6.823 Spring 2014

Handout #14 – Multi-producer/Single-consumer Messaging Queues

http://csg.csail.mit.edu/6.823/

You are writing a queue to be used in a multi-producer/single-consumer application. (Producer threads write messages that are read by one consumer.) We assume here a queue with infinite space.

`TST rs, Imm(rt)` is the test-and-set instruction, which *atomically* loads the value at `Imm(rt)` into `rs`, and if the value is zero, updates the memory location at `Imm(rt)` to 1. This atomic instruction is useful for implementing locks: a value of 1 at the memory location indicates that someone holds the lock, and a value of 0 means the lock is free.

Producer pushes a message onto queue: (memory operations in bold)

```
void push(int** tail_ptr, int* tail_write_lock, int message) {
      while (lock_try(tail_write_lock) == false);
      **tail_ptr = message;
      *tail_ptr++;
      lock_release(tail_write_lock);
}
```

```
# R1 - contains address of data to enqueue
# R2 - contains the address of the tail pointer of queue
# R3 - address of tail pointer write lock
P1 SpinLock:TST R4, 0(R3)          # try to acquire tail write lock
P2          BNEZ R4, R4, SpinLock
P3          LD R4, 0(R2)           # get tail pointer
P4          ST R1, 0(R4)           # write message to tail
P5          ADD R4, R4, 4          # update tail pointer
P6          ST R4, 0(R2)
P7          ST R0, 0(R3)           # release lock
```

Consumer pops a message off queue: (memory operations in bold)

```
int pop(int** head_ptr, int** tail_ptr) {
      while (*head_ptr == *tail_ptr);
      int message = **head_ptr;
      *head_ptr++;
      return message;
}

# R1 - will receive address contained in message
# R2 - contains the address of the head pointer of queue
# R3 - contains the address of the tail pointer of the queue
C1 Retry:   LD R4, 0(R2)            # get head pointer
C2          LD R5, 0(R3)            # get tail pointer
C3          SUB R5, R4, R5          # is there a message?
C4          BNEZ R5, Pop
C5          JMP Retry
C6 Pop:     LD R1, 0(R4)            # read message from queue
C7          ADD R4, R4, 4           # update head pointer
C8          ST R4, 0(R2)
```