Machine Problem 5: Kernel-Level Thread Scheduling

Nachiket Umesh Naganure
UIN: 532008698
CSCE611: Operating System

Assigned Tasks

Main Task Completed.
Bonus Option 1: Completed.
Bonus Option 2: Did not attempt
Bonus Option 3: Did not attempt.

System Design

The goal of the machine problem was to design a FIFO scheduler for threads to implement scheduling of multiple kernel-level threads.

1. Implement a scheduler class which handles thread.
2. Implement a queue to handle the threads.

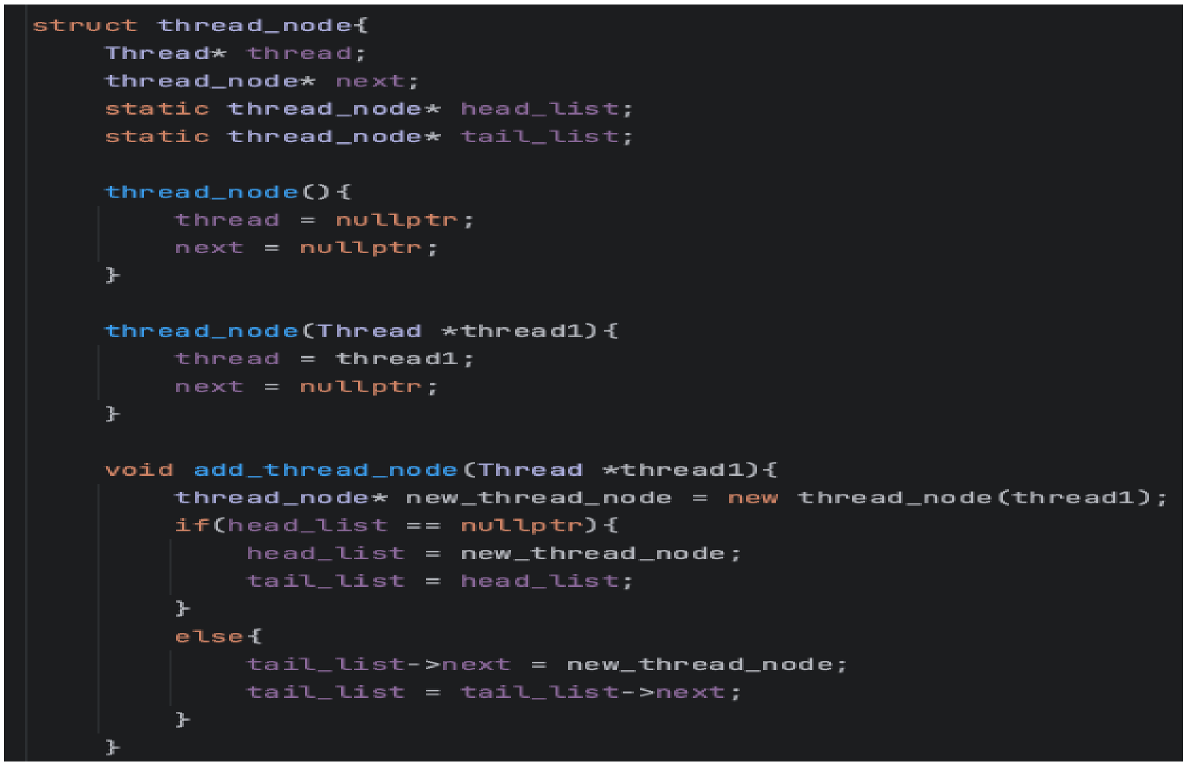
Code Description

I mainly made changes in "schedular.c", "schedular.h" and "thread.c". To compile the code, you need to run the make file. To run the simulator, use copykernel.sh script and run the command "bochs -f bochsrc.bxrc" to start the bochs simulator.

Part I: FIFO Implement a queue using struct storing a thread pointers.

Part II: Handling of interrupts Using Machine::enable_Interrupts() and Machine::disable_interrupts() we handle interrupts for yeilding and adding threads.

