described as a set of processors in a ring, where each shares a resource (encoded as a RMW FIFO queue) with each of its adjacent neighbors. In order to enter the critical section, a philosopher needs both resources. Thus, any algorithm for the philosophers can be summarized as follows:

1. REM - thinking.
2. ENTRY - wants to eat. Tries to pick up chopsticks.
3. CS - has both chopsticks; eating.
4. EXIT - release chopsticks.

3 Symmetry and the DP Problem

3.1 No Symmetric (or Anonymous) Solution

Theorem 1 There is no symmetric solution to the DP problem.
Figure 1: The Dining Philosophers: Each philosopher $P_i$ is sharing a resource $f_i$ shown by an edge between the two. The philosophers and the resources form a ring topology.

**Proof:** Suppose for contradiction that there is a symmetric algorithm $A$ for $n$ processors. Consider an execution $\alpha$ for $A$ that begins with all processors in the same state and with all shared variables in the same initial value. Execution $\alpha$ proceeds in round robin fashion for some number of steps $n$. (e.g. $1, 2, ..., n, 1, 2, ... n, ...$ etc.)

Since each processor has the same state at the beginning of a round (and philosophers cannot send messages to each other), they will read/write the same value in their shared registers, and end in the same state at the end of the round. By induction, we can show that after any number of rounds, all processors will be in the same state. So, if any processors enter CS after some round $k$, to satisfy progress then all processors will enter CS after round $k$, violating exclusion. Thus there is no symmetric solution.

3.2 A wrong DP Algorithm

Examine the following symmetric algorithm. Each procedure after entering ENTRY waits first for the right chopstick and then the left chopstick after. Getting both chopsticks, it enters CS. On exit, it releases both chopsticks and enters REM. All processors then take the following steps:

1. All processors enter ENTRY.
2. All processors grab right chopsticks.
3. All processors wait for left chopstick.

Thus progress fails.
4 Right-Left Dining Philosopher’s Algorithm

4.1 Introducing the Algorithm

The Right-Left (RL) Dining Philosopher’s Algorithm guarantees progress as well as no starvation. It follows from thm 1 that the RL algorithm is not symmetric. Some processors seek the left chopstick first, while some seek the right chopstick first. It may, however, create long waiting chains. Waiting time is proportional to the length of the waiting chain, which is proportional to the number of philosophers. Further on, we will need to guarantee a small bound on the length of the waiting chain.

Idea for the RL Algorithm

Each chopstick has a shared variable, which is accessible via a FIFO queue of the two adjacent processors. For an even number of processors the algorithm works as follows:

1. Odd-numbered processors $P_i$ (starting with $P_1$) seek the right chopstick $f_{i-1}$ first.
2. Even-numbered processors $P_i$ seek the left chopstick $f_i$ first.
3. Processors seek a chopstick by placing their index at the end of a queue.
4. Processors obtain chopstick when their index reaches the front of the queue.
5. Processors enter CS after obtaining chopsticks.
6. On exit processors remove their indices from chopstick queues.

For an odd number of processors the algorithm works similarly:

1. Processors $P_1, P_2, ..., P_n$ work as defined by the RL algorithm for an even number of processors.
2. The last processor $P_{n+1}$ where $n = 2k, k \in \mathbb{N}$, will seek the right chopstick $f_n$ first and the left chopstick $f_{n+1}$ second.
3. The processors then proceed through the algorithm as if there were an even number of processors.

Note that the odd implementation of the algorithm will not affect the following proof for mutual exclusion. The proof for mutual exclusion relies on the definition of the Dining Philosopher’s problem and is independent of the number of philosophers. The time complexity for processors $P_n$ and $P_{n+1}$ in the odd case will be more, and the time complexity for processor $P_1$ will be less as shown in lemma 5, but all other processors will not be affected.
Lemma 2 Each processor $P_i$ is in the head of queues for chopsticks $f_i$ and $f_{i+1}$ while processor $P_i$ is in $CS$. 

Proof: $P_i$ must be at the head of the queues for both shared resources $f_i$ and $f_{i+1}$ according to the definition of the Dining Philosopher’s Problem. ■

Theorem 3 Mutual Exclusion is satisfied by the RL algorithm.

Proof: Suppose for contradiction that there exists two adjacent processors $P_i, P_{i+1}$ such that both are in $CS$ at the same time. Let the variable $f_{i+1}$ be shared between $P_i, P_{i+1}$, informally called a “chopstick” and encoded as RMW FIFO queue. According to lemma 2, if processor $P_i$ is in $CS$, it is in front of the queue for chopstick $f_{i+1}$, and it will leave $CS$ before $P_{i+1}$ can enter $CS$ (due to the properties of queues given in lecture 7, theorem 8). Thus, suppose processor $P_{i+1}$ is in $CS$. Again, if processor $P_{i+1}$ is in $CS$, it is in front of the queue for chopstick $f_{i+1}$, and it will leave $CS$ before $P_i$ can enter $CS$. ■

4.2 Time Complexity of Right-Left Dining Philosopher’s Algorithm

Let $l$ be the upper bound on a time step of each processor, and let $c$ be the upper bound on time a process is in $CS$.

Lemma 4 In the Right-Left Dining Philosopher’s Algorithm, the time between when a processor enters $ES$ and enters $CS$ is at most $3c + 18l$ for all processors when there are an even number of processors.

