Quiz #4, on Wednesday, October 17th, will consist of questions taken from or inspired Parts I and II of this homework.

Part I — Questions

1. Give three signal numbers and explain their role.

2. What statement in `C` would you use to send the signal `14` to a process with id `234`?

3. Why would a programmer want to implement a custom signal handler?

4. What justifies disallowing the handling of the `SIGKILL` and `SIGSTOP` signals?

5. Give one similarity and one difference between thread and process.

6. Give two benefits provided by threads, and one difficulty the programmer has to face to use them.

7. From a programmer perspective, what makes a program easy to divide between threads?

8. What is the difference between data parallelism, and task parallelism?

9. Why is there normally no need to "protect" one thread from the others within the same process?

10. What is the datatype used to store the `id` of a thread?

11. What does the `C` statement `pthread_join(tid, NULL)` do?

12. If the `C` statement `pthread_create(&var_pthread, NULL, func, &y)` returns the value 0, what can you conclude? You can use `man pthread_create` to find out.

13. What is implicit threading?

14. What does the `C` statement `pthread_exit()` do?

15. What is the name of the model where many user-level threads are all mapped onto a single kernel thread?

Part II – Problems

This time, the three problems require a computer. They are rather short, but critical for your understanding of this lecture. As usual, I'll assume that you will have successfully completed those tasks by Wednesday, October 17th, so don't wait and let me know if you had difficulties solving them.

Preamble Open VirtualBox, select the OSC-2016 image (or whatever image you are using), and click on "Settings". Then, click on "System", and then "Processor", and make sure that the virtual machine is running with more than one processor. Start your virtual machine, create a "4" folder in your "Desktop/HW/" folder, and download and extract the code located at `http://spots.augusta.edu/caubert/teaching/2018/fall/csci3271/code/hw4.zip`.

Problem 1

Consider the code displayed below and called `thread_pb1.c` in the archive you downloaded and answer the following:

1. Compile the program, using `gcc -l pthread thread_pb1.c`, and execute it *multiple times*, using `./a.out`. Is the result always the same? How do you explain this behavior?
2. Edit a copy of the program, so that the variable declaration `int i = 0;` appears only once, instead of three. We want the variable to be shared among all the threads of this program, so where should that variable declaration go? What do you observe when you run the program: are the printing instructions performed five time, overall? Why, or why not? Using statement like `printf("%d\n", i);`, "track" the value of `i`.

```c
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void* print1() {
        int i = 0;
        while (i < 5) {
                printf("Hi, how do you do?\n");
                sleep(0.1);
                i++;
        }
        pthread_exit(0);
}

void* print2() {
        int i = 0;
        while (i < 5) {
                printf("Hello, how are you?\n");
                sleep(0.1);
                i++;
        }
        pthread_exit(0);
}

int main() {
        pthread_t tr1, tr2;
        pthread_create(&tr1, NULL, print1, NULL);
        pthread_create(&tr2, NULL, print2, NULL);
        int i = 0;
        while (i < 5) {
                printf("Let me introduce you.\n");
                sleep(0.1);
                i++;
        }
        pthread_join(tr1, NULL);
        pthread_join(tr2, NULL);
        exit(0);
}
```

Problem 2

Read the section 26.3, *Why It Gets Worse: Shared Data* of "Operating Systems: Three Easy Pieces", at `http://pages.cs.wisc.edu/~remzi/OSTEP/threads-intro.pdf` (archived version at `https://web.archive.org/web/*/http://pages.cs.wisc.edu/~remzi/OSTEP/threads-intro.pdf`). You can ignore the discussion about `Pthread_create()`, and find the code (slightly adapted) below and called `thread_sync.c` in the archive you downloaded

Beware, the code needs to be compiled using `gcc -l pthread thread_sync.c`, and executed with, e.g., `./a.out 100000`: you need to feed an integer when you execute it at the command line.

We will use this code and to get the discussion started on the need to synchronize threads.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int max;
volatile int counter = 0; // shared global variable

void *mythread(void *arg) {
        char *letter = arg;
        int i; // stack (private per thread)
        printf("%s: begin [addr of i: %p, addr of counter: %p]\n", letter, &i, &counter);
        for (i = 0; i < max; i++) {
                counter = counter + 1; // shared: only one
        }
        printf("%s: done\n", letter);
        return NULL;
}

int main(int argc, char *argv[]) {
        if (argc != 2) {
                fprintf(stderr, "usage: main-first <loopcount>\n");
                exit(1);
        }
        max = atoi(argv[1]);

        pthread_t p1, p2;
        printf("main: begin [counter = %d] [addr of counter: %p]\n", counter, &counter);
        pthread_create(&p1, NULL, mythread, "A");
        pthread_create(&p2, NULL, mythread, "B");
        // join waits for the threads to finish
        pthread_join(p1, NULL);
        pthread_join(p2, NULL);
        printf("main: done\n [counter: %d]\n [should: %d]\n", counter, max*2);
        return 0;
}
```