ECE 3574: Thread Safe Queue

Changwoo Min

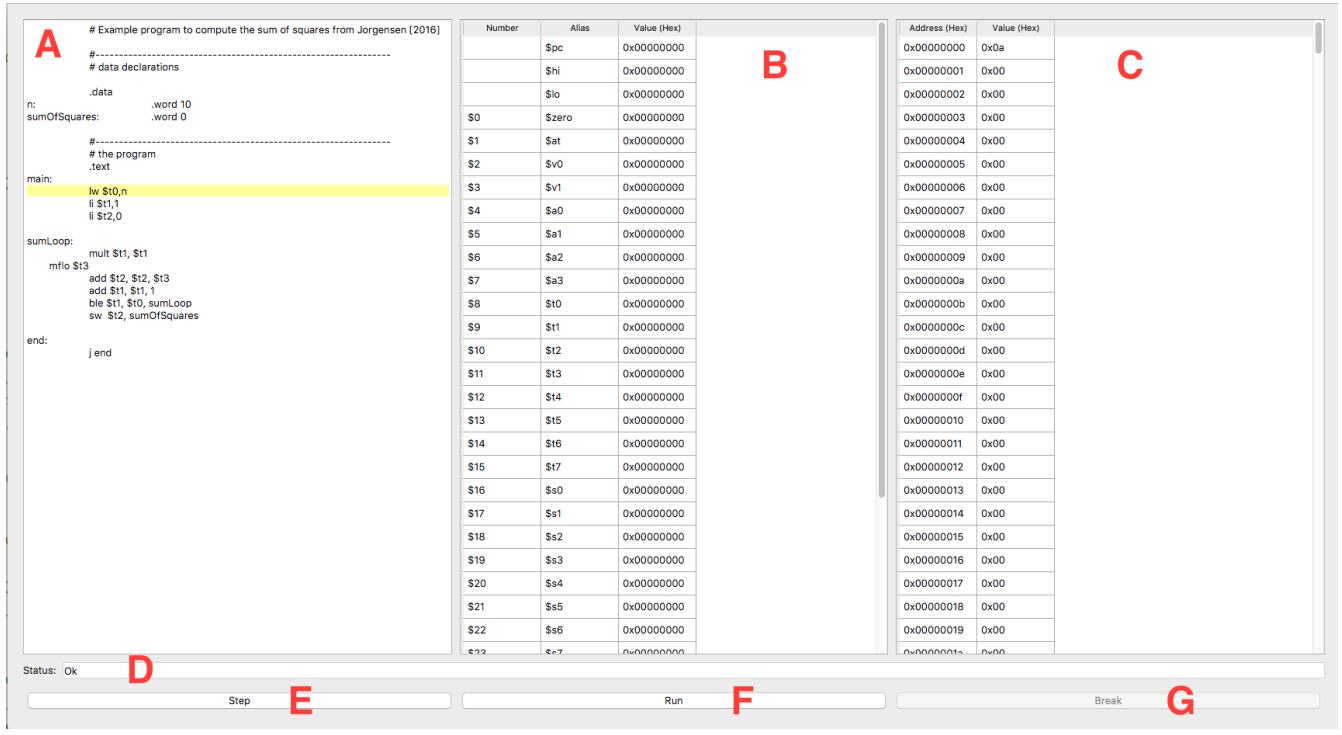
Project milestones

Milestone	Duration	Points
Milestone 0	3 weeks	20
Milestone 1	3 weeks	48
Milestone 2	4 weeks	70
Milestone 3	2 weeks	92
Milestone 4	2.5 weeks	100

Milestone 4

- **Add `run` and `break` to `simmips`**
  - Due: 4/26 by 11:59 PM-
  - Specification
- New simmips commands
  - `run` : start execution of the current program (repeated step)
  - `break` : stop execution of the current program (stop at end of current step). Does nothing if the simualtion is not running.