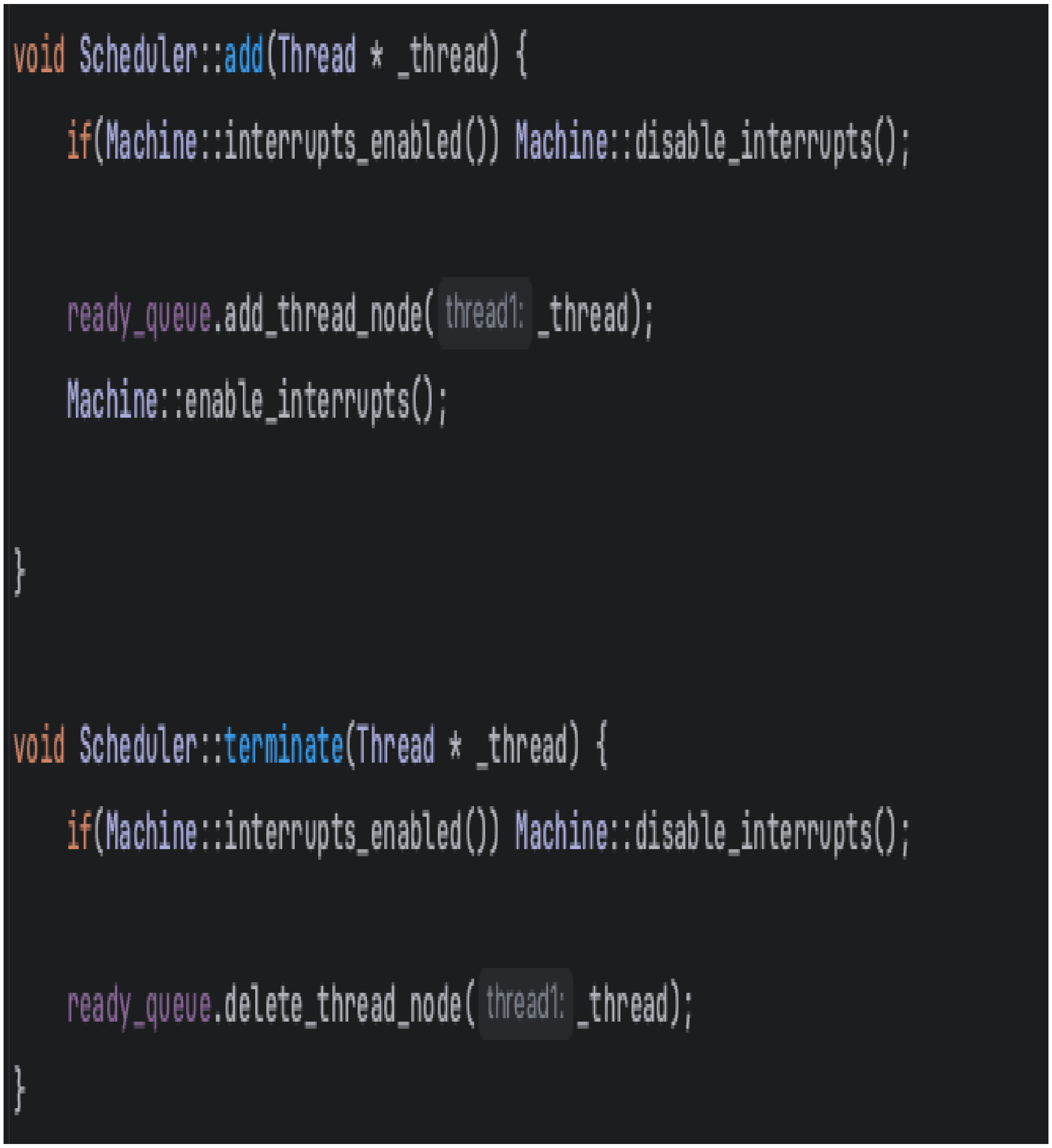
struct_add_thread :



```
struct thread_node{
    Thread* thread;
    thread_node* next;
    static thread_node* head_list;
    static thread_node* tail_list;

    thread_node(){
        thread = nullptr;
        next = nullptr;
    }

    thread_node(Thread *thread1){
        thread = thread1;
        next = nullptr;
    }

    void add_thread_node(Thread *thread1){
        thread_node* new_thread_node = new thread_node(thread1);
        if(head_list == nullptr){
            head_list = new_thread_node;
            tail_list = head_list;
        }
        else{
            tail_list->next = new_thread_node;
            tail_list = tail_list->next;
        }
    }
```

