# Section 13: Intro to Distributed Systems

November 25, 2015

## Contents

# 1 Warmup

## 1.1 General's Paradox

Problem setup:

1. There are two generals, standing on opposite mountains.
2. They are communicating via messenger to plan an attack for the next morning.
3. The messengers who run between the two generals can be captured.

Let's say we're a spy in the valley, and we have seen the following exchange taking place:

```
General 1: Shall we attack at 11 AM tomorrow?
General 2: 11 AM is OK for me.
General 1: I shall have my troops ready at 11 AM then. Will you?
General 2: Yes, I shall.
```

Can we conclude that the two generals will attack at 11 AM on the morning after this exchange takes place?

What does this example demonstrate about systems that require communication?

# 2 Problems

## 2.1 General's Paradox Continued

How can we modify the protocol that the Generals use to ensure that both Generals attack at 11 AM the day after the exchange takes place? (Hint: can we change *how* they communicate?)

## 2.2 Message Passing With Threads

In lecture, we talked about message passing APIs, where threads/processes/computers communicate by sending and receiving messages. Let's come up with an example message passing implementation using the `pthread` or Pintos threading APIs.

### 2.2.1 Message Inbox Datastructure

```
struct inbox_t {
```

```
};
```

#### 2.2.2 Inbox Initialization

```
void inbox_init(inbox_t * inbox) {



}
```

#### 2.2.3 Message Sending Function

```
int msg_send(inbox_t * inbox, void * msg, size_t msg_size) {









}
```

#### 2.2.4 Message Sending Function

```
int msg_recv(inbox_t * inbox, void * msg) {









}
```

### 2.3 Baking with Messages

In a previous section, we went through a problem where we were trying to coordinate the baking of a cake between multiple people who only know a single skill (threads) using condition variables. Now, let's say that we are trying to solve the same problem, but our multiple people are spread across a network,

and we are coordinating to cook the cake via messages. Let's use the message passing API we came up with in the last problem to build this.
A cake requires:

- 1 cake batter
- 2 eggs

Instructions:

1. Add ingredients to bowl
2. Heat bowl (it's oven-safe)
3. Eat the cake, clean out the bowl, and go back to step 1

Requirements:

- Don't start heating the cake in the oven unless there are exactly the right number of ingredients in the bowl.
- Don't add raw ingredients to a currently-baking cake or a finished cake.
- Don't eat the cake unless it's done baking.
- Given enough time, the code should bake an unbounded number of cakes, and should never stop.

In this problem, if you are allocating memory, you can assume that `malloc` always succeeds.

```
inbox_t ________________________________________;

// add message struct definitions here, if necessary









void batterAdder()
{



}

void eggBreaker()
{



}
```

```
void bowlHeater()
{



}

void cakeEater()
{




}

int main(int argc,char *argv[])
{
  // Initialize mailboxes
  // Starts threads that communicate via mailboxes:
  //   - 1 batterAdder
  //   - 2 eggBreakers
  //   - 1 bowlHeater
  //   - 1 cakeEater
  // main() sleeps forever
}
```