# CSc 360
# Operating Systems
# Process Synchronization

Jianping Pan

Fall 2006



# Review

- Race condition
  - concurrent access to shared data: inconsistency
    - dependent on the order of execution at CPU level!
- Critical section
  - where shared data are accessed
- Mutual exclusion
  - pthread_mutex_lock(), pthread_mutex_unlock()
  - how is mutual exclusion implemented indeed?

# Properties of "solutions"

- Mutual exclusion
  - no more than one process in the critical section
- Making process
  - if no process in the critical section, one can in
- Bounded waiting
  - for processes that want to get in the critical section, their waiting time is bounded



# Problem formulation

- Only 2 processes, $P_0$ and $P_1$
- General structure of $P_i$ (other process $P_j$)

```
do {
    entry section
        critical section
    exit section
        reminder section
} while (1);
```

- Processes may ***share*** some common variables to synchronize their actions
  - do not get into loop!

# Algorithm 1

- Shared variables
  - int turn;
    initially **turn = 0**
  - turn == i Þ $P_i$ can enter its critical section
- Process $P_i$

```
do {
    while (turn != i) ;
        critical section
    turn = j;
        reminder section
} while (1);
```

- Fate on other's hands: any problems?



# Algorithm 2

- Shared variables
  - **boolean flag[2]**;
    initially **flag [0] = flag [1] = false.**
  - **flag [i] = true** Þ $P_i$ ready to enter its critical section
- Process $P_i$

```
do {
    flag[i] := true;
    while (flag[j]) ;
    critical section
    flag [i] = false;
        remainder section
} while (1);
```

- Fight for access: any problems?

# Dekker's solution

- Combined shared variables of Algos 1 and 2
- Process Pi

```
while (true) {
    flag[i] = true;
    while (flag[j]) {
      if (turn == j) {
          flag[i] = false;
          while (turn == j) { }
          flag[i] = true;
      }
    /* critical section */
    turn = j;
    flag[i] = false;
    }
}
```

- Be polite: meet all three requirements; solve the critical-section problem for ***two*** processes



# Peterson's solution

- A simpler solution
  - combined shared variables of Algorithms 1 and 2
- Process $P_i$

```
do {
    flag [i]:= true;
    turn = j;
    while (flag [j] and turn == j) ;
  /* critical section */
    flag [i] = false;
    /* remainder section */
} while (1);
```

- Meet all three requirements; solve the critical-section problem for ***two*** processes

# This lecture

- Process synchronization
  - software solution for 2 processes
    - Peterson's solution
- Explore further
  - Lamport's bakery algorithm
    - for ***n*** processes
    - it's time to google!



# Next lecture

- Process synchronization
  - other alternatives (read OSC7Ch6)