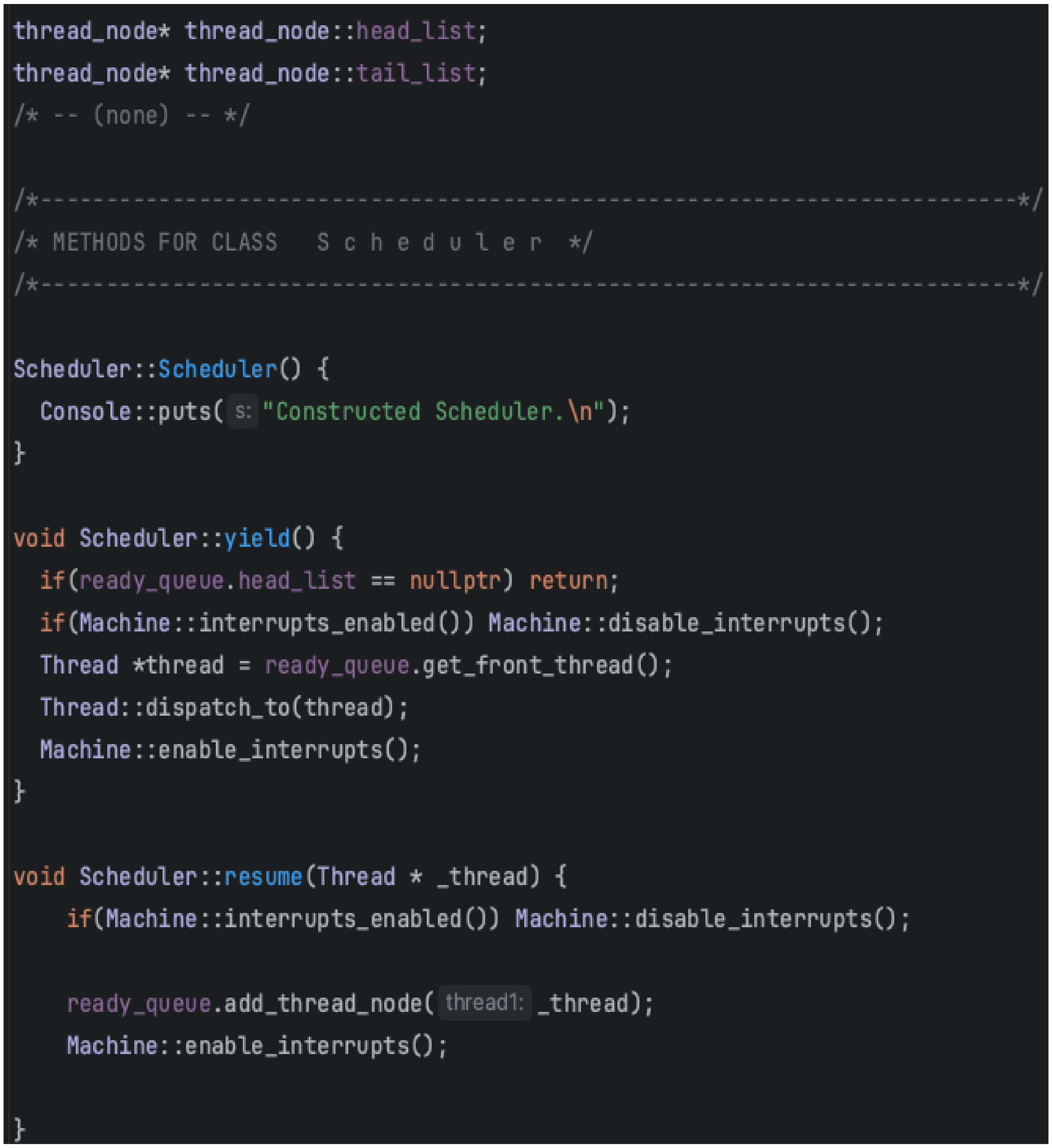
Figure 1: Structure of thread Node

Adding Threads to Queue :



```
void Scheduler::add(Thread * _thread) {
    if(Machine::interrupts_enabled()) Machine::disable_interrupts();


    ready_queue.add_thread_node( thread1: _thread);
    Machine::enable_interrupts();


}


void Scheduler::terminate(Thread * _thread) {
    if(Machine::interrupts_enabled()) Machine::disable_interrupts();


    ready_queue.delete_thread_node( thread1: _thread);
}
```

