**Classic Synchronization Problems**  
- Classic Problems (CS4225)

**Ken Wong**  
Washington University

kenw@wustl.edu  
www.arl.wustl.edu/~kenw

---

**Bad Dining Philosopher Algorithm**

Semaphore  
fork[4] = {1, 1, 1, 1};  
Process philosopher(i) {
   do {
      Wait (R(i));  
      Wait(L(i));  
      ... Eat ...  
      Signal (R(i));  
      Signal (L(i));  
      ... Think ...  
   } until (DONE);
}

where  
R(i) := fork[i]  
L(i) := fork[(i+1) mod 4]  
"defined to be"

- Algorithm deadlocks
  - How? Why?
- Why do we care?
- Fix: 2-philosopher case
  - Increase resources
  - Larger atomic operation
  - Asymmetric algorithm
- Extend to N philosophers?
- What about starvation?
  - Only 1 philosopher starves?

---

**The Dining Philosopher Problem**

- **Philosopher States**
  - Thinking
    - Has no forks
  - Hungry
    - Wants both forks
  - Eating
    - Has both forks
- **Example**
  - 4 philosophers: 0, 1, 2, 3
  - 4 forks

---

**An Assymetric Algorithm**

Semaphore  
fork[4] = {1, 1, 1, 1};  
Process philosopher(i) {
   do {
      Wait (A(i));  
      Wait(B(i));  
      ... Eat ...  
      Signal (B(i));  
      Signal (A(i));  
      ... Think ...  
   } until (DONE);
}

where  
A(i) := fork[i], if even(i)  
A(i) := fork[(i+1) mod 4], otherwise

B(i) := fork[(i+1) mod 4], if even(i)  
B(i) := fork[i], otherwise

A(i) := R(i), if even(i)  
L(i), otherwise

B(i) := L(i), if even(i)  
R(i), otherwise

---

---

---

---
Readers-Writers (Writers-First)

- An object is shared among M readers and N writers
- Requirements
  - Only 1 writer at a time may modify the shared object
  - If a writer is modifying the object, no reader may read it
  - Any number of readers can simultaneously read the object
  - Writers have priority over readers
- Algorithm is more complex than Readers-First algorithm???
  - Can I just do some lexical substitutions that switch the role of the reader and the writer processes??

Readers-First Algorithm

```c
int nR = 0; // #active rdrs
Semaphore lock = 1, writeOk = 1;

Process reader() {  // reader process
  Wait(lock);
  nW = nW + 1;
  if (nW <= 1) Wait(readOk);
  Signal(lock);
  ... Read object ...
  Wait(lock);
  if (nW == 0) Signal(readOk);
  Signal(lock);
}

Process writer() {  // writer process
  Wait(writeOk);
  ... Write object ...
  Signal(lock);
  ... Read object ...
  Wait(lock);
  if (nW == 1) Wait(writeOk);
  nR = nR - 1;
  if (nR == 0) Signal(writeOk);
  Signal(lock);
}
```

Bad Writers-First Algorithm

```c
int nW = 0;
Semaphore lock = 1, readOk = 1;
Process writer() {
  Wait(lock);
  nW = nW + 1;
  if (nW <= 1) Wait(readOk);
  Signal(lock);
  ... Write object ...
  Wait(lock);
  if (nW == 0) Signal(readOk);
  Signal(lock);
}
```

Readers-First Algorithm

- Some shared variables
  - nW: Number of writers
- Writer Process
  - Provide critical section for writing object
  - Provide process queue for writers
  - First writer blocks all readers
  - Last writer unblocks readers
- Reader Process
  - Fit in with writer process control structure

Writers-First Algorithm

- Mutual exclusion for writing?
- Deadlock free?
- Fair?
- Responsive?
Better Writers-First Algorithm

```c
int nW = 0;
Semaphore enter = 1, writeOk = 1;
readOk = 1;
Process writer() {
    Wait(enter);
    nW = nW + 1;
    if (nW == 1) Wait(readOk);
    Signal(enter);
    Wait(writeOk);
    Write object ... Signal(writeOk);
    nW = nW - 1;
    if (nW == 0) Signal(readOk);
    Signal(enter);
}
```

- Mutual exclusion for writing?
- Deadlock free?
- Fair?
- Responsive?

Conditional Critical Region

- B(i) is a boolean expression for process i
- Process i waits until B(i) is True and the critical section is not in use
- The critical section can change the value of any B(i)
- How would you implement the entry and exit sections?

Implementation Notes

- Need to reevaluate B(i) for each process when a process exits the critical section
- After reevaluating all B(i), allow only 1 process whose B(i) is true to enter the critical section
- Sketch 1

```
Entry Section:
Get lock;
while (Not B(i)) {
    Release lock;
    Enter Q;
    Wait for wakeup signal;
    Get lock;
}
```

A Troublesome Case

- Process 1 is in the critical section
- Process 2 is at the head of the reevaluation queue Q
- Process 3 is in the reevaluation queue Q
- Suppose process 1 leaves the critical section and
  » B(3) is True
  » B(2) is False
- What happens to process 2 and 3?