zihang zhou 861090400
kenneth chan 861092781

Lab 2 Report

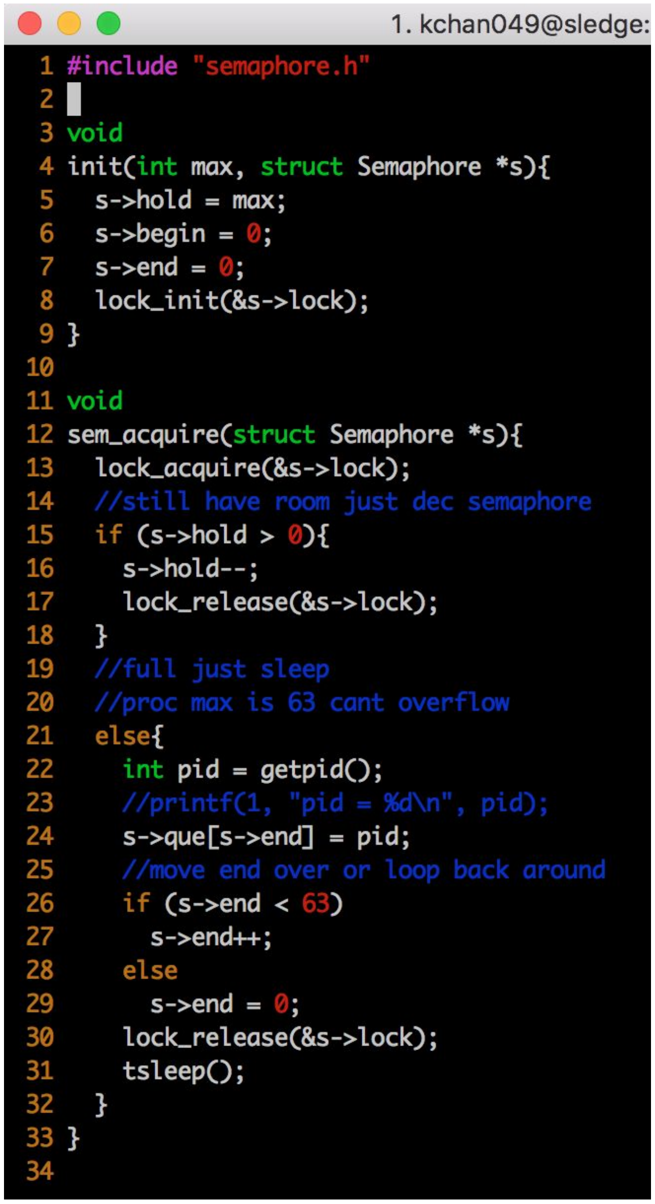
part1.1: include your data structure, and design of semaphore.

uses a queue, a lock, and an integer for the data structure of the semaphore.

init(int, semaphore*) sets the integer of the semaphore

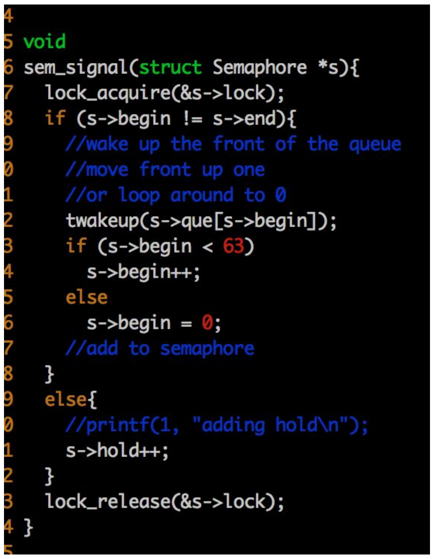
sem_acquire(semaphore*) decreases the int of the semaphore or sleeps the thread if the int is 0

sem_signal(semaphore*) increases the int of the semaphore and wakes threads.



1. kchan049@sledge:

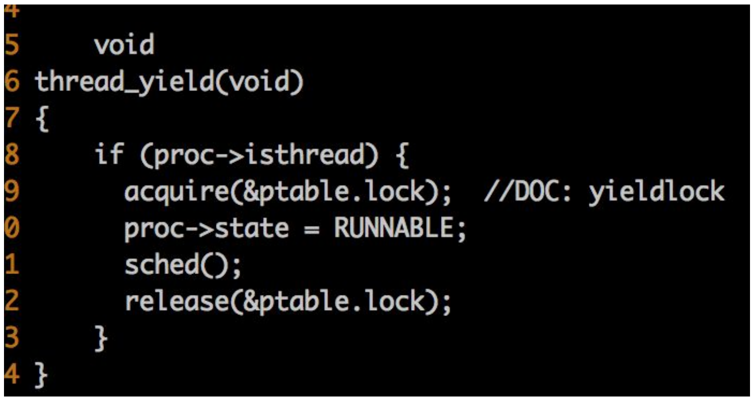
```
#include "semaphore.h"

void
init(int max, struct Semaphore *s){
  s->hold = max;
  s->begin = 0;
  s->end = 0;
  lock_init(&s->lock);
}

void
sem_acquire(struct Semaphore *s){
  lock_acquire(&s->lock);
  //still have room just dec semaphore
  if (s->hold > 0){
    s->hold--;
    lock_release(&s->lock);
  }
  //full just sleep
  //proc max is 63 cant overflow
  else{
    int pid = getpid();
    //printf(1, "pid = %d\n", pid);
    s->que[s->end] = pid;
    //move end over or loop back around
    if (s->end < 63)
      s->end++;
    else
      s->end = 0;
    lock_release(&s->lock);
    tsleep();
  }
}

```



```
void
sem_signal(struct Semaphore *s){
  lock_acquire(&s->lock);
  if (s->begin != s->end){
    //wake up the front of the queue
    //move front up one
    //or loop around to 0
    twakeup(s->que[s->begin]);
    if (s->begin < 63)
      s->begin++;
    else
      s->begin = 0;
    //add to semaphore
  }
  else{
    //printf(1, "adding hold\n");
    s->hold++;
  }
  lock_release(&s->lock);
}
```

part1.2: 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.

you copy the yield system call. You then check the process to make sure it is a thread with a if statement.



```
    void
thread_yield(void)
{
    if (proc->isthread) {
      acquire(&ptable.lock);  //DOC: yieldlock
      proc->state = RUNNABLE;
      sched();
      release(&ptable.lock);
    }
}
```