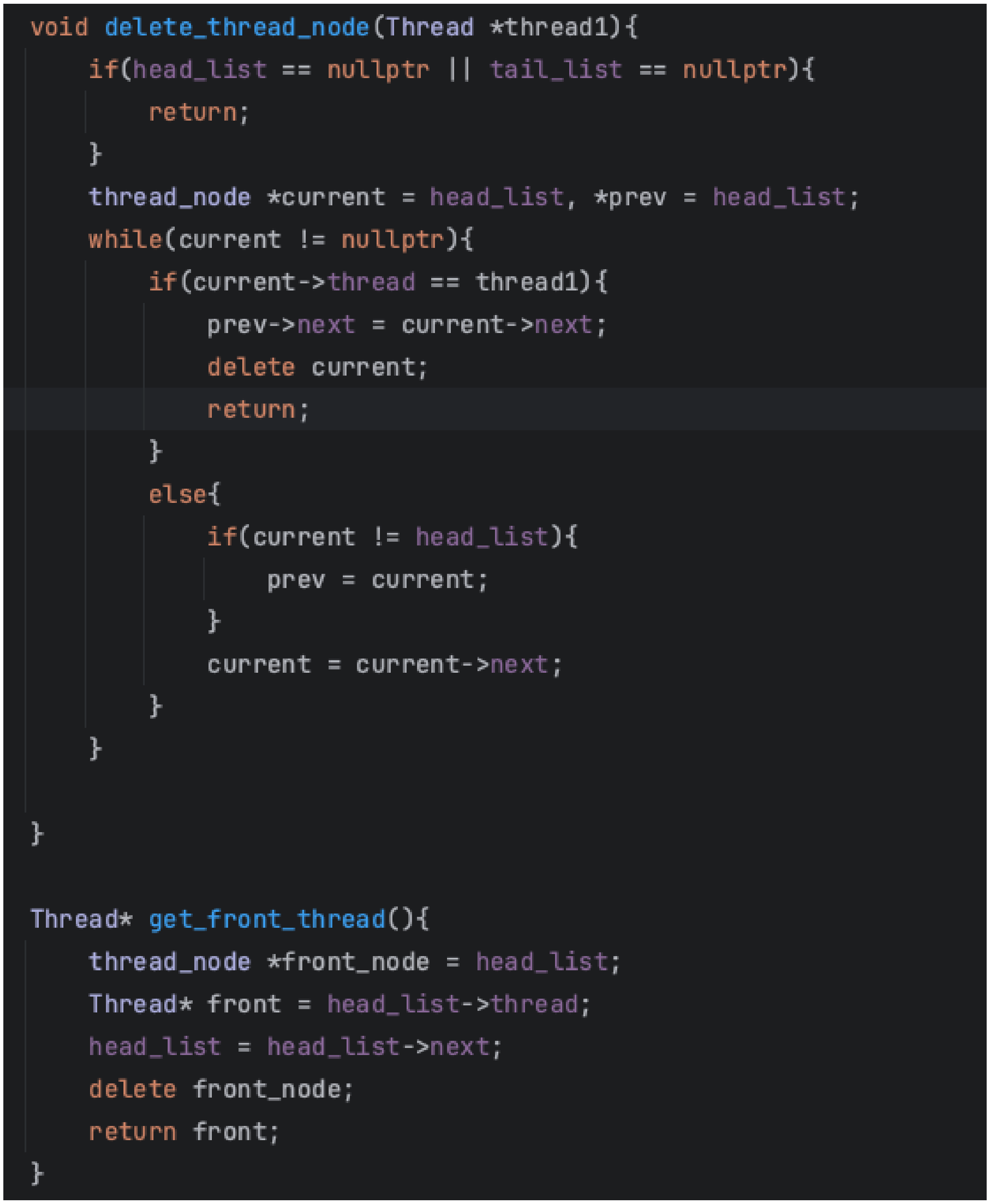
Figure 2: Adding Threads to Queue



```
thread_node* thread_node::head_list;
thread_node* thread_node::tail_list;
/* -- (none) -- */

/*--------------------------------------------------------------------------*/
/* METHODS FOR CLASS   S c h e d u l e r  */
/*--------------------------------------------------------------------------*/

Scheduler::Scheduler() {
  Console::puts( s: "Constructed Scheduler.\n");
}

void Scheduler::yield() {
  if(ready_queue.head_list == nullptr) return;
  if(Machine::interrupts_enabled()) Machine::disable_interrupts();
  Thread *thread = ready_queue.get_front_thread();
  Thread::dispatch_to(thread);
  Machine::enable_interrupts();
}

void Scheduler::resume(Thread * _thread) {
    if(Machine::interrupts_enabled()) Machine::disable_interrupts();

    ready_queue.add_thread_node( thread1: _thread);
    Machine::enable_interrupts();

}
```

