



More Threads and Synchronization

1



Review of Scheduler Implementation

Test your Understanding

- In the context switch algorithm we described, when the scheduler is called to select the next thread, which stack does it use (e.g., for its local variables)?
- If you are given a user-level thread package with a function that implements a voluntary `thread_yield()`, how can you “emulate” hardware interrupts and do preemptive scheduling?

2



Synchronization Primitives

- What are examples of interactions between threads that need help with synchronization?

3



Synchronization Primitives

- What are examples of interactions between threads that need help with synchronization?
 - Avoid editing the same data at the same time (mutual exclusion)
 - Wait for output (or another event to happen)
 - Enqueue/dequeue problem
 - Readers/writer problem
 - Barriers and rendezvous

4



Mutual Exclusion Example

- Processes and threads can be preempted at arbitrary times, which may generate problems.
- Example: What is the execution outcome of the following two threads (initially $x=0$)?

Thread 1:

Read X
Add 1
Write X

Thread 2:

Read X
Add 1
Write X

5



Critical Region Requirements

- Mutual Exclusion
- Progress
- Bounded Wait

No Speed and Number of CPU assumptions

6



Critical Region Requirements

- **Mutual Exclusion:** No other process must execute within the critical section while a process is in it.
- **Progress:** If no process is waiting in its critical section and several processes are trying to get into their critical section, then entry to the critical section cannot be postponed indefinitely.

7



Critical Region Requirements

- **Bounded Wait:** A process requesting entry to a critical section should only have to wait for a bounded number of other processes to enter and leave the critical section.
- **Speed and Number of CPUs:** No assumption may be made about speeds or number of CPUs.

8

Bounded Wait

Mutual Exclusion

Progress



Bounded Wait

Mutual Exclusion

Progress

No cutting in!







When to Break Bounded Wait?

- Are there examples of cases where we do not care for bounded wait for all threads?

13



Enforcing Mutual Exclusion

- Software Solutions
- Hardware solutions
 - Disabling Interrupts; Test-and-set; Swap (Exchange)
- Semaphores

14



Lock Variables

```
while (lock) { /* spin spin spin spin */  
lock = 1;  
/* EnterCriticalSection; */  
    access shared variable;  
/* LeaveCriticalSection; */  
lock = 0;
```

15



Lock Variables

```
while (lock) { /* spin spin spin spin */  
lock = 1;  
/* EnterCriticalSection; */  
    access shared variable;  
/* LeaveCriticalSection; */  
lock = 0;
```

What's the problem?

16



Turn-based Mutual Exclusion with Strict Alternation

```
...  
while ( turn != my_process_id) { }  
access shared variables  
turn = other_process_id  
...
```

17



Turn-based Mutual Exclusion with Strict Alternation

```
...  
while ( turn != my_process_id) { }  
access shared variables  
turn = other_process_id  
...
```

What's the problem?

18



Other Flag Mutual Exclusion

```
int owner[2] = { false, false }  
  
...  
while (owner[other_process_id] ) { }  
owner[my_process_id] = true  
access shared variables  
owner[my_process_id] = false  
  
...
```

19



Other Flag Mutual Exclusion

```
int owner[2] = { false, false }  
  
...  
while (owner[other_process_id] ) { }  
owner[my_process_id] = true  
access shared variables  
owner[my_process_id] = false  
  
...
```

What's the problem?

20



Two Flag Mutual Exclusion

```
int owner[2] = { false, false }  
  
...  
owner[my_process_id] = true  
while (owner[other_process_id] ) { }  
access shared variables  
owner[my_process_id] = false  
  
...
```

21



Two Flag Mutual Exclusion

```
int owner[2] = { false, false }  
  
...  
owner[my_process_id] = true  
while (owner[other_process_id] ) { }  
access shared variables  
owner[my_process_id] = false  
  
...
```

What's the problem?

22



Two Flag and Turn Mutual Exclusion

```
int owner[2]={false, false}
int turn;
...
owner[my_process_id] = true;
turn = other_process_id;
while (owner[other_process_id] and turn == other_process_id ) { }
access shared variables
owner[my_process_id] = false
...
```

23



Two Flag and Turn Mutual Exclusion

```
int owner[2]={false, false}
int turn;
...
owner[my_process_id] = true;
turn = other_process_id;
while (owner[other_process_id] and turn == other_process_id ) { }
access shared variables
owner[my_process_id] = false
...
```

Peterson Solution

24



Hardware Solutions

- To guarantee mutual exclusion, **hardware support** could help by allowing **disabling interrupts**

```
While(true) {  
    /* disable interrupts */  
    /* critical section */  
    /* enable interrupts */  
    /* remainder */  
}
```

- testandset (flag)