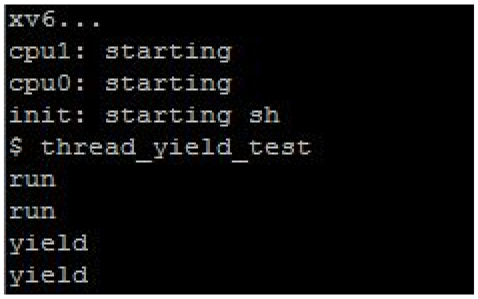
test:

compile and run xv6

run thread_yield_test

the threads with thread_yield prints yield other threads print run

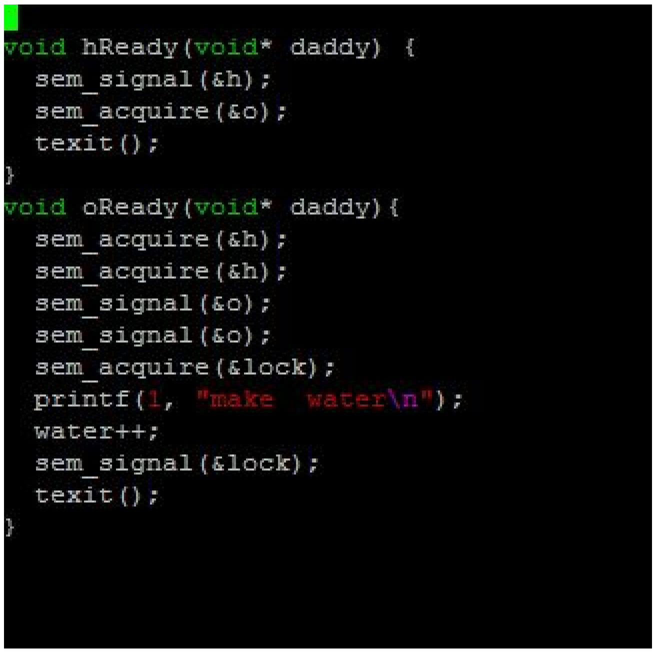
the yielded threads should be at the bottom of output list



```
xv6...
cpu1: starting
cpu0: starting
init: starting sh
$ thread_yield_test
run
run
yield
yield
```

part1.3: pseudo code for algorithms. how to run your test cases and screen shots of outputs.

H2O:

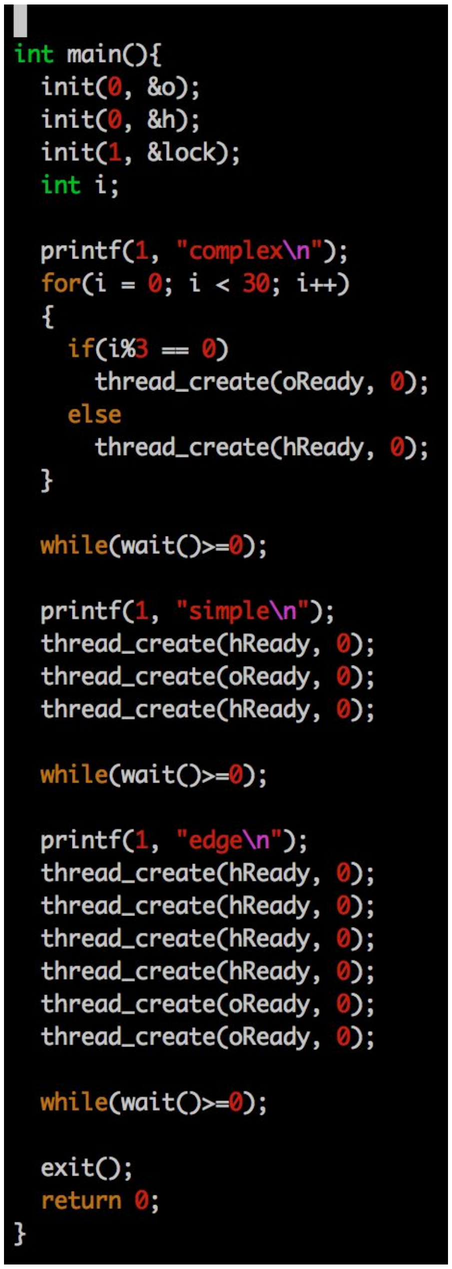


```
void hReady(void* daddy) {
  sem_signal(&h);
  sem_acquire(&o);
  texit();
}
void oReady(void* daddy){
  sem_acquire(&h);
  sem_acquire(&h);
  sem_signal(&o);
  sem_signal(&o);
  sem_acquire(&lock);
  printf(1, "make  water\n");
  water++;
  sem_signal(&lock);
  texit();
}
```

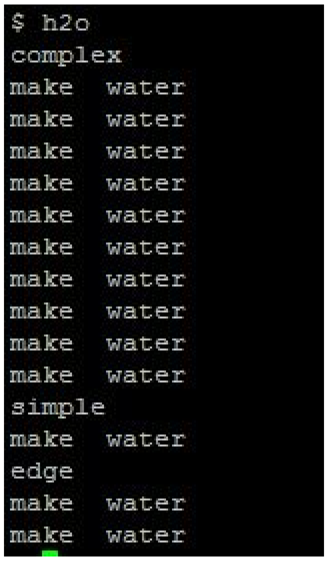
h calls o

o waits for 2 h and inc water and signals the 2 h

Testing:



```
int main(){
  init(0, &o);
  init(0, &h);
  init(1, &lock);
  int i;

  printf(1, "complex\n");
  for(i = 0; i < 30; i++)
  {
    if(i%3 == 0)
      thread_create(oReady, 0);
    else
      thread_create(hReady, 0);
  }

  while(wait()>=0);

  printf(1, "simple\n");
  thread_create(hReady, 0);
  thread_create(oReady, 0);
  thread_create(hReady, 0);

  while(wait()>=0);

  printf(1, "edge\n");
  thread_create(hReady, 0);
  thread_create(hReady, 0);
  thread_create(hReady, 0);
  thread_create(hReady, 0);
  thread_create(oReady, 0);
  thread_create(oReady, 0);

  while(wait()>=0);

  exit();
  return 0;
}
```



```
$ h2o
complex
make  water
make  water
make  water
make  water
make  water
make  water
make  water
make  water
make  water
make  water
simple
make  water
edge
make  water
make  water
```

