OPERATING SYSTEMS CompSci 510, SPRING 2018

Out: Monday, January 8, 9:00
Due via Hardcopy: Monday, January 8, 12:00

This is a closed-book, no-computer exam. You must answer the questions on your own without any external assistance.

Read the entire exam through before you begin working. Read each question carefully, and note all that is required of you. Keep your answers *clear and concise*. Often when you write more than you need you end up saying things that are incorrect. I don't expect you to need more than a page for any question.

You are to abide by the Duke University honor code. Additionally, you may not consult with any other person about any part of this exam until the exam period ends at 12:00, Monday. This is true even if you have handed in your exam. Please sign this page to indicate that you have completed your exam within these rules. Exams without corresponding, signed cover pages will not be graded

NetID:

Signature:

1. Happy New Year!

Write pseudo-code for three thread functions, happy, new, and year. For one happy thread, one new thread, and one year thread, the program must strictly switch from thread happy printing “Happy ” to thread new printing “New ” to thread year printing “Year!” for a total output of N lines. In other words, the output should always be of the following form:

Happy New Year!

…

Happy New Year!

with **N** total lines of output. The value of N is set for you in main. *Your solution must work for any thread interleaving, and you must use locks and condition variables for synchronization.*

```
// global variables

int N; // num times to print “Happy New Year!”

happy () {




  printf (“Happy “);





}

new () {




  printf (“New “);





}
```

```
year () {






  printf (“Year!\n“);





}

main(int argc, char *argv[]) {
  N = atoi (argv[1]); // read N from command-line args
  thread_create(happy); // fork the “happy” thread
  thread_create(new); // fork the “new” thread
  thread_create(year); // fork the “year” thread
}
```

2. Miscellany

For each sub-question below, I will only read the first two sentences of your answer. Think before you start writing.

a) During copy-on-write what should happen when a child process tries to write to a page that it inherits from its parent?

b) What actions must the kernel take when switching processes to ensure that address translations are correct for the incoming process?

c) Describe how user code safely invokes the kernel.

d) Describe a file-system operation that demonstrates why file systems must write blocks to disk in the correct order to prevent corruption.

3. Reader-writer locks

Write pseudo-code for a simple implementation of reader/writer locks using locks and conditional variables. Do not worry about initializing your global variables.

```
// GLOBAL VARIABLES GO HERE




acquireReadLock () {











}

releaseReadLock () {









}
```

```
acquireWriteLock () {









}

releaseWriteLock () {









}
```