Threads

CS439: Principles of Computer Systems

Bringing It All Together

- Processes
  - Abstraction for protection
  - Define address space
- Threads
  - Every process has at least one kernel thread
  - More threads can be created using either system calls (kernel threads) or libraries (user threads)
  - Help us model the real world and exploit concurrency in the system
  - Share (and communicate) through global and static data, share the heap, each has its own stack and full use of the registers
- Dual-Mode Execution
  - Processes and threads can execute in either mode
- CPU Schedulers
  - Schedule both kernel threads and processes
  - Since each process has one kernel thread, sometimes processes are treated as threads
  - Helps with the illusion of infinite resources!

Vocabulary

threads
logical parallelism
heap
TCB
address space
SDS

multi-threaded process
process
shared memory
kernel-level thread
user-level thread

thread type
stack
true parallelism
code
thread library

Question

Threads have their own …?

A. Address Space
B. PCB
C. Stack

Too Much Milk!

CS439: Principles of Computer Systems

Too Much Milk!

You

- Arrive home
- Look in the fridge; out of milk
- Go to store
- Buy milk
- Arrive home; put milk away

Your Roommate

- Arrive home
- Look in fridge; out of milk
- Go to store
- Buy milk
- Arrive home; put milk away
- Oh, no!

Too Much Milk!

- What do we want to happen?
  - Only one person buys milk at a time AND
  - Someone buys milk if you need it

  These are the correctness properties for this problem.
- What happened?
  - Lack of communication!

Race Conditions

- What would the result have been if:
  - your roommate had arrived home for the first time after you had come back from the store?
  - you arrived home after your roommate came back from the store?
  - you were at the store when your roommate came back, but your roommate waited to look in the fridge until after you were back from the store?
- Instances where the result changes based on scheduling are *race conditions*
- What guarantees do we have about how our people/threads will be scheduled?
- How can we solve this problem?

Too Much Milk: One Solution

You (Thread A)

```
if(noMilk && noNote)
{
   leave note;
   buy milk;
   remove note;
}
```

Your Roommate (Thread B)

```
if(noMilk && noNote)
{
   leave note;
   buy milk;
   remove note;
}
```

Does this work? A. Yes B. No

Too Much Milk: Another Solution

You (Thread A)

```
leave note A
if(noNote B)
  if(noMilk)
    buy milk;

remove note A
```

Your Roommate (Thread B)

```
leave note B
if(noNote A)
  if(noMilk)
    buy milk;

remove note B
```

Does this work? A. Yes B. No

Too Much Milk: Maybe This Solution

You (Thread A)

```
leave note A
while(note B)
   do nothing;

if(noMilk)
   buy milk;

remove note A
```

Your Roommate (Thread B)

```
leave note B
if(noNote A)
   if(noMilk)
      buy milk;

remove note B
```

Does this work? A. Yes B. No

Why is it correct?

Your Roommate (Thread B)

leave note B

if(noNote A)

  if(noMilk)

    buy milk;

remove note B

At this if, either there is a note A or not.

If not, it is safe for B to check and buy milk , if needed. (Thread A has not started yet.)

If yes, then thread A is checking and buying milk as needed or is waiting for B to quit, so B quits by removing note B.

Why is it correct?

You (Thread A)

leave note A

while(note B)

  do nothing;

if(noMilk)

  buy milk;

remove note A

At this while, either there is a note B or not.

If not, it is safe for A to buy since B has either not started yet or quit.

If yes, A waits until there is no longer a note B, and either finds milk that B bought or buys it if needed.

Why is it correct?

So Thread B buys milk (which Thread A finds) or not, but either way it removes note B. Since Thread A loops, it waits for B to buy milk or not, and then if B did not buy it, it buys the milk.

So it's correct, but… is it good?

1. It is too complicated. It was hard to convince ourselves this solution worked.
2. It is asymmetrical---thread A and thread B are different. *What would we need to do to add new threads?*
3. A is *busy waiting*, or consuming CPU resources despite the fact it is not doing any useful work.

Too Much Milk: Lock Solution

Lock lock;

You (Thread A)

```
Lock->Acquire();
if(noMilk)
   buy milk;
Lock->Release();
```

Your Roommate (Thread B)

```
Lock->Acquire();
if(noMilk)
   buy milk;
Lock->Release();
```

Race Conditions in Code

```
int global_x  = 0
char * global_str = NULL
int main(){
   int a = 3;
   thread_create(foo())
   thread_create(bar())
   a += 4;
   thread_join();
}
```

1) Where are the variables/data stored?
2) What data is shared?
3) What shared data is accessed or modified?

```
foo(){
   int b = 1;
   global_x++;
   if(global_str==NULL)
      b++;
}

bar(){
   int b = 1;
   if(global_x != 1)
      global_str=malloc(256);
 }
```

Summary

- Threads share the same address space
  - Processes have *separate* address spaces
  - Child processes start with a *copy* of their parent's address space
- Each thread is its own thread of control
  - Program counter, register values, execution stack
- It is easy for threads to inadvertently disrupt each other since they share the entire address space (!)
- Communication among threads is typically done through shared variables
- Operating system can switch from any kernel thread at any time