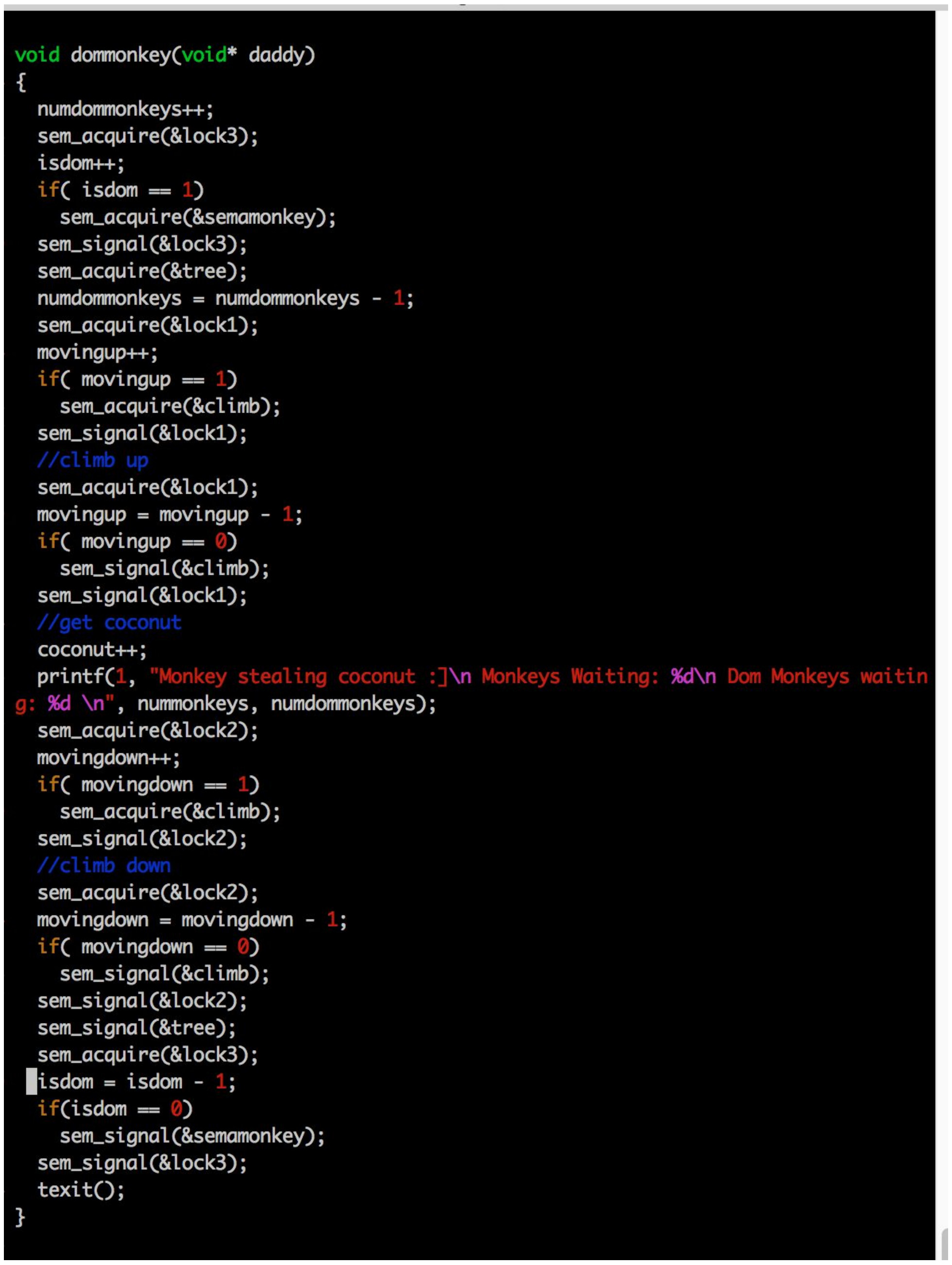
complex makes multiple waters

simple makes 1 water

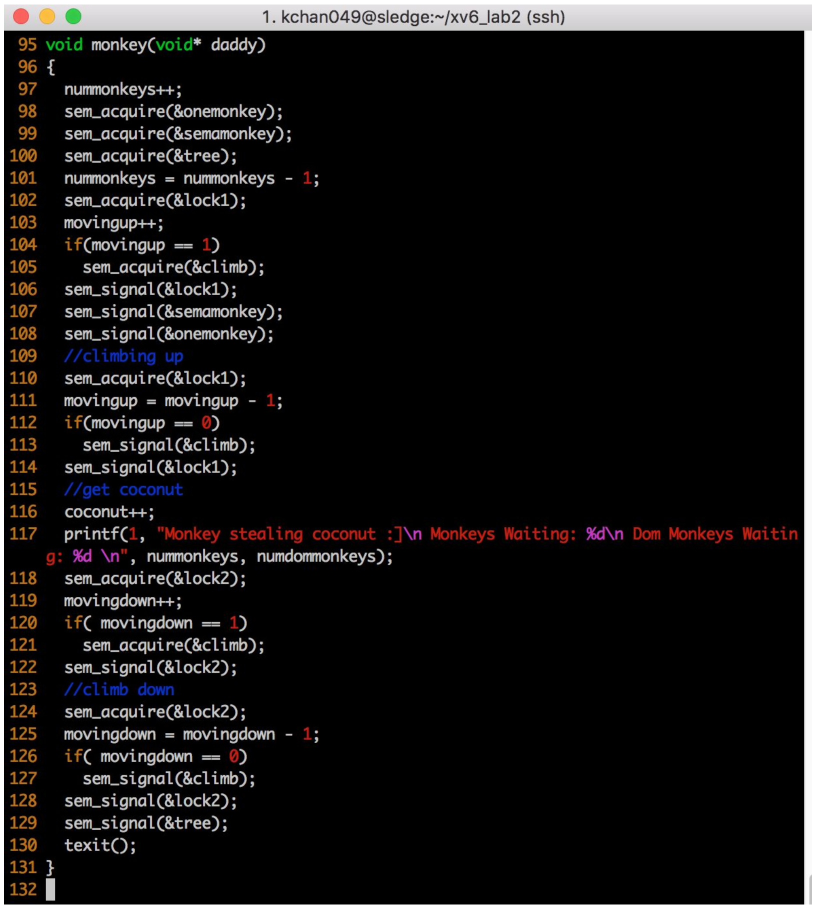
edge makes water by sending in 4 h and then 2 o

Monkey:

Pseudo Code:



```
void dommonkey(void* daddy)
{
  numdommonkeys++;
  sem_acquire(&lock3);
  isdom++;
  if( isdom == 1)
    sem_acquire(&semamonkey);
  sem_signal(&lock3);
  sem_acquire(&tree);
  numdommonkeys = numdommonkeys - 1;
  sem_acquire(&lock1);
  movingup++;
  if( movingup == 1)
    sem_acquire(&climb);
  sem_signal(&lock1);
  //climb up
  sem_acquire(&lock1);
  movingup = movingup - 1;
  if( movingup == 0)
    sem_signal(&climb);
  sem_signal(&lock1);
  //get coconut
  coconut++;
  printf(1, "Monkey stealing coconut :]\n Monkeys Waiting: %d\n Dom Monkeys waitin
g: %d \n", nummonkeys, numdommonkeys);
  sem_acquire(&lock2);
  movingdown++;
  if( movingdown == 1)
    sem_acquire(&climb);
  sem_signal(&lock2);
  //climb down
  sem_acquire(&lock2);
  movingdown = movingdown - 1;
  if( movingdown == 0)
    sem_signal(&climb);
  sem_signal(&lock2);
  sem_signal(&tree);
  sem_acquire(&lock3);
  isdom = isdom - 1;
  if(isdom == 0)
    sem_signal(&semamonkey);
  sem_signal(&lock3);
  texit();
}
```


