Synchronization (1)

Dave Eckhardt
de0u@andrew.cmu.edu

Synchronization

- Makefile shakeup
- Documentation is done, right?
  - “Coding standards”
- Watch for announcements
  - Handin
  - Partner selection for Project 2
- ***How*** to watch for announcements
  - .../410/pub/410biff (an X program)
  - (or any other manner of your choice)

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 clarify it

Outline

- An intrusion from the “real world”
- Two fundamental operations
- Three necessary critical-section properties
- 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;
```

Mind your P's and Q's

- What you write

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

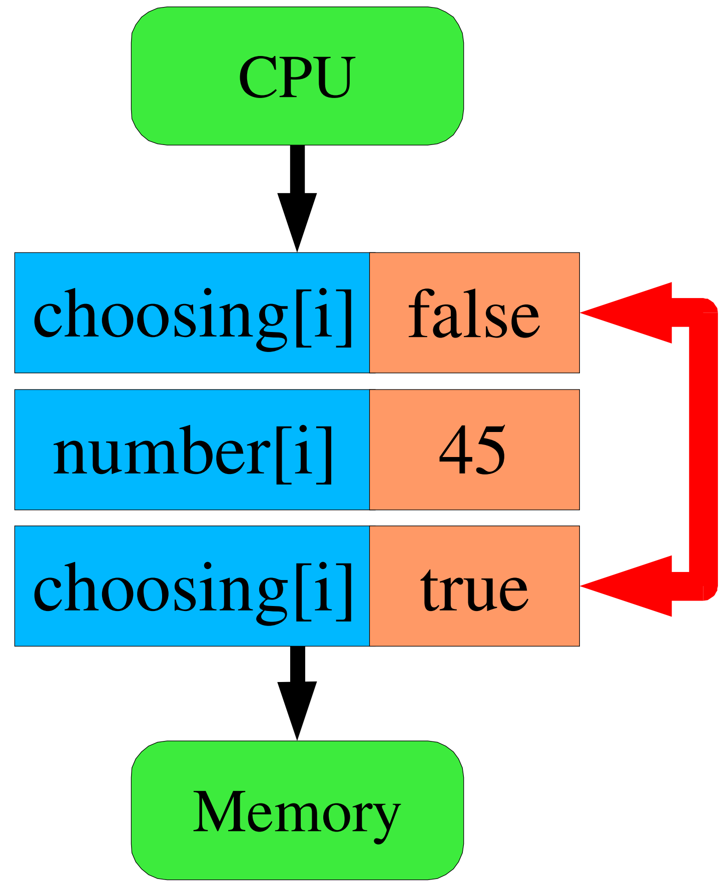
- Or maybe...

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

- “Computer Architecture for $200, Dave”...

My computer is broken?!

- No, your computer is "modern"
  - Processor "write pipe" queues memory stores
  - ...and coalesces "redundant" writes!
- Crazy?
  - Not if you're pounding out pixels!



My computer is broken?!

- 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”

Synchronization Fundamentals

- 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

Synchronization Fundamentals

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

Non-interference

- Multiprocessor simulation (think: “Sim City”)
  - Coarse-grained “turn” (think: hour)
  - Lots of activity within turn
  - Think: M:N threads, M=objects, N=#processors
- ***Most*** cars don't interact in a turn...
  - Must model those that do!

Commerce

Customer 0	*Customer 1*
`cash = store->cash;`	`cash = store->cash;`
`cash += 50;`	`cash += 20;`
`wallet -= 50;`	`wallet -= 20;`
`store->cash = cash;`	`store->cash = cash;`

Should the store call the police?

Is deflation good for the economy?

Commerce – Observations

- Instruction sequences are “short”
  - Ok to force competitors to wait
- Probability of collision is "low"
  - Want cheap non-interference method

Voluntary de-scheduling

- Problem domain
  - “Are we there yet?”
  - “Waiting for Godot”
- Example - "Sim City" disaster daemon

```
while (date < 1906-04-18) cwait(date);
while (hour < 5) cwait(hour);
for (i = 0; i < max_x; i++)
  for (j = 0; j < max_y; j++)
    wreak_havoc(i,j);
```

Voluntary de-scheduling

- Making others wait is wrong
  - It will be a while
- We don't ***want*** exclusion
- We ***want*** others to run - they ***enable*** us
- CPU ***de***-scheduling is an OS service!

Voluntary de-scheduling

- Wait pattern

```
LOCK WORLD
while (!(ready = scan_world()))
  UNLOCK WORLD
  WAIT_FOR(progress_event)
```

- Your partner/competitor will

```
SIGNAL(progress_event)
```

Nomenclature

- Textbook's code skeleton / naming

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

Nomenclature

- What's muted by this picture?
- What's in that critical section?
  - Quick atomic sequence?
  - Need for a long sleep?

Three Critical Section Requirements

- ***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

- Process[i] = "us"
- Process[j] = "the other process"
- i, j are ***process-local*** variables
  - {i,j} = {0,1}
  - j == 1 - i

First idea - "Taking Turns"

```
int turn = 0;
while (turn != i)
  ;
...critical section...
turn = j;
```

- 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;
```

- Mutual exclusion – yes
- Progress - ***almost***

Failing "progress"

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

It works the rest of the time!

“Taking turns when necessary”

- Rubbing two ideas together

```
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

- Both in c.s. implies want[i] == want[j] == true
- Thus 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!

Bakery Algorithm

- More than two processes?
  - Generalization based on bakery/deli counter
- Take a ticket from the dispenser
  - Unlike "reality", two people can get the same ticket number
  - Sort by (ticket number, process number)

Bakery Algorithm

- Phase 1 – Pick a number
  - Look at all presently-available numbers
  - Add 1 to highest you can find
- Phase 2 – Wait until you hold ***lowest*** number
  - Well, lowest (ticket, process) number

Bakery Algorithm

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

Bakery Algorithm

- Phase 1: Pick a number

```
choosing[i] = true;

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

choosing[i] = false;
```

Bakery Algorithm

- Phase 2: Wait to hold lowest number

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

Summary

- Memory is ***weird***
- Two fundamental operations
  - ***Brief exclusion*** for atomic sequences
  - ***Long-term yielding*** to get what you want
- Three necessary critical-section properties
- Two-process solution
- N-process "Bakery Algorithm"