Figure 3: Yield and resumes threads in scheduler

Delete and get front thread from queue :

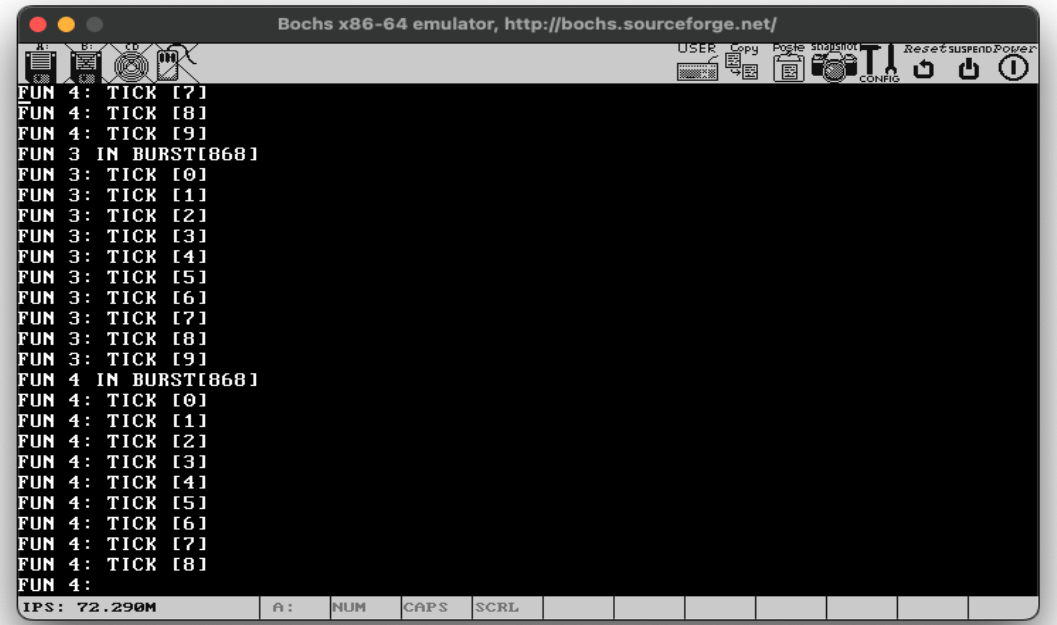


```cpp
void delete_thread_node(Thread *thread1){
    if(head_list == nullptr || tail_list == nullptr){
        return;
    }
    thread_node *current = head_list, *prev = head_list;
    while(current != nullptr){
        if(current->thread == thread1){
            prev->next = current->next;
            delete current;
            return;
        }
        else{
            if(current != head_list){
                prev = current;
            }
            current = current->next;
        }
    }

}

Thread* get_front_thread(){
    thread_node *front_node = head_list;
    Thread* front = head_list->thread;
    head_list = head_list->next;
    delete front_node;
    return front;
}
```

Figure 4: Delete and get front thread from queue

Testing

For testing, I used the given test function from kernel.C. I ran two scenarios where the _USES_SCHEDULER_ and _TERMINATING_FUNCTIONS_ macro was set and unset. All the tests are passing, for given and additional scenarios.



```
FUN 4: TICK [7]
FUN 4: TICK [8]
FUN 4: TICK [9]
FUN 3 IN BURST[868]
FUN 3: TICK [0]
FUN 3: TICK [1]
FUN 3: TICK [2]
FUN 3: TICK [3]
FUN 3: TICK [4]
FUN 3: TICK [5]
FUN 3: TICK [6]
FUN 3: TICK [7]
FUN 3: TICK [8]
FUN 3: TICK [9]
FUN 4 IN BURST[868]
FUN 4: TICK [0]
FUN 4: TICK [1]
FUN 4: TICK [2]
FUN 4: TICK [3]
FUN 4: TICK [4]
FUN 4: TICK [5]
FUN 4: TICK [6]
FUN 4: TICK [7]
FUN 4: TICK [8]
FUN 4:
```

Figure 5: Testing