1. kchan049@sledge:~/xv6_lab2 (ssh)

```
void monkey(void* daddy)
{
  nummonkeys++;
  sem_acquire(&onemonkey);
  sem_acquire(&semamonkey);
  sem_acquire(&tree);
  nummonkeys = nummonkeys - 1;
  sem_acquire(&lock1);
  movingup++;
  if(movingup == 1)
    sem_acquire(&climb);
  sem_signal(&lock1);
  sem_signal(&semamonkey);
  sem_signal(&onemonkey);
  //climbing up
  sem_acquire(&lock1);
  movingup = movingup - 1;
  if(movingup == 0)
    sem_signal(&climb);
  sem_signal(&lock1);
  //get coconut
  coconut++;
  printf(1, "Monkey stealing coconut :]\n Monkeys Waiting: %d\n Dom Monkeys Waitin
g: %d \n", nummonkeys, numdommonkeys);
  sem_acquire(&lock2);
  movingdown++;
  if( movingdown == 1)
    sem_acquire(&climb);
  sem_signal(&lock2);
  //climb down
  sem_acquire(&lock2);
  movingdown = movingdown - 1;
  if( movingdown == 0)
    sem_signal(&climb);
  sem_signal(&lock2);
  sem_signal(&tree);
  texit();
}
```

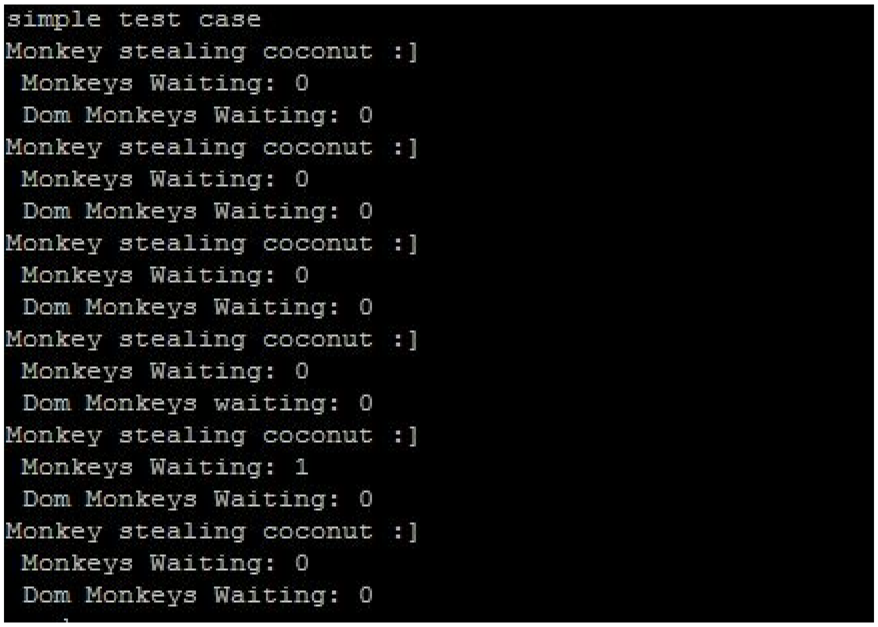
Test:

Instructions to run tests:

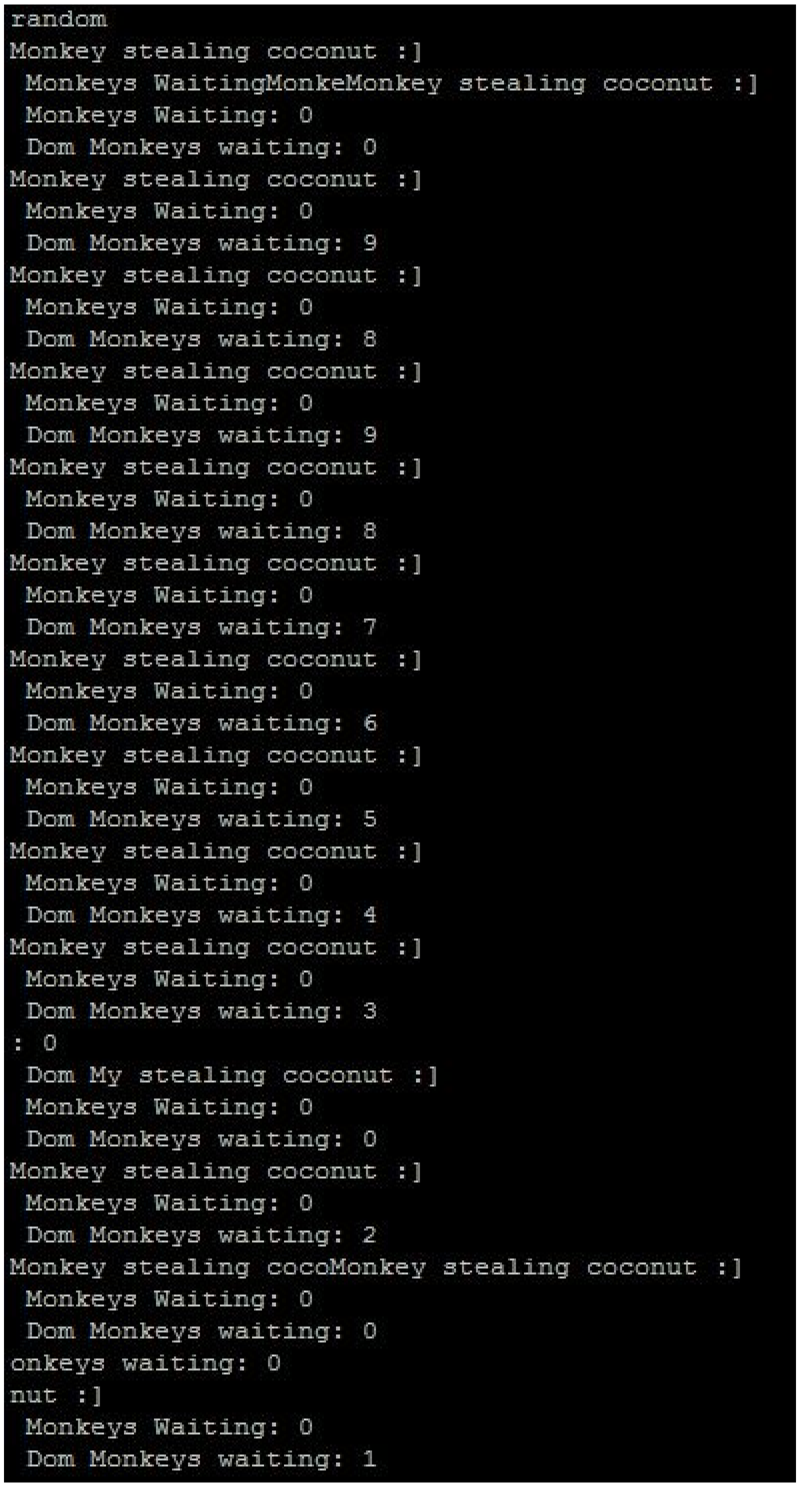
Type Make qemu-nox

Type monkey

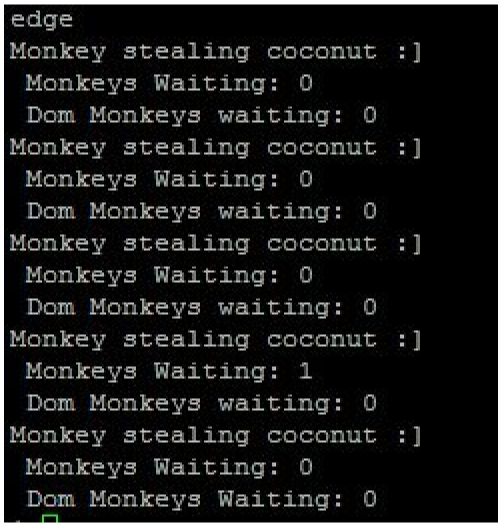
When we ran our monkey test cases, the timer interrupt would cut our text here and there, but the outputs with the corresponding numbers match as expected.



```
simple test case
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys Waiting: 0
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys Waiting: 0
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys Waiting: 0
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 0
Monkey stealing coconut :]
 Monkeys Waiting: 1
 Dom Monkeys Waiting: 0
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys Waiting: 0
```