Proof: When a processor (i.e. philosopher) is at the head of a queue for a resource (i.e. chopstick) $f_i$, we say the processor has the chopstick $f_i$. If a processor has the chopstick when it adds itself to the queue, we say the processor knows it has the chopstick. The whole operation of adding itself to the queue and knowing it has the chopstick takes $l$ time due to the properties of the RMW registers composing the queues. If a processor does not have the chopstick after it adds itself to the queue (taking $l$ time) it must perform another check to know if it has a chopstick after it attains it, which takes an additional $l$ time.

Let some processor $i$ enter ENTRY. Let $t$ be the time $i$ takes to enter $CS$. A chopstick between 2 processors is either chosen first for both or second for both, with the exception of when we have an odd number of processors. Lets examine the even case. This implies that the max waiting chain is 2. Let $s$ be the max time between obtaining the first chopstick and entering $CS$. After time $l$, processor $i$ tries to get the first chopstick $f_m$. If available, then the total time needed is equal to $l + s$. Otherwise, neighbor $j$ has its first chopstick $f_m$ when $i$ tries to get the first chopstick.

If neighbor $j$ has $f_m$, recall that it takes processor $i$ $l$ time to add itself to $f_m$’s queue, even though it does not attain the chopstick. Furthermore, neighbor $j$ will take $s$ time to enter the
CS, $c$ time in CS, and $l$ time before putting the chopstick down at which point $i$ will finally have the chopstick. After neighbor $j$ releases $f_m$, it will take processor $i$'s time to enter CS. Thus, in the worst case we need $l + s + c + l + s$ time for processor $i$ to enter CS. It then follows that $t \leq \max(l + s, c + 2l + 2s)$, and $t \leq c + 2l + 2s$.

Next, examine time $s$. The best case is $s = 3l$. The best case can be explicated as follows: when both chopsticks are available, processor $i$ needs $l$ time to know it has its first chopstick (which is true since $i$ has the first chopstick by the definition of $s$), $l$ time to attain and know it has its second chopstick, and $l$ time to enter CS. Otherwise $s = c + 8l$:

- $l$ to know we have the first chopstick.
- $l$ time to put $i$ in the second chopstick $f_n$’s queue (and realize we do not have the second chopstick).
- $(2l + c + 2l)$ for the time another adjacent processor $k$ takes to get into and out of CS. We need $l$ to know we have $k$’s second chopstick and $l$ to enter CS. It takes $k$'s $c$ time to get out of CS. Finally we need $l$ for dropping its first chopstick, and $l$ to drop its second chopstick $f_n$ in EXIT.
- $l$ for processor $i$ to know it has the second chopstick $f_n$.
- $l$ to enter CS.

Thus $s \leq c + 8l$. Substituting this into the equation for $T$ results in:

$$t \leq c + 2l + 2s$$
$$t \leq c + 2l + 2c + 16l$$
$$t \leq 3c + 18l$$

Thus the maximum time between when a processor enters ES and enters CS is $3c + 18l$. Intuitively, one can see that up to three processors are entering CS before processor $i$. The processors are 1. the processor $j$ sharing $i$’s high priority chopstick, 2. the processor sharing $j$’s low priority chopstick, and 3. the processor sharing $i$’s low priority chopstick.

**Lemma 5** For the odd case of the Right-Left Dining Philosopher’s Algorithm, the time between when a processor enters ES and enters CS is at most $4c + 21l$ for processors $P_n$ and $P_{n+1}$, $2c + 15l$ for processor $P_1$, and $3c + 18l$ for $P_2, P_3...P_{n-1}$.

**Proof:** Examine processor $P_{n+1}$. If neighbor $P_1$ or $P_n$ has the chopstick, then they will take $s$ time to enter the CS, $c$ time in CS, and $l$ time before putting the chopstick down. Let $t_{n+1}$ be the time it takes $P_{n+1}$ to get into CS. It then follows that $t_{n+1} \leq \max(l + s, 2c + 5l + 2s)$, and $t_{n+1} \leq 2c + 5l + 2s$. We need $c + l + s$ time for processors $P_1$ and $P_n$, and another $3l$ for processor $P_{n+1}$. Time $s$ remains the same as in lemma 4. Thus:
Examine processor $P_n$. Let $t_n$ be the time it takes $P_n$ to get into CS. $t_n \leq \max(l+s, 2c+5l+2s)$, where $c+2l+2s$ comes from $P_1$ and $P_{n+1}$ similar to the even case, and $3l+c$ for $P_n$. This will also lead to:

$$t_n \leq 4c + 21l$$

Examine processor $P_1$. Let $t_1$ be the time for $P_1$ to enter CS. $t_1 \leq \max(l+s, c+7l+s)$, where $l+s$ is the time it takes for $P_{n+1}$ to enter and drop its chopstick and $3l+c$ is the time it takes for $P_2$ to enter CS and drop its chopstick, and $3l$ is the time it takes for $P_1$ to check both chopsticks and enter CS. Substituting for $s$ we get:

$$t_1 \leq 2c + 15l$$

For all other processors $P_i$, the adjacent processors that can enter CS during $P_i$’s time in ES are congruent to the processor $i$ used to prove lemma 4. Their max times remain the same.