SIMMIPS GUI


A
Example program to compute the sum of squares from Jorgensen [2016]
#--
data declarations
.data
n: .word 10
sumOfSquares: .word 0
#--
the program
.text
main:
lw $t0,n
li $t1,1
li $t2,0
sumLoop:
mult $t1, $t1
mflo $t3
add $t2, $t2, $t3
add $t1, $t1, 1
ble $t1, $t0, sumLoop
sw $t2, sumOfSquares
end:
j end
Number
Alias
Value (Hex)
$pc 0x00000000
$hi 0x00000000
$lo 0x00000000
$0 $zero 0x00000000
$1 $at 0x00000000
$2 $v0 0x00000000
$3 $v1 0x00000000
$4 $a0 0x00000000
$5 $a1 0x00000000
$6 $a2 0x00000000
$7 $a3 0x00000000
$8 $t0 0x00000000
$9 $t1 0x00000000
$10 $t2 0x00000000
$11 $t3 0x00000000
$12 $t4 0x00000000
$13 $t5 0x00000000
$14 $t6 0x00000000
$15 $t7 0x00000000
$16 $s0 0x00000000
$17 $s1 0x00000000
$18 $s2 0x00000000
$19 $s3 0x00000000
$20 $s4 0x00000000
$21 $s5 0x00000000
$22 $s6 0x00000000
B
Address (Hex)
Value (Hex)
0x00000000 0x0a
0x00000001 0x00
0x00000002 0x00
0x00000003 0x00
0x00000004 0x00
0x00000005 0x00
0x00000006 0x00
0x00000007 0x00
0x00000008 0x00
0x00000009 0x00
0x0000000a 0x00
0x0000000b 0x00
0x0000000c 0x00
0x0000000d 0x00
0x0000000e 0x00
0x0000000f 0x00
0x00000010 0x00
0x00000011 0x00
0x00000012 0x00
0x00000013 0x00
0x00000014 0x00
0x00000015 0x00
0x00000016 0x00
0x00000017 0x00
0x00000018 0x00
0x00000019 0x00
C
Status: Ok
D
Step
E
Run
F
Break
G

Re-submitting one past milestone

- You are allowed to re-submit one of past milestones for re-grading.
    - You will get full credit.
- Will notify you details later.

Thread Safe Queue

- Today we are going to look in detail at how to make a data structure thread-safe.
  - Review of `std::queue`
  - `push`
  - `empty`
  - `try_pop`
  - `wait_and_pop`
  - Message Queues
  - Exercise

Review `std::queue`

- A first-in-first-out queue with (basic) methods
  - `push`
  - `pop`
  - `empty`

Like all standard containers, `std::queue` is not thread-safe

- *Q: How can we adapt the queue to protect access?*
- *A: mutexes and condition variables*
- We protect each method with a mutex.

The interface

```
template<typename T>
class ThreadSafeQueue
{
public:
    void push(const T & value);
    bool empty() const;
    bool try_pop(T& popped_value);
    void wait_and_pop(T& popped_value);
private:
    std::queue<T> the_queue;
    mutable std::mutex the_mutex;
    std::condition_variable the_condition_variable;
};
```

- C++ `mutable` keyword

Simplest case: `empty` member function

```
template<typename T>
bool ThreadSafeQueue<T>::empty() const {
    std::lock_guard<std::mutex> lock(the_mutex);
    return the_queue.empty();
}
```

- std::lock_guard
- `const` member functions in C++

`push` member function

```
template<typename T>
void ThreadSafeQueue<T>::push(const T& value) {
    std::unique_lock<std::mutex> lock(the_mutex);
    the_queue.push(value);
    lock.unlock();
    the_condition_variable.notify_one();
}
```

- std::unique_lock
- std::condition_variable::notify_one

`try_pop` member function

- No waiting, returns true on success, popped value as an output argument.

```
template<typename T>
bool ThreadSafeQueue<T>::try_pop(T &popped_value) {
    std::lock_guard<std::mutex> lock(the_mutex);
    if (the_queue.empty()) {
        return false;
    }

    popped_value = the_queue.front();
    the_queue.pop();
    return true;
}
```

`wait_and_pop` member function

- Wait for available, returns popped value as an output argument.

```
template<typename T>
void ThreadSafeQueue<T>::wait_and_pop(T &popped_value) {
    std::unique_lock<std::mutex> lock(the_mutex);
    while (the_queue.empty()) {
        the_condition_variable.wait(lock);
    }
    popped_value = the_queue.front();
    the_queue.pop();
}
```

- std::condition_variable::wait

Thread-safe queues

- Thread-safe queues are a good way to implement message passing between threads, where they are called Message Queues.
  - Each thread has a pointer or reference to a shared input ThreadSafeQueue holding units of work
  - Each thread has a pointer or reference to a shared output ThreadSafeQueue holding results of work
  - Each thread calls `wait_and_pop` on input queue, does the work, then calls `push` on the output queue

Thread-safe queues

- Often a single thread, the **Producer**, pushes into the input queue and pops from the output queue.
- The other threads act as Workers or **Consumers**.

Exercise

- See the website.

Next Actions and Reminders

- Read about Producer/Consumer Pattern