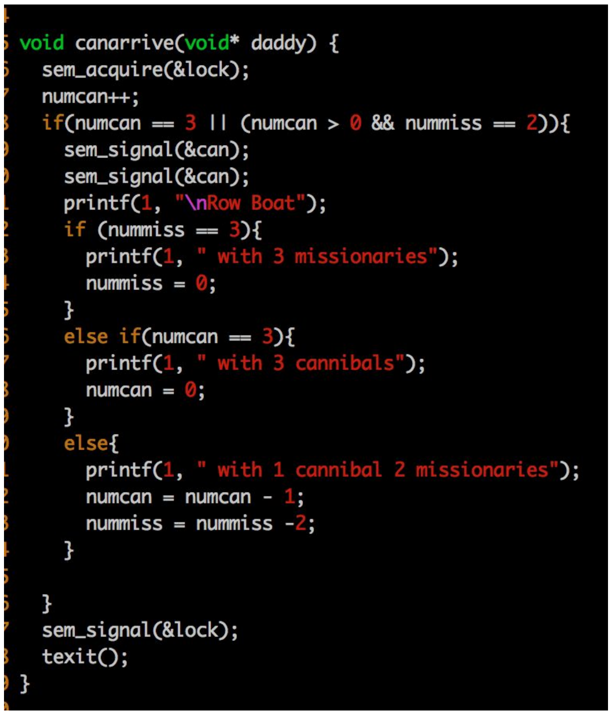
```
random
Monkey stealing coconut :]
 Monkeys WaitingMonkeMonkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 0
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 9
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 8
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 9
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 8
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 7
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 6
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 5
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 4
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 3
: 0
 Dom My stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 0
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 2
Monkey stealing cocoMonkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 0
onkeys waiting: 0
nut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 1
```



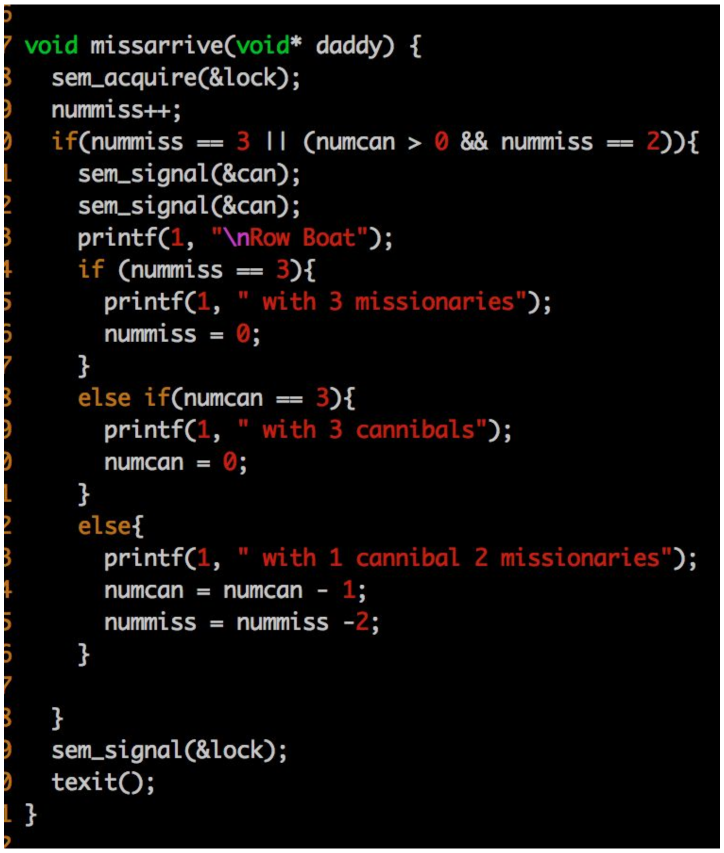
```
edge
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 0
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 0
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys waiting: 0
Monkey stealing coconut :]
 Monkeys Waiting: 1
 Dom Monkeys waiting: 0
Monkey stealing coconut :]
 Monkeys Waiting: 0
 Dom Monkeys Waiting: 0
```

River Problem:

This is our psuedo code:



```
void canarrive(void* daddy) {
  sem_acquire(&lock);
  numcan++;
  if(numcan == 3 || (numcan > 0 && nummiss == 2)){
    sem_signal(&can);
    sem_signal(&can);
    printf(1, "\nRow Boat");
    if (nummiss == 3){
      printf(1, " with 3 missionaries");
      nummiss = 0;
    }
    else if(numcan == 3){
      printf(1, " with 3 cannibals");
      numcan = 0;
    }
    else{
      printf(1, " with 1 cannibal 2 missionaries");
      numcan = numcan - 1;
      nummiss = nummiss -2;
    }

  }
  sem_signal(&lock);
  texit();
}
```



```
void missarrive(void* daddy) {
  sem_acquire(&lock);
  nummiss++;
  if(nummiss == 3 || (numcan > 0 && nummiss == 2)){
    sem_signal(&can);
    sem_signal(&can);
    printf(1, "\nRow Boat");
    if (nummiss == 3){
      printf(1, " with 3 missionaries");
      nummiss = 0;
    }
    else if(numcan == 3){
      printf(1, " with 3 cannibals");
      numcan = 0;
    }
    else{
      printf(1, " with 1 cannibal 2 missionaries");
      numcan = numcan - 1;
      nummiss = nummiss -2;
    }

  }
  sem_signal(&lock);
  texit();
}
```

Instructions for river test cases:

