Synchronization (1)

David A. Eckhardt
School of Computer Science
Carnegie Mellon University

de0u@andrew.cmu.edu

Status Rendezvous

Everybody *has* run simics?

- de0u+licenses@andrew
  - not de0u+licenses@*cs*
  - not de0u+*license*@andrew
- will generate a “bounce” message
  - https://www.simics.net/evaluation/scripts/academic.php

Handin

- Watch academic.cs.15-412.announce

Partner selection for Project 2

- de0u+partner@andrew
  - or de0u+partners@andrew (I am learning)
- By Tuesday 2002-03-04 23:59 EST

Steve is out of town

- Variant office hours this week
- See academic.cs.15-412.announce

Outline

Me vs. Chapter 7

- Mind your P's and Q's
- Atomic sequences vs. voluntary de-scheduling
  - "Sim City" example
- You *will* need to read the chapter
- Hopefully my preparation/review will clairfy it

Three critical-section necessities

Two-process solution

N-process "Bakery Algorithm"

Mind your P's and Q's

What you write

```
choosing[i] = true;
number[i] = max(number[0], number[1], ...) + 1;
choosing[i] = false;
```

What happens...

```
number[i] = max(number[0], number[1], ...) + 1;
choosing[i] = false;
```

Or maybe...

```
choosing[i] = false;
number[i] = max(number[0], number[1], ...) + 1;
```

My computer is broken?!

No, your computer is “modern”

- Processor “write pipe” queues memory stores
  - ...and coalesces “redundant” writes!
- Magic “memory barrier” instructions available
  - ...stall processor until write pipe is empty

Ok, now I understand

- Probably not
- http://www.cs.umd.edu/~pugh/java/memoryModel/
  - "Double-Checked Locking is Broken" Declaration
- See also “release consistency”

Textbook’s memory model

- ...*is* “what you expect”

Atomic sequences vs. voluntary de-scheduling

Two fundamental operations

- Atomic instruction sequence
- Voluntary de-scheduling

Multiple implementations of each

- Uniprocessor vs. multiprocessor
- Special hardware vs. special algorithm
- Different OS techniques
- Performance tuning for special cases

Multiple client abstractions

- Textbook covers: semaphore, critical region, monitor
- *Very* relevant
  - mutex/condition variable (POSIX pthreads)
  - Java “synchronized” keyword (3 uses)

Atomic instruction sequence

Problem class

- *Short* sequence of instructions
- Nobody else may interleave same sequence
    - or a “related” sequence
- “Typically” nobody is trying

Example

- Multiprocessor simulation (think: “Sim City”)
- Coarse-grained “turn” (think: hour)
- Lots of activity within turn
    - Think: M:N threads, M=objects, N=#processors

Commerce

Multithreaded commerce

Customer 1	Customer 2
cash = store->cash;	cash = store->cash;
cash += 50;	cash += 20;
personal_cash -= 50;	personal_cash -= 20;
store->cash = cash;	store->cash = cash;

- Should the store call the police?
- Is deflation good for the economy?

Observations

- Instruction sequence is “short”
  - Ok to force competitors to wait
- Probability of collision is “low”
  - Avoid expensive exclusion method

Voluntary de-scheduling

Problem class

- “Are we there yet?”
- “Waiting for Godot”

Example - “Sim City” disaster daemon

```
while (date < 1906-04-18) sleep(date);
while (hour < 5) sleep(hour);
iterate over squares:
 wreak_havoc(square);
```

Observations

- Making others wait is wrong
  - It will be a while
  - We *want* others to run - they *enable* us
- CPU *de*-scheduling is an OS service!

Voluntary de-scheduling

While (not ready)

- *Atomic instruction sequence*
  - Scan shared state
  - State indicates “it will be a while”
  - Register (in state) interest in the happy event
  - Release control of shared state
  - *De-schedule* yourself (until somebody says “event!”)

Nomenclature

Textbook's code skeleton / naming

```
do {
  entry section
  critical section
    ...computation on shared state...
  exit section
  remainder section
    ...private computation...
} while (1);
```

What's muted by this picture?

- Critical section contents
  - Sleep?
  - Or just atomic sequence?

“Critical Section Problem”

“Entry/exit protocol problem”

- *Mutual Exclusion*
    - At most one process executing critical section
- *Progress*
    - Choosing next entrant cannot involve nonparticipants
    - Choosing protocol must have bounded time
- *Bounded waiting*
    - Cannot wait forever once you begin entry protocol
    - ...bounded number of entries by others

Conventions for 2-process algorithms

- P[i] = “us”, P[j] = “the other”
- {i,j} = {0,1}
- j == 1 - i

First idea

Taking turns

```
int turn = 0;

while (turn != i)
  ;
...critical section...
turn = j;
```

How'd we do?

- Mutual exclusion - yes
- Progress - no
  - *Strict* turn-taking is fatal
  - If P[i] never tries to enter, P[j] will wait forever

Second idea

Registering interest

```
boolean want[2] = {false, false};

want[i] = true;
while (want[j])
  ;
...critical section...
want[i] = false;
```

Evaluation

- Mutual exclusion - yes
- Progress - almost

Customer 0	Customer 1
want[0] = true	want[1] = true
while (want[1]) ;	while (want[0]) ;

Rubbing two ideas together

Taking turns when necessary

```
boolean want[2] = {false, false};
int turn = 0;

want[i] = true;
turn = j;
while (want[j] && turn == j)
  ;
...critical section...
want[i] = false;
```

Proof sketch of exclusion, by contradiction

- Both in c.s. implies want[i] == want[j] == true
- That implies both while loops exited because “turn != j”
- Cannot have (turn == 0 && turn == 1), so one exited first
- w.l.o.g., P0 exited first, so turn==0 before turn==1
- So P1 had to set turn==0 *before* P0 set turn==1
- So P0 could not see turn==0, could *not* exit loop first, C!

Bakery Algorithm

Take a ticket from the dispenser

- Unlike "reality", two people can get the same ticket number
- Sort by (lowest wallet dollar bill serial number, ticket number)

Two-phase entry protocol

- Pick a number
  - Look at all presently-available numbers
  - Add 1 to highest you can find
- Wait until you have the *lowest* number curently issued
  - Well, the lowest (serial,ticket) number, anyway

Bakery Algorithm

Code

```
boolean choosing[n] = { false, ... };
int number[n] = { 0, ... } ;

choosing[i] = true;
number[i] = max(number[0], number[1], ...) + 1;
choosing[i] = false;

for (j = 0; j < n; ++j) {
 while (choosing[j])
   ;
 while ((number[j] != 0) &&
   ((number[j], j) < (number[i], i)))
   ;
}
...critical section...
number[i] = 0;
```