Programm- & Systemverifikation

Assignment 1

Georg Weissenbacher

184.741

Part 1 of Assignment 1 – Assertions

For each of the following examples:

- Provide (if possible) initial values for the unsigned 32-bit variables `x` and `y` such that the first assertion does not fail, but the second does.
- If this is not possible, use the substitution trick from the lecture and mathematical/logical reasoning to explain why.

1. `assert(y==41); x=(y+1); assert(x==42);`
2. `assert(x==y); y--; assert(x>y);`
3. `assert(x!=y && x!=0); y=(y%x); assert(y>0 && x!=0);`
   (In ANSI-C, % denotes the modulo operator)
4. `assert(x==y); x=x^y; y=x^y; x=x^y; assert(x==y);`
   (In ANSI-C, ^ denotes the XOR operator ($\oplus$))

Assertions and Concurrency

- *Locks* can be used to prevent simultaneous or concurrent access to critical regions or resources
- Simplified API:
  - `lock(A)` succeeds if lock `A` is available
  - `lock(A)` blocks if lock is already held/acquired (by this or another thread)
  - `unlock(A)` releases a lock previously acquired
  - `unlock(A)` never blocks

Assertions and Concurrency

- Deadlocks happen if locks are acquired in wrong order

```
lock (A);
  lock (B);
  unlock (B);
unlock (A);
```

```
lock (B);
  lock (A);
  unlock (A);
unlock (B);
```

Assertions and Concurrency

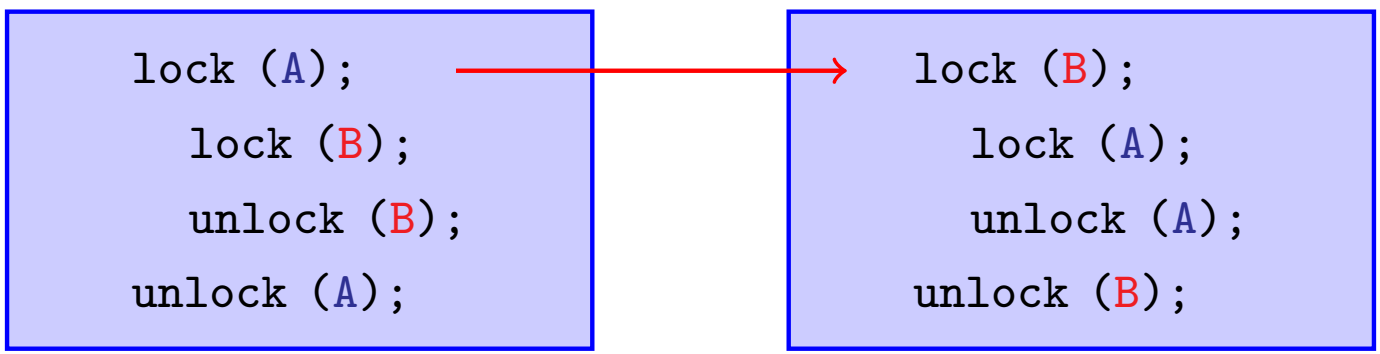
- Deadlocks happen if locks are acquired in wrong order
  - Thread one acquires lock A

```
lock (A);
  lock (B);
  unlock (B);
unlock (A);
```

```
lock (B);
  lock (A);
  unlock (A);
unlock (B);
```

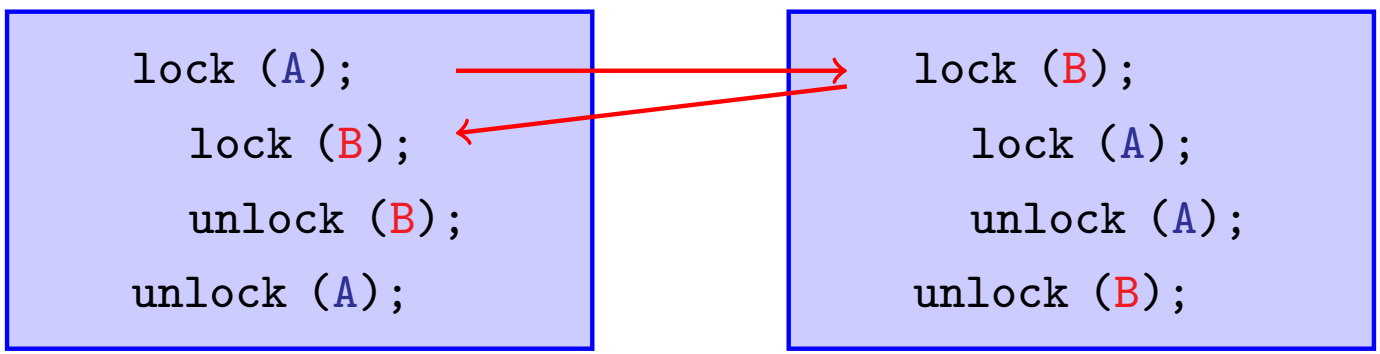
Assertions and Concurrency

- Deadlocks happen if locks are acquired in wrong order
  - Thread one acquires lock A
  - Thread two acquires lock B