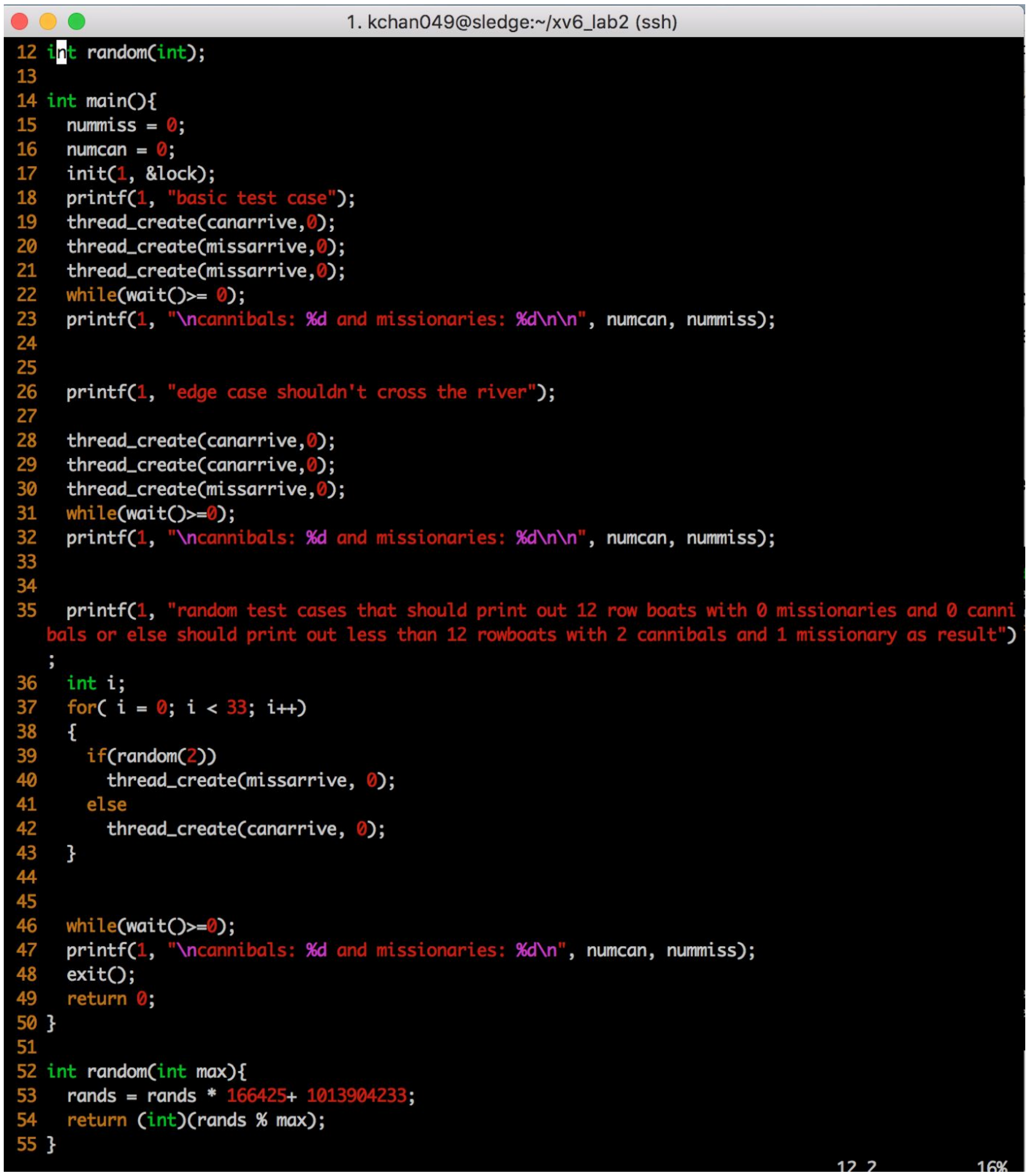
Type make qemu-nox.

Type river

For our test cases, we did a basic test case which was 2 cannibals and 1 missionary.

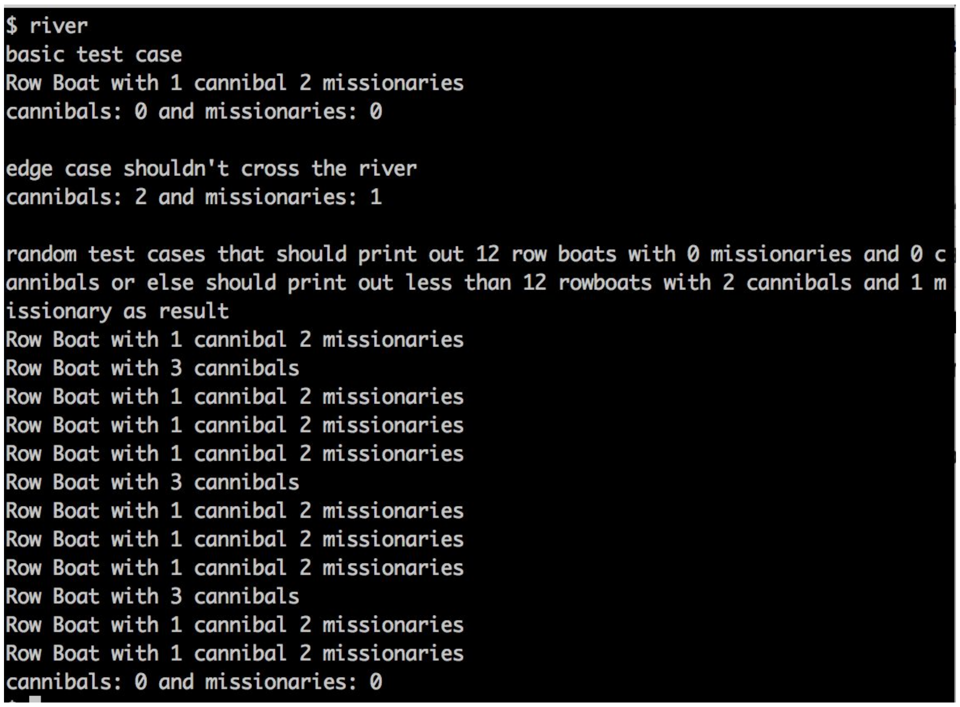
We did an edge case of 2 missionaries and 1 cannibal.

We did random tests that prints out 12 row boats with 0 cannibals and 0 missionaries remaining or it prints less than 12 row boats with a number of cannibals and missionaries remaining.


1. kchan049@sledge:~/xv6_lab2 (ssh)

