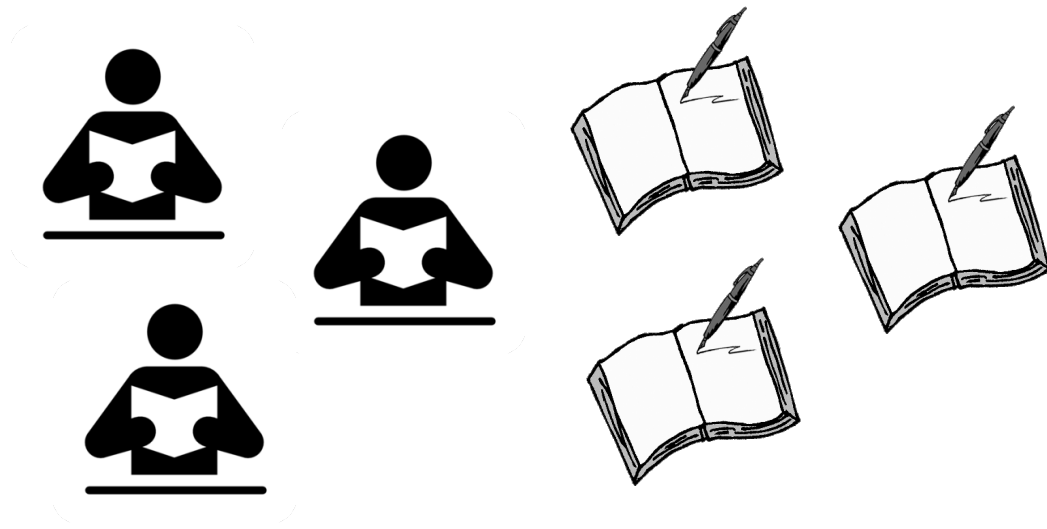


Readers/Writers Problem



Readers/Writers with Semaphores

```
class ReadWrite {  
    public:  
        void Read();  
        void Write();  
    private:  
        ???  
}
```

Readers/Writers with Semaphores

```
class ReadWrite {
    public:
        void Read();
        void Write();
    private:
        Semaphore wrt = 1;
}

ReadWrite::Write() {
    wrt.Wait();
    <perform write>
    wrt.Signal();
}
```

Readers/Writers with Semaphores

```
class ReadWrite {
    public:
        void Read();
        void Write();
    private:
        Semaphore wrt = 1;
        Semaphore mutex = 1;
        int readers = 0;
}

ReadWrite::Write() {
    wrt.Wait();
    <perform write>
    wrt.Signal();
}

ReadWrite::Read() {
    mutex.Wait();
    readers++;
    if (readers == 1)
        wrt.Wait();
    mutex.Signal();
    <perform read>
    mutex.Wait();
    readers--;
    if (readers == 0)
        wrt.Signal();
    mutex.Signal();
}
```

Readers/Writers with Monitors

```
private int numReaders = 0;
private int numWriters = 0;

private synchronized void prepareRead() {
    while (numWriters > 0) wait();
    numReaders++;
}

private synchronized void doneRead() {
    numReaders--;
    if (numReaders == 0) notify();
}

public ... doRead() {
    // reads NOT synchronized
    prepareRead();
    <do the reading>
    doneRead();
}
```

Readers/Writers with Monitors

```
private int numReaders = 0;
private int numWriters = 0;

private synchronized void prepareRead() {
    while (numWriters > 0) wait();
    numReaders++;
}

private synchronized void doneRead() {
    numReaders--;
    if (numReaders == 0) notify();
}

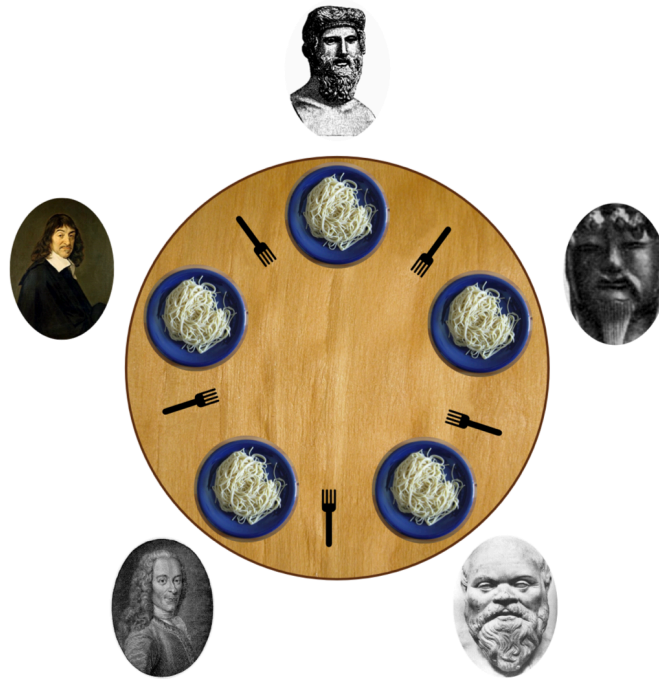
public ... doRead() {
    // reads NOT synchronized
    prepareRead();
    <do the reading>
    doneRead();
}

private void prepareWrite() {
    numWriters++;
    while (numReaders > 0) wait();
}

private void doneWrite() {
    numWriters--;
    notify();
}

public synchronized void doWrite (...) {
    // writes synchronized
    prepareWrite();
    <do the writing>
    doneWrite();
}
```

The Dining Philosophers



Dining Philosophers (first attempt)

```
Lock chopsticks[5];

void philosopher(int i) {
    while (true) {

        chopsticks[i].acquire(); // left chopstick
        chopsticks[(i+1)%5].acquire(); // right chopstick

        eat();

        chopsticks[i].release();
        chopsticks[(i+1)%5].release();

        think();
    }
}
```

Dining Philosophers with Monitors

```
monitor DiningPhilosophers {
    enum {THINK, HUNGRY, EAT}
        state[5];
    condition self[5];
}

void synchronized pickup(int i) {
    state[i] = HUNGRY;
    test(i);
    if (state[i] != EAT)
        self[i].wait();
}

void synchronized putdown(int i) {
    state[i] = THINK;
    // test left and right neighbors
    test((i+4)%5);
    test((i+1)%5);
}

void test(int n) {
    // check left and right of n
    if (state[(n+4)%5] != EAT &&
        state[n] == HUNGRY &&
        state[(n+1)%5] != EAT) {
        state[n] = EAT;
        self[n].signal();
    }
}

void init() {
    for (int i = 0; i < 5; i++)
        state[i] = THINK;
}
```