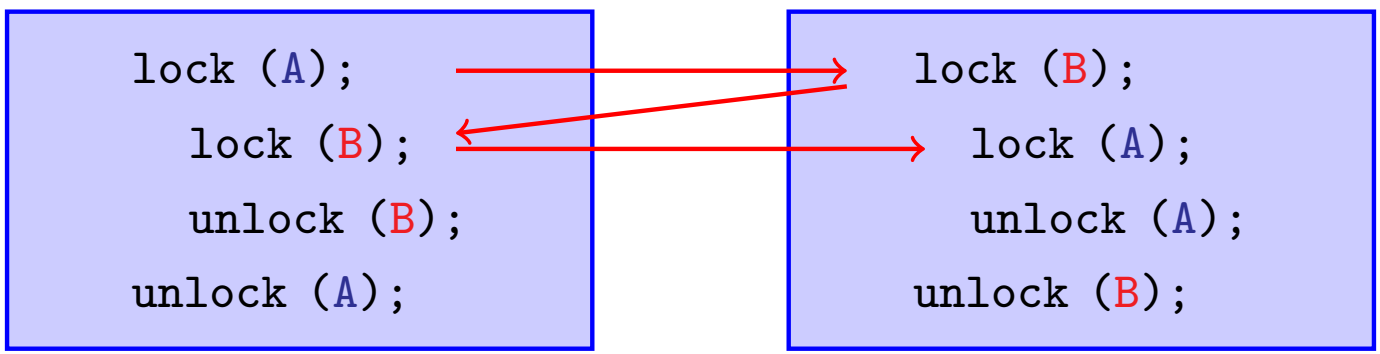
Assertions and Concurrency

- Deadlocks happen if locks are acquired in wrong order
  - Thread one acquires lock A
  - Thread two acquires lock B
  - Thread one waits for lock B (thread two still running)



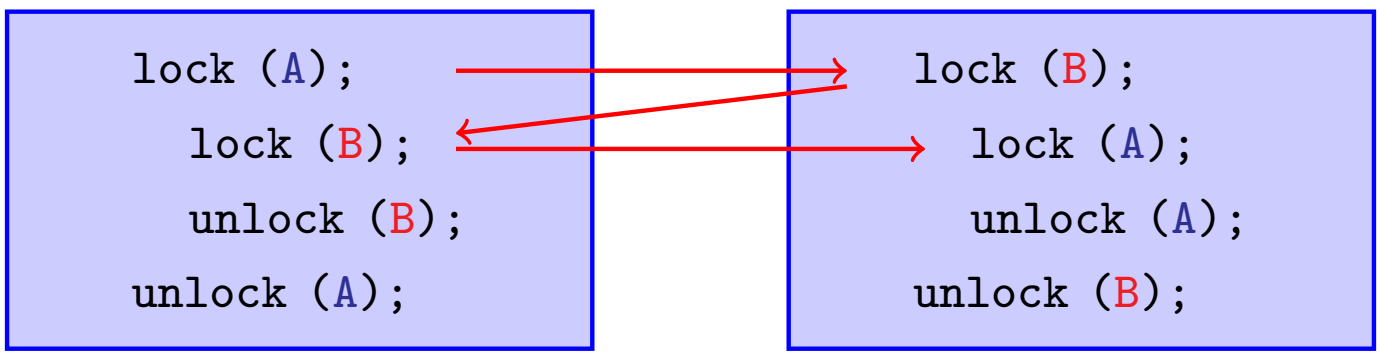
Assertions and Concurrency

- Deadlocks happen if locks are acquired in wrong order
  - Thread one acquires lock A
  - Thread two acquires lock B
  - Thread one waits for lock B
  - Thread two waits for lock A



Assertions and Concurrency

- Deadlocks happen if locks are acquired in wrong order
  - Thread one acquires lock A
  - Thread two acquires lock B
  - Thread one waits for lock B
  - Thread two waits for lock A
  - Now both threads are stuck...



Part 2 of Assignment 1

- Add assertions that fail if a deadlock is about to occur!
- Assertions must *not* fail if no deadlock occurs!
- **Hints:**
  - You need to augment the code with auxiliary code and variables indicating when a process is waiting for a lock
  - The assertions must be executed *before* the deadlock occurs

For the specialists among you: assume sequential consistency

Assertions and Concurrency: Solution

```
flagA = 0;
lock (A);
  flagA = 1;
  assert (!flagB);
  lock (B);
  flagA = 0;
  unlock (B);
unlock (A);
```

```
flagB = 0;
lock (B);
  flagB = 1;
  assert (!flagA);
  lock (A);
  flagB = 0;
  unlock (A);
unlock (B);
```

Note:

- If only one thread contains an assertion, then there's a potential deadlock without an assertion failure
- If `flagA` and `flagB` are reset after the inner locks are released, then there's a potential assertion failure even if the deadlock doesn't happen

Part 3 of Assignment 1

- Add an *inductive invariant* to the code
- Use it to show that the assertion after the loop holds
- Add comments to the code explaining
  - why your assertion is an inductive invariant
  - why it shows that the assertion after the loop holds

```
unsigned x = i;
unsigned y = j;
while (x != 0)
{
  x--;
  y++;
  assert (?); // add invariant here
}
assert ((i != j) || (y == 2 * i));
```

Inductive invariant: Solution

```
assert (true);
assert (j == j + (i - i));
unsigned x = i;
assert (j == j + (i - x));
unsigned y = j;
assert (y == j + (i - x));
while (x != 0)
{
  assert (y == j + (i - x));
  assert ((y + 1) == j + (i - (x - 1)));
  x--;
  assert ((y + 1) == j + (i - x));
  y++;
  assert (y == j + (i - x));
  // Number of iterations n := i - x
}
assert ((x == 0) && y == j + (i - x));
assert ((i != j) || (y == 2 * i));
```

Submitting your solution

- Your solution must be submitted via TUWEL by April 15, 4pm
  - Late submissions will not be accepted
- Answer all questions and submit your solution as a *single PDF*
- Make sure the file contains your student ID and your name
- You can get up to 5 points for each part of this assignment