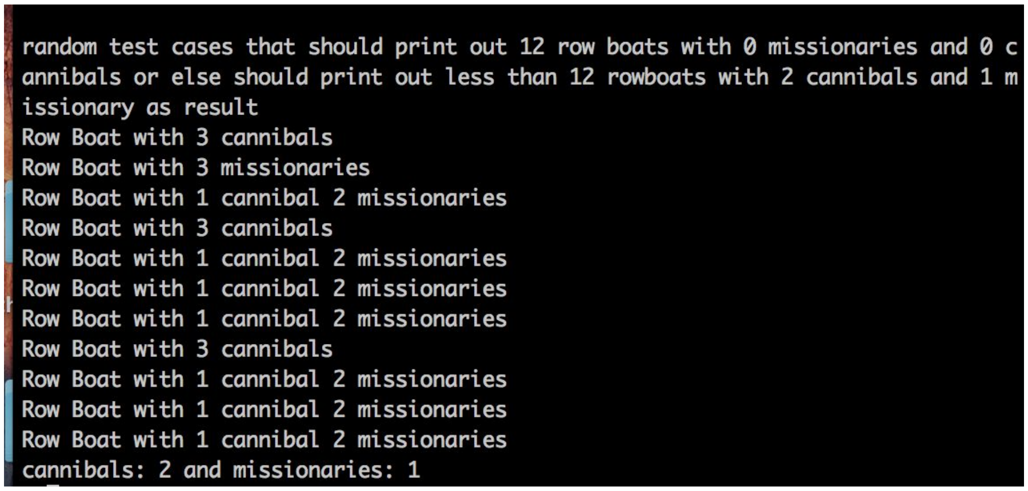
```
int random(int);

int main(){
  nummiss = 0;
  numcan = 0;
  init(1, &lock);
  printf(1, "basic test case");
  thread_create(canarrive,0);
  thread_create(missarrive,0);
  thread_create(missarrive,0);
  while(wait()>= 0);
  printf(1, "\ncannibals: %d and missionaries: %d\n\n", numcan, nummiss);


  printf(1, "edge case shouldn't cross the river");

  thread_create(canarrive,0);
  thread_create(canarrive,0);
  thread_create(missarrive,0);
  while(wait()>=0);
  printf(1, "\ncannibals: %d and missionaries: %d\n\n", numcan, nummiss);


  printf(1, "random test cases that should print out 12 row boats with 0 missionaries and 0 canni
bals or else should print out less than 12 rowboats with 2 cannibals and 1 missionary as result")
;
  int i;
  for( i = 0; i < 33; i++)
  {
    if(random(2))
      thread_create(missarrive, 0);
    else
      thread_create(canarrive, 0);
  }


  while(wait()>=0);
  printf(1, "\ncannibals: %d and missionaries: %d\n", numcan, nummiss);
  exit();
  return 0;
}

int random(int max){
  rands = rands * 166425+ 1013904233;
  return (int)(rands % max);
}
                                                                          12,2          16%
```

This is what we get for our test results.



```
$ river
basic test case
Row Boat with 1 cannibal 2 missionaries
cannibals: 0 and missionaries: 0

edge case shouldn't cross the river
cannibals: 2 and missionaries: 1

random test cases that should print out 12 row boats with 0 missionaries and 0 c
annibals or else should print out less than 12 rowboats with 2 cannibals and 1 m
issionary as result
Row Boat with 1 cannibal 2 missionaries
Row Boat with 3 cannibals
Row Boat with 1 cannibal 2 missionaries
Row Boat with 1 cannibal 2 missionaries
Row Boat with 1 cannibal 2 missionaries
Row Boat with 3 cannibals
Row Boat with 1 cannibal 2 missionaries
Row Boat with 1 cannibal 2 missionaries
Row Boat with 1 cannibal 2 missionaries
Row Boat with 3 cannibals
Row Boat with 1 cannibal 2 missionaries
Row Boat with 1 cannibal 2 missionaries
cannibals: 0 and missionaries: 0
```

This test result for random was all successful in rowing 12 boats.

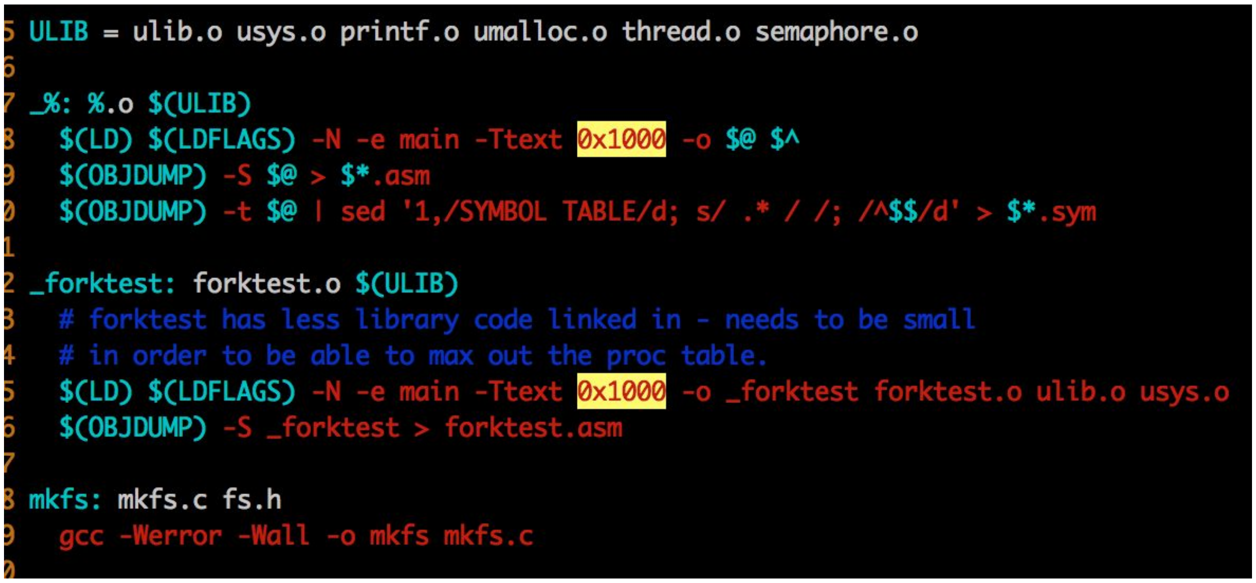


```
random test cases that should print out 12 row boats with 0 missionaries and 0 c
annibals or else should print out less than 12 rowboats with 2 cannibals and 1 m
issionary as result
Row Boat with 3 cannibals
Row Boat with 3 missionaries
Row Boat with 1 cannibal 2 missionaries
Row Boat with 3 cannibals
Row Boat with 1 cannibal 2 missionaries
Row Boat with 1 cannibal 2 missionaries
Row Boat with 1 cannibal 2 missionaries
Row Boat with 3 cannibals
Row Boat with 1 cannibal 2 missionaries
Row Boat with 1 cannibal 2 missionaries
Row Boat with 1 cannibal 2 missionaries
cannibals: 2 and missionaries: 1
```

Here is another output where the random test case only prints out 11 row boats to prove that it is random.

