2 Possible Questions

(There are ~~63~~ ~~11~~ 10 questions left. Right now there are ~~10~~ 12 people in this doc, so we could grab ~~6.3~~ ~~1.1~~ ~~0.92~~ 0.8333333333333 questions each and be done if everyone is okay with that. **(TONY DELIVERED)**

RED Questions = AREN'T IN THE GOOGLE DOC

- The non-reentrancy of some functions can be spotted based only on a description of the function's interface. Explain how to do so.
    - **Standard convention for the naming of reentrant function by suffixing the function name with "_r".**
    - In general, must understand function implementation to determine if it is reentrant
    - **But some non-reentrant functions can be spotted from interface alone**
    - **Giveaway: returns pointer to static/global**
    - Example: asctime(3)
    - There is now also asctime r(3) — caller must supply buffer to accept return value
    - Others: localtime(3), gmtime(3), ctime(3), strtok(3), readdir(3)

- Write C code to create a new Pthread, wait for it to finish executing, then collect its result. The thread's start function should accept 3 arguments: an integer, a string, and a floating point number. The thread should return an integer.
    - ```
      typedef struct {
              int i;
              char* str;
              float fpt;
      } args_t;

      void* myfunction(void* data) {
              int retval;
              args_t* args = (args_t*)data;
              …
              return (void*)retval;
      }

      pthread_t thread;
      void* retval;
      args_t args;

      pthread_create(&thread, NULL, &myfunction, &args);
      ```
      ```

```
pthread_join(thread, &retval);
return (int)retval;
```

- Explain why the disabling of interrupts is no longer a viable synchronization technique, even inside operating systems.
    - You can't trust people to make sure they re-enable interrupts in their own code
    - Wasteful - halts all threads, even if only some want mutex
    - Doesn't work on multiprocessors
- Contrast the functionality provided by a monitor versus the functionality provided by Java's synchronized keyword.
    - *synchronized* means: at most one of all the methods declared synchronized may be executing at a time
    - *synchronized* keyword is not the complete monitor concept—that requires "wait" and "signal" operations—but it is the "monitor lock" part of the concept

PREVIOUS EXERCISE TO #5: Write Erlang code that starts a new thread. The thread should initially execute function "func" from module "mod" with three arguments a,b, and c. There is no need to retrieve the function's result.

ANSWER: NewThread = spawn(mod, func, [a, b, c]).

- Repeat the exercise above except that the new thread should be "monitored" by the initial thread.
    - { NewThread, _ } = spawn_monitor(mod, func, [a, b, c]).
- When an Erlang thread that is being monitored crashes, how does the monitoring thread learn about the crash?
    - 5 tuple message is sent to the parent if you used spawn_monitor
    - **{ 'DOWN', Ref, process, Pid, Reason }**
    - 'DOWN' and process are atoms; i.e., literal values
    - Ref is variable; has same value as returned by spawn monitor/3
    - Pid and Reason are variables
- Erlang does not permit threads to share memory. Explain how this feature aids the construction of reliable concurrent code.
    - Don't have to worry about deadlocks from locking mutexes
    - Failure Isolation:
        i. **Threads and Locks Model**
            1. Thread failure leaves address space in an unpredictable state. All surviving threads must be able to cope.
        ii. **Actors Model**
            1. Thread failure removes one Actor. All other Actors are unaffected.
    - Distribution:
        i. **Threads and Locks Model**
            1. No simple way to have only one thread execute on another machine.
        ii. **Actors Model**
            1. If one Actor is on another machine, messages to/from it are network messages. Very simple.

- If many Erlang threads each had to read and make updates to a single very large data structure, how would you design the code to do so?
    - Make a single module that receives messages from threads and updates the large data structure accordingly???
    - Sounds right… the receiver module would wait on a recursive receive loop for a message pattern from the sender threads. As the messages come in, they are received or wait on the queue for the receiver module to grab and process them. No messages drop off since they would be put on the queue if they cannot be received right away, and each message would be processed one-by-one since the receiver module uses the message queue, so there would be no concurrency issue or missing messages.
- Explain what "model checking" is.
    - A formal method for the verification of programs, usually concurrent ones. Model checking verifies a program by using software to analyze its state space, as opposed to mathematical deductive methods.
    - Steps: Develop a *model* of the program. Prove properties of the model. Translate model into code.
- What advantage does model checking have compared to testing?
    - Testing is flawed – cannot cover all the cases & the hardest cases to program are also hardest to test
    - Approach of "prove the model, not the program" is valuable provided that there is simple, accurate correspondence between model and program
        i. Easy to create model corresponding to program's design
        ii. Easy to prove properties of model
        iii. Easy to create program corresponding to proven model
    - Testing interleavings in concurrent programs is super hard because interleaving is controlled by OS scheduler. Model checking tests all interleavings.
- What disadvantage does model checking have compared to testing? What disadvantage does model checking have compared to program proof?
    - Testing
        i. TODO
        ii. Testing scales better (more easily). Model checking can suffer from state space explosion.
    - Proof
        i. TODO
        ii. A model might not have a perfectly accurate correspondence with the program it is modelling.
        iii. A program proof will guarantee the program is correct (assuming the correct?(model?) has no errors)