CS 340
Computer Systems

#11: Deadlocks and The Sleeping Barber

Sep. 26, 2023 · G Carl Evans

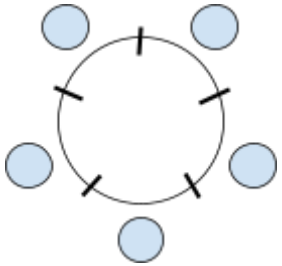
Critical Sections

We know that critical sections require exclusive access to a resource. We also know locking a resource is computationally expensive. However, are there other concerns?

The Dining Philosophers

Imagine five philosophers and five chopsticks at a circular table. Each philosopher has two states: **eating** and **thinking**:

- When a philosopher is thinking, she holds no chopsticks.
- When a philosopher starts the process of eating, she must take the chopstick to her left, then her right, and then begin eating.



Q: Using the strategy described above (take left, take right, then eat), what happens over a long period of time?

`See Lecture Code: 09/dinning-philosophers.c`

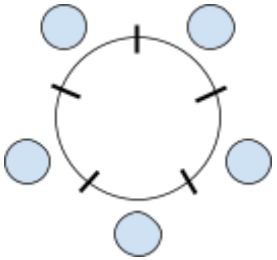
Deadlock:

- Definition:

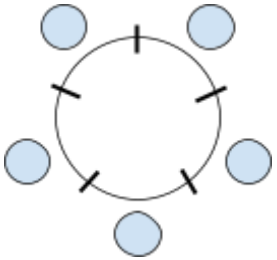
Coffman Conditions:

- Four **necessary** conditions of deadlock:

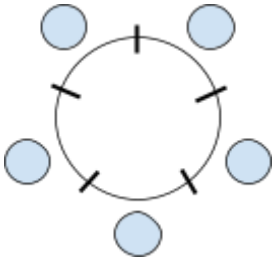
1)



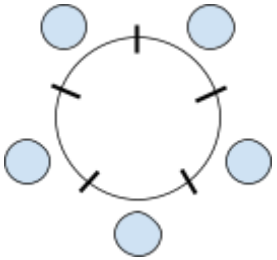
2)



3)



4)



Sleeping Barber
A classic problem in synchronization is the "Sleeping Barber Problem" *(a barber is someone who cuts hair)*.

There are two requirements for the barber:

1. If there are no customers, the barber sleeps in the chair.

2. When there is a customer, the barber wakes up and the customer sits in the chair for a haircut. If there's already a customer in the chair, the customer waits until the chair is available.

Code Overview

Barber: There is only one barber. The barber runs as a single thread in the **`barber`** function. The **`barber`** function contains a **`while (1)`** loop. The barber thread must only exit the **`while (1)`** loop when a customer with a **`name == NULL`** arrives.

Customer: Each customer runs as a separate thread in the **`customer`** function. Each customer has a unique name provided to the **`customer`** function. There will be many customers arriving concurrently.

Requirements

Complete the sleeping barber problem to ensure that:

- When no customers are present, the barber is waiting for a customer in a **blocked** state (**NOT** "busy waiting").

- When a customer arrives and no other customers are present, the barber is unblocked and cuts the customer's hair. The barber cuts hair by calling **`cut_hair(const char *name)`**, as provided in the example code. The name must be the **`name`** of the customer whose hair is to be cut.

- When a customer arrives and another customer is getting their haircut, the customer must wait in a **blocked** state (**NOT** busy waiting).

`10/sleeping-barber/sleeping-barber.c`

```c
struct _sleeping_barber_t {



};
typedef struct _sleeping_barber_t sleeping_barber_t;

void init(sleeping_barber_t *sb) {


}

void barber(sleeping_barber_t *sb) {
  while (1) {









    cut_hair("waf");









  }
}

void customer(sleeping_barber_t *sb, const char *name) {
  // Each customer is a thread! :)









}
```