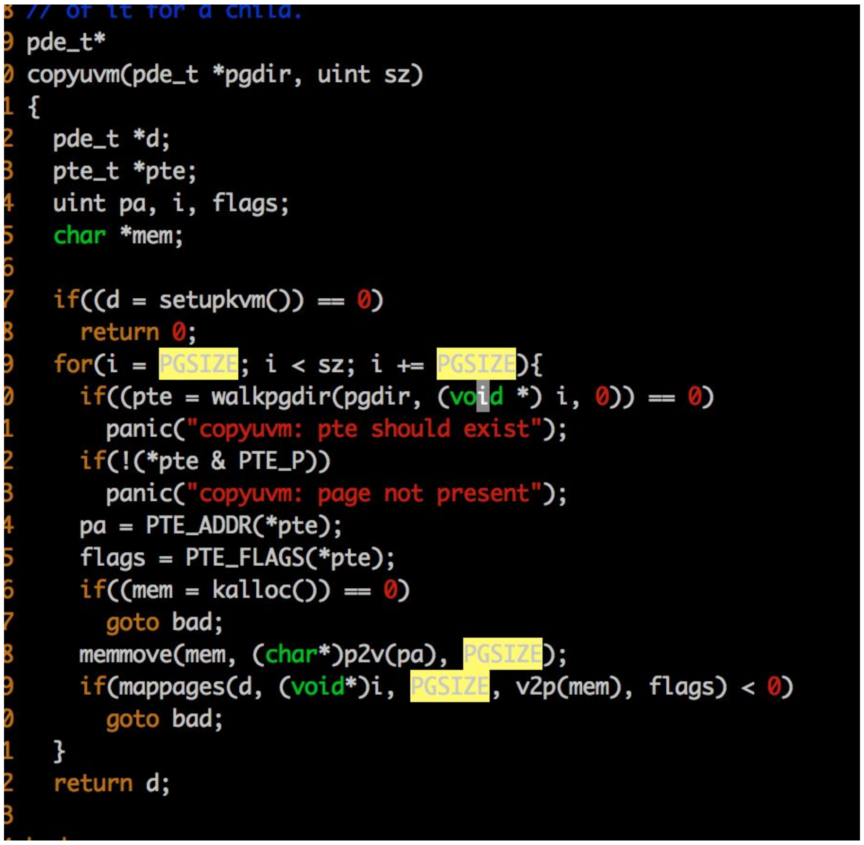
part 2.1: what parts you changed. how to run your test and screen shots of outputs.

We changed the Makefile addresses from 0 to 0x1000



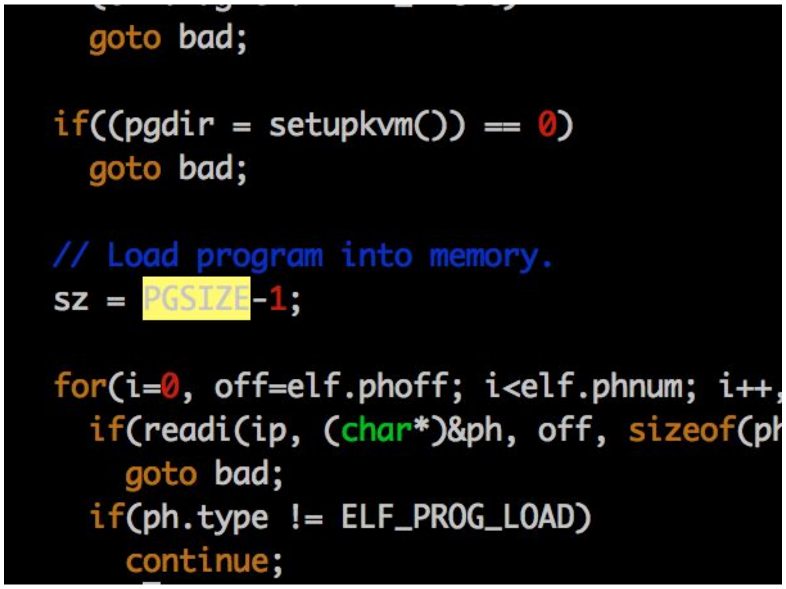
```
ULIB = ulib.o usys.o printf.o umalloc.o thread.o semaphore.o

_%: %.o $(ULIB)
	$(LD) $(LDFLAGS) -N -e main -Ttext 0x1000 -o $@ $^
	$(OBJDUMP) -S $@ > $*.asm
	$(OBJDUMP) -t $@ | sed '1,/SYMBOL TABLE/d; s/ .* / /; /^$$/d' > $*.sym

_forktest: forktest.o $(ULIB)
	# forktest has less library code linked in - needs to be small
	# in order to be able to max out the proc table.
	$(LD) $(LDFLAGS) -N -e main -Ttext 0x1000 -o _forktest forktest.o ulib.o usys.o
	$(OBJDUMP) -S _forktest > forktest.asm

mkfs: mkfs.c fs.h
	gcc -Werror -Wall -o mkfs mkfs.c
```

We changed starting index in vm.c from 0 to PGSIZE



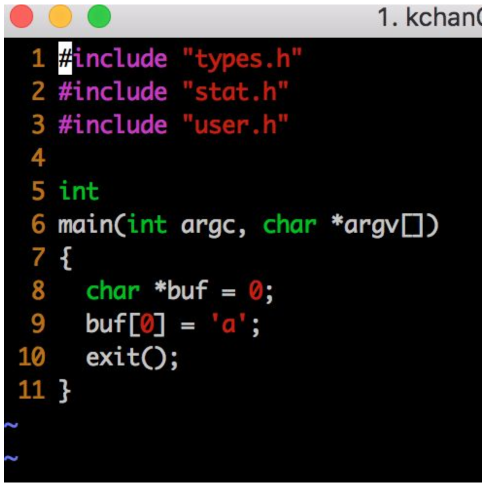
```
// of it for a child.
pde_t*
copyuvm(pde_t *pgdir, uint sz)
{
  pde_t *d;
  pte_t *pte;
  uint pa, i, flags;
  char *mem;

  if((d = setupkvm()) == 0)
    return 0;
  for(i = PGSIZE; i < sz; i += PGSIZE){
    if((pte = walkpgdir(pgdir, (void *) i, 0)) == 0)
      panic("copyuvm: pte should exist");
    if(!(*pte & PTE_P))
      panic("copyuvm: page not present");
    pa = PTE_ADDR(*pte);
    flags = PTE_FLAGS(*pte);
    if((mem = kalloc()) == 0)
      goto bad;
    memmove(mem, (char*)p2v(pa), PGSIZE);
    if(mappages(d, (void*)i, PGSIZE, v2p(mem), flags) < 0)
      goto bad;
  }
  return d;
```

We changed sz to PGSIZE-1 in exec.c

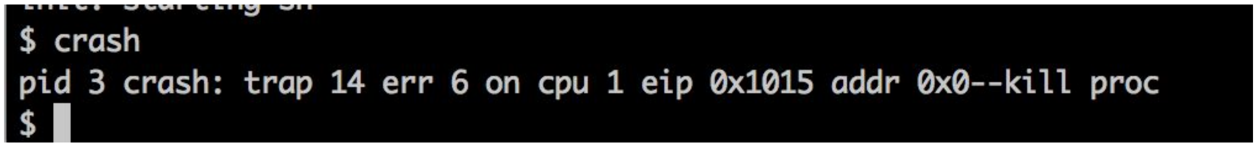


```
    goto bad;

  if((pgdir = setupkvm()) == 0)
    goto bad;

  // Load program into memory.
  sz = PGSIZE-1;

  for(i=0, off=elf.phoff; i<elf.phnum; i++,
    if(readi(ip, (char*)&ph, off, sizeof(ph
      goto bad;
    if(ph.type != ELF_PROG_LOAD)
      continue;
```

In our test case, we just dereferenced a null pointer.



```
#include "types.h"
#include "stat.h"
#include "user.h"

int
main(int argc, char *argv[])
{
  char *buf = 0;
  buf[0] = 'a';
  exit();
}
~
~
```



```
$ crash
pid 3 crash: trap 14 err 6 on cpu 1 eip 0x1015 addr 0x0--kill proc
$ 
```