Common Concurrency Problems

Producer Consumer Problem

- Processes share bounded buffer of size K
- Producers produce items to insert into the buffer only when buffer empty and consumers remove items from buffer only when buffer not empty

```
Count = In = Out = 0
Mutex = S(1)
CanProduce = True, CanConsume = False
```

```
// Producer
While (True) {
  Produce Item;
  While (!CanProduce);
  Wait(Mutex);
  If (Count < K) {
    Buffer[In] = Item;
    In = (In + 1) % K;
    Count++;
    CanConsume = True;
  } else {
    CanProduce = False;
  }
  Signal(Mutex);
}
```

```
// Consumer
While (True) {
  While (!CanConsume);
  Wait(Mutex);
  If (Count > 0) {
    Item = Buffer[Out];
    Out = (Out + 1) % K;
    Count--;
    CanProduce = True;
  } else {
    CanConsume = False;
  }
  Signal(Mutex);
  Consume Item;
}
```

- Uses busy waiting

```
Count = In = Out = 0
Mutex = S(1), NotFull = S(K), NotEmpty = S(0)
```

```
// Producer
While (True) {
  Produce Item;
  Wait (NotFull);
  Wait(Mutex);
  Buffer[In] = Item;
  In = (In + 1) % K;
  Count++;
  Signal(Mutex);
  Signal(NotEmpty);
}
```

```
// Consumer
While (True) {
  Wait (NotEmpty);
  Wait(Mutex);
  Item = Buffer[Out];
  Out = (Out + 1) % K;
  Count--;
  Signal(Mutex);
  Signal(NotFull);
  Consume Item;
}
```

Readers Writers Problem

- Processes share data structure D where reader retrieve information from D while writers modify information in D
- Writer needs exclusive access to D while readers can read the same spot with other readers

```
RoomEmpty = S(1), Mutex = S(1)
NReader = 0
```

```
// Writer
While (True) {
  Wait(RoomEmpty);
  Modifies data
  Signal(RoomEmpty)
}
```

```
// Reader
While (True) {
  Wait(Mutex);
  NReader++;
  If (NReader == 1)
    // Ensure that no Writers
    Wait(RoomEmpty);
  Signal(Mutex);

  Read data

  Wait(Mutex);
  NReader--;
  If (NReader == 0)
    Signal(RoomEmpty);
  Signal(Mutex);
}
```

- Readers could potentially starve the writer since it could continue reading infinitely and then the writer cannot start (since it waits for the room to be empty)

Dining Philosophers Problem

- 5 philosophers seat in round fashion and there’s 5 single chopsticks between each pair of philosopher; each philosopher must pick up both left and right chopsticks to eat
- Common problems: deadlock if all pick up at the same time, livelock if they keep pick up and putting down

Never deadlock: if 1 philosopher picks up right first, or if missing 1 philosopher

```
#define N 5
#define LEFT ((i+N-1) % N)
#define RIGHT ((i+1) % N)
#define THINKING 0
#define HUNGRY 1
#define EATING 2
int state[N];
Semaphore mutex = 1;
Semaphore s[N];

void philosopher(int i){
    while (TRUE){
        Think();
        takeChpStcks(i);
        Eat();
        putChpStcks(i);
    }
}
```

```
void takeChpStcks(i) {
  wait(mutex);
  state[i] = HUNGRY;
  safeToEat(i);
  signal(mutex);
  wait(s[i]);
}

void safeToEat(i) {
  if ((state[i] == HUNGRY) &&
        (state[LEFT] != EATING) &&
        (state[RIGHT] != EATING)) {
    state[i] = EATING;
    signal(s[i]);
  }
}

void putChpStcks(i) {
  wait(mutex);
  state[i] = THINKING;
  safeToEat(LEFT);
  safeToEat(RIGHT);
  signal(mutex);
}
```