Isaac Lino 860869360
Steven Lopez 861106782

# Lab2 Report

**ISSUES/PROGRAM**

To create a implementation on thread_yield we had to create a system call. We have some compiling issues since no changes were made in **defs.h** after calling the system call under proc.c. defs.h is a header file that calls all the system calls.

We had some problems on Makefile when we were implementing part 2.1 Null pointer dereference. The page validation was made in the first page, clearly the entry point will have to be on Makefile and change kernel memory to 0x1000. After this implementation, second part worked.

**END ISSUES/PROGRAM**

## Files added

**semaphore.h**
**H2O_test.c**
**Missionary_test.c**
**Monkey_test.c (didn't complete this)**
**null_ptr.c**
**thread_yield_test.c**
**To test any program simply make qemu-nox and type either H2O_test, Missionary_test, null_ptr, thread_yield_test.**

### 1.1 include your data structure, and design of semaphore.

**All inside semaphore.h**

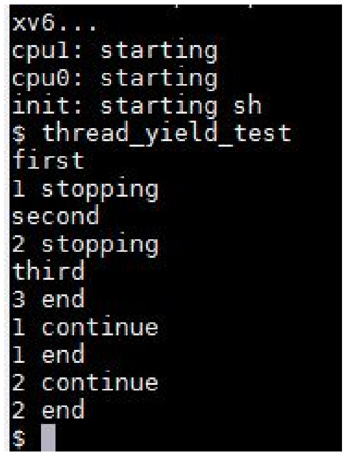
```
struct Semaphore
{
        struct queue sem_q;
        int count;
        lock_t sem_lock;
};
void sem_start(struct Semaphore *s, int start_value){
        s->count = start_value;
        lock_init(&s->sem_lock);
        init_q(&s->sem_q);
}
```

## 1.2 explain how to implement thread_yield using one or two sentences. How to run your test for yield. and screen shots of output in your test.

**We created a new system call, very similar to yield. Just an extra check for "isthread".**

**TEST: simple make qemu-nox**

**Then type in thread_yield_test**



```
xv6...
cpu1: starting
cpu0: starting
init: starting sh
$ thread_yield_test
first
1 stopping
second
2 stopping
third
3 end
1 continue
1 end
2 continue
2 end
$ 
```

**Explaing the picture:**

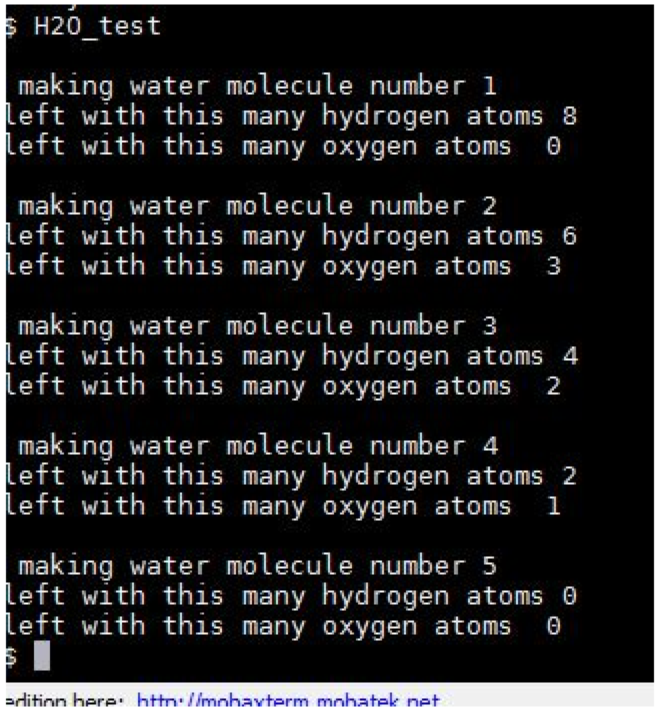
**We start our first, we call thread_yield() during it. It then moves onto the next(second), we call thread_yield() on this one as well. Move onto our third one which just completes(no yield on this one). It finishes up the first one then finishes up the second.**

## 1.3

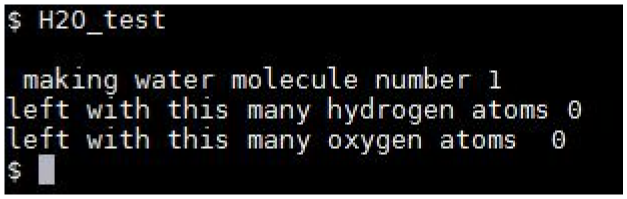
### A.

**This one was straight forward, the pseudo code was given to us so we just had to call acquire and signal where needed. My number of hydrogen/oxygen atoms are off since they don't have a lock around them.**

**The water molecules are made correctly.**



```
$ H2O_test

 making water molecule number 1
left with this many hydrogen atoms 8
left with this many oxygen atoms  0

 making water molecule number 2
left with this many hydrogen atoms 6
left with this many oxygen atoms  3

 making water molecule number 3
left with this many hydrogen atoms 4
left with this many oxygen atoms  2

 making water molecule number 4
left with this many hydrogen atoms 2
left with this many oxygen atoms  1

 making water molecule number 5
left with this many hydrogen atoms 0
left with this many oxygen atoms  0
$ 
edition here: http://mobaxterm.mobatek.net
```



```
$ H2O_test

 making water molecule number 1
left with this many hydrogen atoms 0
left with this many oxygen atoms  0
$ 
```

A.

