Synchronization
Spinlocks - Semaphores

Summer 2013
Cornell University
Today

• How can I synchronize the execution of multiple threads of the same process?
• Example
• Race condition
• Critical-Section Problem
• Spinlocks
• Semaphors
• Usage
Problem Context

- Multiple threads of the same process have:
  - Private registers and stack memory
  - Shared access to the remainder of the process “state”

- Preemptive CPU Scheduling:
  - The execution of a thread is interrupted unexpectedly.

- Multiple cores executing multiple threads of the same process.
Share Counting

- Mr Skroutz wants to count his $1-bills.
- Initially, he uses one thread that increases a variable `bills_counter` for every $1-bill.
- Then he thought to accelerate the counting by using two threads and keeping the variable `bills_counter` shared.
Share Counting

\[ \text{bills\_counter} = 0 \]

- **Thread A**
  
  ```
  while (machine\_A\_has\_bills) 
  \textit{bills\_counter}++
  ```

- **Thread B**
  
  ```
  while (machine\_B\_has\_bills) 
  \textit{bills\_counter}++
  ```

- **print bills\_counter**

- **What it might go wrong?**
Share Counting

- Thread A
  
  \[
  r1 = bills\_counter \\
  r1 = r1 +1 \\
  bills\_counter = r1
  \]

- Thread B
  
  \[
  r2 = bills\_counter \\
  r2 = r2 +1 \\
  bills\_counter = r2
  \]

- If \( bills\_counter = 42 \), what are its possible values after the execution of one A/B loop?
Shared counters

• One possible result: everything works!

• Another possible result: lost update!

• Called a “race condition”.
Race conditions

- Def: a timing dependent error involving shared state
  - It depends on how threads are scheduled.
- Hard to detect
Critical-Section Problem

\[ bills\_counter = 0 \]

- **Thread A**
  ```plaintext
  while (my\_machine\_has\_bills)
  - enter critical section
  bills\_counter++
  - exit critical section
  ```

- **Thread B**
  ```plaintext
  while (my\_machine\_has\_bills)
  - enter critical section
  bills\_counter++
  - exit critical section
  ```

\textit{print} bills\_counter
Critical-Section Problem

- The solution should satisfy:
  - Mutual exclusion
  - Progress
  - Bounded waiting

- enter section
  - critical section
  - exit section
  - remainder section
General Solution

- LOCK
- A process must acquire a lock to enter a critical section.
- Hardware or Software-based implementation
TestAndSet

- **Hardware** instruction.
- **Test** and **modify** the content of **one word atomically**.

```c
boolean TestAndSet(boolean *target){
    boolean rv = *target;
    *target = TRUE;
    return rv;
}
```
Share Counting

\[
bills\_counter = 0
\]
\[
lock = FALSE
\]

• Thread A

\[
while \ (machine\_A\_has\_bills)\
    while \ (TestAndSet(&lock))
    ;
    bills\_counter++
    lock = FALSE;
\]

print bills\_counter

\[
\]

• Thread B

\[
while \ (machine\_B\_has\_bills)\
    while \ (TestAndSet(&lock))
    ;
    bills\_counter++
    lock = FALSE;
\]

\[
\]

boolean TestAndSet(boolean *target){
    boolean rv = *target;
    *target = TRUE;
    return rv;
}

\[
\]
Swap

- **Hardware** instruction.
- Swap the contents of two **words atomically**.

```c
void Swap (boolean *a, boolean *b){
    boolean temp = *a;
    *a = *b;
    *b = temp;
}
```
Share Counting

\[ \text{bills\_counter} = 0 \]
\[ \text{lock} = \text{FALSE} \]

- **Thread A**
  
  \[
  \text{while} \ (\text{machine\_A\_has\_bills})\
  \quad \text{keyA} = \text{TRUE}; \\
  \quad \text{while} \ (\text{keyA} == \text{TRUE}) \\
  \quad \quad \text{Swap}(&\text{lock}, &\text{keyA}); \\
  \quad \text{bills\_counter}++ \\
  \quad \text{lock} = \text{FALSE}; \\
  \]

- **Thread B**
  
  \[
  \text{while} \ (\text{machine\_B\_has\_bills})\
  \quad \text{keyB} = \text{TRUE}; \\
  \quad \text{while} \ (\text{keyB} == \text{TRUE}) \\
  \quad \quad \text{Swap}(&\text{lock}, &\text{keyB}); \\
  \quad \text{bills\_counter}++ \\
  \quad \text{lock} = \text{FALSE}; \\
  \]

**print** \( \text{bills\_counter} \)

```c
void Swap (boolean *a, boolean *b){
    boolean temp = *a;
    *a = *b;
    *b = temp;
}
```
But...

- TestAndSet and Swap are complicated for application programmers to use.
- What is the alternative?
Semaphores

• Integer value
• Atomic operations
  • wait
  • signal

```c
wait(S) {
    while S <= 0
    ;
    S--;  
}

signal(S) {
    S++; 
}
```
Share Counting

\[\text{bills\_counter} = 0\]
\[S = 1\]

- **Thread A**
  
  \[
  \text{while (machine\_A\_has\_bills)\{}
  
  \hspace{1cm} \text{wait}(S);
  
  \hspace{1cm} \text{bills\_counter}++
  
  \hspace{1cm} \text{signal}(S);
  
  \}
  \]

- **Thread B**
  
  \[
  \text{while (machine\_B\_has\_bills)\{}
  
  \hspace{1cm} \text{wait}(S);
  
  \hspace{1cm} \text{bills\_counter}++
  
  \hspace{1cm} \text{signal}(S);
  
  \}
  \]

\textbf{print} \text{bills\_counter}
Spinlock

- This implementation of Semaphors, TestAndSet and Swap are spinlocks.
  - They require **busy waiting**.
  - The waiting processes should **loop continuously** in the entry code.
- Valuable **CPU** cycles are **wasted**.
- Solution:
  - **Block** the waiting process.
  - **Signal** blocked process when the semaphore is “available”.
Semaphores

typedef struct {
    int value;
    struct process *list;
} semaphore;

wait (semaphore *S) {
    S->value--;
    if (S->value < 0) {
        add this process to S->list;
        block();
    }
}

signal (semaphore *S) {
    S->value++;
    if (S->value <= 0) {
        remove a process P from S->list;
        wakeup(P);
    }
}
Usage

- Binary semaphore (mutex)
  - Ranges between 0 and 1.
  - Ex. Only one process can access a resource.
- Counting semaphore
  - Ranges, usually, between 0 and N.
  - Ex. N resources and M processes that share the resources.
- Synchronization
  - Ranges between 0 and 1.
  - Ex. Process A should do task At after process B having done task Bt.
Today

• How can I synchronize the execution of multiple threads of the same process?
• Example
• Race condition
• Critical-Section Problem
• Spinlocks
• Semaphors
• Usage