High Performance Computing
Lecture 20

Matthew Jacob

Indian Institute of Science

Implementing a Lock

```
int L=0;                    /* 0: lock available */

AcquireLock(L):
    while (L==1);   /* `BUSY WAITING' */
    L = 1;

ReleaseLock(L):
    L = 0;
```

Why this implementation fails

while (L == 1) ;

L = 1;

```
wait:   LW      R1, Addr(L)
        BNEZ R1, wait
        ADDI    R1, R0, 1
        SW      R1, Addr(L)
```

Why this implementation fails

Process 1	Process 2
LW R1 = 0	
CSwitch	
	LW R1 = 0
	BNEZ
	ADDI
	SW
	Enter CS
	CSwitch
BNEZ	
ADDI	
SW	
Enter CS	

time

```
wait:   LW      R1, Addr(L)
        BNEZ R1, wait
        ADDI    R1, R0, 1
        SW      R1, Addr(L)
```

Assume that lock L is currently available (L = 0) and that 2 processes, P1 and P2 try to acquire the lock L

IMPLEMENTATION ALLOWS PROCESSES P1 and P2 TO BE IN CRITICAL SECTION TOGETHER!

Why this implementation fails

```
while ( L == 1) ;

L = 1;
```

⇒

```
wait:   LW      R1, Addr(L)

        BNEZ R1, wait

        ADDI    R1, R0, 1

        SW      R1, Addr(L)
```

Busy Wait Lock Implementation

- Hardware support will be useful to implement a lock
- Example: Test&Set instruction
  - A machine instruction with one memory operand

```
Test&Set  Lock
    tmp = Lock
    Lock = 1
    return tmp
```

Where these 3 steps happen atomically or indivisibly.

i.e., all 3 happen as one operation (with nothing happening in between)

Atomic Read-Modify-Write (RMW) instruction

Busy Wait Lock with Test&Set

Lock variable declared as int L

L == 0 means that the lock is available

L == 1 means that the lock is in use

AcquireLock(L)

```
while (Test&Set(L))  /*  busy wait */  ;
/  Busy wait until L has been Test&Set from 0 to 1
/      i.e., the return value from Test&Set is 0
```

ReleaseLock(L)

```
L = 0;
```

Busy Wait Lock with Test&Set

AcquireLock(L): while (Test&Set(L)) ;

ReleaseLock(L): L = 0;

P1	P2	P3
while(Test&Set(L));	while(Test&Set(L));	while(Test&Set(L));
Critical Section	Critical Section	Critical Section
L=0;	L=0;	L = 0;

Suppose that process P1 is in its Critical Section.

Processes P2 and P3 are trying to Acquire the Lock in order to enter their Critical Sections

Busy Wait Lock with Test&Set

AcquireLock(L): while (Test&Set(L)) ;

ReleaseLock(L): L = 0;

P1	P2	P3
while(Test&Set(L));	while(Test&Set(L));	while(Test&Set(L));
Critical Section	Critical Section	Critical Section
L=0;	L=0;	L = 0;

The lock L == 1 due to Test&Set(L) that was executed by P1

When P2 and P3 execute Test&Set(L), they overwrite the 1 and get a return value of 1

Then P1 exits its critical section

Busy Wait Lock with Test&Set

AcquireLock(L): while (Test&Set(L)) ;

ReleaseLock(L): L = 0;

P1	P2	P3
while(Test&Set(L));	while(Test&Set(L));	while(Test&Set(L));
Critical Section	Critical Section	Critical Section
L=0;	L=0;	L = 0;

More on Locks

- Other names for this kind of lock
  - Mutex
  - Spin wait lock
  - Spinlock
  - Busy wait lock
- There are also locks where instead of busy waiting, an unsuccessful process gets blocked by the operating system
  - i.e., moved into the Waiting state until the lock becomes available

Semaphore

- A more general synchronization mechanism
- Operations: *P* (wait) and *V* (signal)
- *P*(S)
  - if S is nonzero, decrements S and returns
  - Else, blocks the process until S becomes nonzero, when the process is restarted
    - After restarting, decrements S and returns
- *V*(S)
  - Increments S by 1
  - If there are other processes blocked for S, restarts exactly one of them

Critical Section Problem & Semaphore

- Initialize a Semaphore S = 1
- Surround each critical section in the concurrent program by calls to P(S) and V(S)

Critical Section Problem & Semaphore

- Initialize a Semaphore S = 1
- Surround each critical section in the concurrent program by calls to P(S) and V(S)

<u>Process P1</u>

P(S);

Critical Section code

V(S);

```
if (S != 0)
        S--;  return        / S==0
else
        block process until S!=0
        S--;  return        / S==0
```

```
S++                     / S==1
Unblock a process blocked on S
```

Semaphore Examples

- The previous example showed how a semaphore can be used to do the work of a mutex lock
- Semaphores can be used for other purposes as well

Semaphore Examples

- Semaphores can do more than mutex locks
- Example: Initialize semaphore S =10
- Suppose that processes surround code by P(S), V(S) as with the previous example

```
P(S);
  code
V(S);
```

P(S):

```
if (S != 0)
        S--;  return
else
        block process until S!=0
        S--;  return
```

V(S):

```
S++
Unblock a process blocked on S
```

Semaphore Examples

- Semaphores can do more than mutex locks
- Example: Initialize semaphore S =10
  - 10 processes will be allowed to proceed
  - Processes beyond that will be blocked until one of the first 10 executes V(S)

Semaphore Examples

- Semaphores can do more than mutex locks
- Example: Consider our concurrent program where process P1 reads 2 matrices; process P2 multiplies them & process P3 outputs the product
    - Semaphores $S_1 = 0$ $S_2 = 0$

Process P1	Process P2	Process P3
	$P(S_1)$	$P(S_2)$
Read A[ ], B[ ]	C[ ] = A[ ] * B[ ]	Write C[ ]
$V(S_1)$	$V(S_2)$	