## Spin Locks

- Busy waiting (*spinning*) is injurious on mono-processors!
- On *multi-processors*, spinning may be faster than context switch
- If process holding lock is descheduled, spinning wastes CPU time
- Locks with *bounded spin* may be provided by OS.

## Caveats of Low-level Sharing

- Consider $\textbf{var}\ x, y : int; \quad x := 1; \quad y := 2$
- $x$ and $y$ may not be assigned as expected:
  - The *compiler* may reorder the assignments
  - The *processor* may reorder the store instructions
  - The *memory system* may reorder the write operations
  - The *compiler/processor* may keep values in registers
- Proper ordring must be ensured by *synchronization instructions*

Java low-level memory model

- Access to variables (except `long` and `double`) is *atomic*
- Monitor entry/exit synchronize memory operations
- Access to `volatile` variables is always synchronized

## Critical Regions with Shared Variables

### Test-and-Set Solution

- Let TS(**var** X) = ⟨**var** r; (r, X) := (X, 1); **return** r⟩
- **var** lock : int := 0;

```
process P1
   loop
      while TS(lock) = 1 do skip;
      critical section1;
      lock := 0;
      noncritical section1
   end loop
```

```
process P2
   loop
      while TS(lock) = 1 do skip;
      critical section2;
      lock := 0;
      noncritical section2
   end loop
```

## Critical Regions with Shared Variables

### Non-spinning Test-and-Set Solution

- Let TS(**var** X) = ⟨**var** r; (r, X) := (X, 1); **return** r⟩
- **var** lock : int := 0; s : semaphore := 0;

```
process P1
   loop
      while TS(lock) = 1 do  P(s);
      critical section1;
      lock := 0;
      V(s)
      noncritical section1
   end loop
```

```
process P2
   loop
      while TS(lock) = 1 do  P(s);
      critical section2;
      lock := 0;
      V(s);
      noncritical section2
   end loop
```

- Uses at least one semaphore operation for each entry/exit
- Semaphore should be binary

## Critical Regions with Shared Variables

### Non-spinning Test-and-Set Attempt

- Let $\mathtt{TS}(\textbf{var}\ \mathtt{X}) = \langle \textbf{var}\ \mathtt{r};\ \ (\mathtt{r},\mathtt{X}) := (\mathtt{X},1);\ \ \textbf{return}\ \mathtt{r}\,\rangle$

- $\textbf{var}\ \mathtt{lock} : \mathtt{int} := 0;\ \ \mathtt{s} : \mathtt{semaphore} := 0;\ \ \mathtt{waiting} : \mathtt{bool} := \mathtt{false};$

```
process P1
   loop
      while TS(lock) = 1 do
         {waiting := true; P(s)};
      critical section1;
      lock := 0;
      if waiting then
         {waiting := false; V(s)};
      noncritical section1
   end loop
```

```
process P2
   loop
      while TS(lock) = 1 do
         {waiting := true; P(s)};
      critical section2;
      lock := 0;
      if waiting then
         {waiting := false; V(s)};
      noncritical section2
   end loop
```

- Suffers from *race conditions*

## Futex (Fast user-space mutual exclusion)

### Idea

- Combine user-space atomic operations with OS-based blocking
- Atomic test and wait using *read-check-modify* principle

### Data Type

- $\textbf{type}\ \mathtt{futex} = \ \textbf{struct}\ \{\ \underbrace{\mathtt{val} : \mathtt{int}}_{\text{userspace}};\ \underbrace{\mathtt{queue} : \mathtt{condition\_queue}}_{\text{kernelspace}}\ \}$

- Fields related by hash-table based *mapping*

### Kernel operations

- $\mathtt{wait\_if\_eq}(\textbf{var}\ \mathtt{f} : \mathtt{futex}, \mathtt{old} : \mathtt{int}) \triangleq \langle \textbf{if}\ \mathtt{f.val} = \mathtt{old}\ \textbf{then}\ \mathtt{wait}(\mathtt{f.queue})\,\rangle$

- $\mathtt{wake}(\textbf{var}\ \mathtt{f} : \mathtt{futex}, \mathtt{n} : \mathtt{int}) \triangleq \langle \textbf{for}\ 1..\mathtt{n}\ \textbf{do}\ \mathtt{signal}(\mathtt{f.queue})\,\rangle$

## Futex Example: Critical Region

- Let $\mathtt{TS}(\textbf{var}\ \mathtt{X}) = \langle \textbf{var}\ \mathtt{r};\ \ (\mathtt{r}, \mathtt{X}) := (\mathtt{X}, 1);\ \ \textbf{return}\ \mathtt{r} \rangle$

  $\mathtt{INC}(\textbf{var}\ \mathtt{x} : \mathtt{int}) = \langle \mathtt{x} := \mathtt{x} + 1;\ \textbf{return}\ \mathtt{x} \rangle$

  $\mathtt{DEC}(\textbf{var}\ \mathtt{x} : \mathtt{int}) = \langle \mathtt{x} := \mathtt{x} - 1;\ \textbf{return}\ \mathtt{x} \rangle$

- **var** f : futex; waiting : int := 0;

```
f.val := 0;
process P1
   loop
      while TS(f.val) = 1 do
         {INC(waiting) ;
          wait_if_eq(f, 1);
          DEC(waiting); }
      critical section1;
      f.val := 0;
      if waiting > 0 then  wake(f, 1);
      noncritical section1
   end loop
```

```
process P2
   loop
      while TS(f.val) = 1 do
         {INC(waiting) ;
          wait_if_eq(f, 1);
          DEC(waiting); }
      critical section2;
      f.val := 0;
      if waiting > 0 then  wake(f, 1);
      noncritical section2
   end loop
```