**B.**
**didn’t implement this one**
**** We have test cases for monkey on the file attached to lab 2 **,**

A monkey community lives on an island that has only one coconut tree. The monkeys take turn going up the tree (which can hold only 3 monkeys at a time). After each monkey grabs a coconut, it climbs down the tree to go and eat it.

(a) Simulate (shop pseudocode) this synchronization problem; each monkey is an independent thread.
(b) Modify your solution if necessary to ensure that monkeys can be climbing up, or down the tree, but not both ways at the same time.
(c) The dominant monkey does not like to wait. When it arrives, it climbs up the tree ahead of any monkeys that got there before it (there still can only be three monkeys up the tree). Implement the synchronization for this problem (show the procedure for the dominant monkey thread and extend the procedure for the regular monkeys). Is starvation possible?

**PSEUDO CODE**
**Monkey colony (dominant monkey)**

**** We have test cases for monkey on the file attached to lab 2 ****

**part(a)**
```
 int monkey_count[3] = {0,0};       /* monkey counter for each direction */ up and down
semaphore mutex[3] = {1,1};

semaphore max_on_tree = 3; // these are the max of monkeys on tree
semaphore tree = 1;

/* mutual exclusion for each direction*/
/* ensure maximum 3 monkey on the tree*/
/* ensure monkey on the tree is heading to one direction up/down */
```

**part(b)**
```
 void WaitUntilOneMonkeyDown(up_downination up_down)
{
        wait(mutex[up_down]);
        monkey_count[up_down]++;
        if (monkey_count[up_down] == 1)

           //is the first monkey in line, waiting to go up to three

           wait(tree);
        signal(mutex[up_down]);
        wait(max_on_tree);

}
```

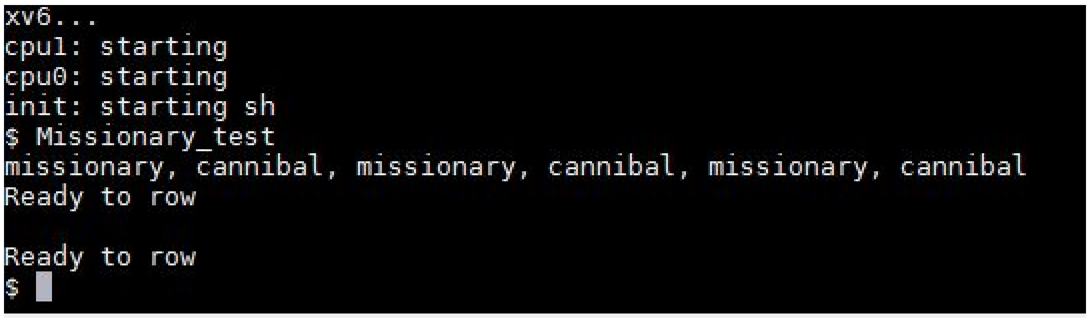
```
 void DoneWithClimbing(up_downination up_down)
{

       wait(mutex[up_down]);
        signal(max_on_tree);
        monkey_count[up_down]--;

             // three can hold 3 monkeys but it will allow a monkey to go up if 0,1 or 2 are on three
        if (monkey_count[up_down] == 0 || monkey_count[up_down] == 1 || monkey_count[up_down] == 2 )

           //is the last monkey, release the tree
         signal(tree);
        signal(mutex[up_down]);

}
```

**Part (c) implementation**

```
void AlphaMonkey(* arg_ptr)        //Alpha = dominant
{
             alpha monkey arrives
             semaphore(&_mutex)
             //check to at least 2 monkeys are in a tree.. No more than 3 monkeys
             DoneWithClimbing();

      //is the last monkey, release the tree
         signal(tree);
        signal(mutex[up_down]);

             lock.acquire(mutex[up_down]);

             monkey is up and grabbing a coconut
             semaphore(&_mutex)
}
```

**END PSEUDO CODE**

C.



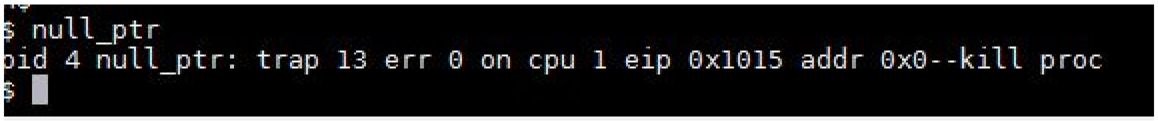
```
xv6...
cpu1: starting
cpu0: starting
init: starting sh
$ Missionary_test
missionary, cannibal, missionary, cannibal, missionary, cannibal
Ready to row

Ready to row
$ 
```

2.1 what parts you changed. how to run your test and screen shots of outputs.

We created a program called null_ptr.c to dereference.

Edited the makefile to change address to 0x1000 from 0.

Fork calls vm.c which was edited, checks in exec



```
$ null_ptr
pid 4 null_ptr: trap 13 err 0 on cpu 1 eip 0x1015 addr 0x0--kill proc
$ 
```

2.2 Bonus

Didn’t do this part