Final Exam

(100 points)

This exam has a total of 6 questions, spread over 9 pages, including this cover page.
Answer ANY 5 questions

Date: May 12, 2015

Student Name: ....

Student ID: ....

You are allocated a maximum amount of space to answer each question. (We have provided sufficient lines.) Adhere to those limitations when you formulate your answers. Do not use the backside of the pages. No additional pages are allowed. Make an effort to write in a readable fashion. We will skip over (and therefore not grade) non-readable portions.

"I have adhered to the Aggie Code of Honor."

1 (20)	
2 (20)	
3 (20)	
4 (20)	
5 (20)	
T (100)	

Signature:

....

1a. (10 points) Match each term in the left column to the definition/description in the right column that **fits best**. Do this by filling in the void entries on the left:

( )	Pthread	(A)	Is performed without interruption
( )	Unix Pipe	(B)	Waits till all id'd signals are caught
( )	FIFO	(C)	Mechanism for IPC via message passing
( )	UDP	(D)	Synchronization primitive
( )	Socket	(E)	Coordination between threads
( )	Mailbox	(F)	Represented by address and port number
( )	SIGWAIT	(G)	Thread that corresponds to POSIX standard
( )	Atomic Operation	(H)	Named IPC structure that resides in Kernel
( )	Mutex	(I)	Protocol for Network Layer in Internet Programming
( )	Synchronization	(J)	Mechanism for inter-process communication via the use of file descriptors

1b. (10 points) Circle the following statements TRUE or FALSE.

T F A critical section is a portion of memory that cannot be shared amongst processes.
T F Blocking a signal means to delay receipt of the signal until later
T F Signal system call is used to specify a response to a particular signal.
T F Applications can experience race conditions only in systems with multi-threaded kernels
T F A signal mask is a set of signals a process is ignoring
T F The major difference between datagram sockets and stream sockets is stream sockets are not reliable
T F A program can create a named pipe by calling "pipe" system call
T F Shared memory is faster for communication than pipes because the kernel stores shared memory segments in the CPU cache
T F Message passing uses shared memory segments as a mechanism for IPC
T F Datagram sockets would be a good choice for transmitting background music for elevators instead of automated teller machine transactions

2 [20 points] Circle the best answer for each of the 10 multiple choice questions below. *If in doubt about your answer, use the space below to explain* ***in less than 2 sentences***.

- A program with two threads of execution can
  - a. do two things at the same time
  - [✓ b. open twice as many files as a program with one thread]
- If a thread uses pthread_join to wait for itself
  - [✓ a. pthread_join returns an error code]
  - b. the process will freeze
- All the threads in a process have access to
  - a. only static global variables
  - [✓ b. all global variables in the process]
- A web server sends output of programs to the client by
  - a. opening a new socket for the program
  - [✓ b. using dup2 to redirect standard output]
- One thing a server does that a client does not do is
  - a. a server loops
  - [✓ b. a server waits for a connection]
- A server can handle several requests at once by
  - [✓ a. using fork to create a new process for each request]
  - b. using socket to create a new socket for each request
- To prevent race conditions, threads should use
  - a. signals
  - [✓ b. mutex objects]
- The input redirection symbol "<" is processed by
  - [✓ a. the program that reads the data]
  - [✓ b. the shell]
- The thread that accepts calls in a web server does not close the socket
  - [✓ a. because that would close it for the worker thread, too]
  - b. because it is closed automatically when the worker thread finishes
- The shell notation "prog1 | prog2" is a request to
  - a. send all output from prog1 to the standard input of prog2
  - [✓ b. send the standard output of prog1 to the standard input of prog2]

3. [20 points] Assume you want to implement the Bounded Buffer Producer-Consumer (BBPC) problem using a pair of hypothetical system calls sleep() and wakeup(procid). The system call sleep() suspends the current process until it is awoken by another process. The system call wakeup(procid) awakens the process with the given process id ‘procid’ if it is sleeping; otherwise, the call simply returns without effect. A possible implementation of BBPC Problem using sleep() and wakeup(procid) would look as follows:

```
/* shared variables */
const N;                              /* size of buffer          */
int count = 0;                        /* number of items in buffer */

process Producer() {
  int item;                           /* local variable          */

  while (TRUE) {

    item = produce_item();

    if (count == N) sleep();          /* if buffer is full, wait */

    deposit_item(item);               /* put item in buffer      */
    count = count + 1;

    if (count == 1) wakeup(Consumer); /* was buffer empty?       */

  }
}

process Consumer() {
  int item;                           /* local variable          */

  while (TRUE) {

    if (count == 0) sleep();          /* if buffer empty, wait   */

    item = remove_item();
    count = count - 1;

    if (count == N-1) wakeup(Producer);/* was buffer full?       */

    consume_item();

  }
}
```

(See next page for questions)

(3a) (10 points) The implementation on the previous page is known to have a fatal race condition. **Describe the race in simple and direct terms**. Note: Assume that all arithmetic operations are atomic.

(3b) (10 points) How would you fix it for the case of a single producer and a single consumer? Note: You are not allowed to use synchronization primitives other than sleep() and wakeup() system calls described above. **Please annotate the code with the proposed fix(es).**

4. (20 points) Rebecca has written a multithreaded program which produces an incorrect answer some of the time, but always completes. She suspects a race condition. Which of the following are strategies that can reduce, and with luck eliminate, race conditions in her program? **Circle the R, E, or U below to indicate that the given approach can Reduce race conditions, Eliminate them, or is Useless when it comes to race conditions. Use the space below to write your reasoning.**

R E U Separate the multithread program into multiple single-threaded programs, and run each thread in its own process. Share data between the programs via named pipes and read and write calls. No other changes to the program.

R E U Same as above, but share data between the programs via shared memory segments.

R E U Apply the "one-writer" rule. Here you modify the program so that each variable has only one writer, i.e. only one thread can write to a shared variable. Multiple threads can read from it.

R E U Ensure that each shared variable is protected by some lock.

5a. (13 points) You are surfing on stackoverflow.com (a popular forum for programming related questions) and you see a post of a programmer trying to understand a strange behavior of his program. When the programmer compiles the program as listed below, the output reads "true". If the programmer switches lines (A) and (B) around, the output becomes "false false". **Explain briefly why this could be happening.**
(Recall that pthread_create() creates a thread to execute the given function)

```
          bool done = false; /* global variable */

          void * dotest(void * _dummy) {
              if (!done) {
/*(A)*/           done = true;
/*(B)*/           cout << done << endl;
              }
          }

          int main(int argc, char * argv[]) {
              pthread_t thread_id;
              pthread_create(& thread_id, NULL, dotest, NULL);
              dotest(NULL);
          }
```

5b. (7 points) A binary semaphore is initialized to TRUE. Then 5 P() operations are performed on it in a row followed by 8 V() operations. Now 5 more P() operations are performed on it. **What is the number of processes waiting on this semaphore? Show your work.**

6a. (10 points) In this program

```
#include <signal.h>

void sigusr(int signo) {
  if (signo == SIGUSR1)
      printf("received signal SIGUSR1\n");
  else
      printf("received some weird signal!\n");
  return;
}

int main(int argc, char** argv) {
  if (signal(SIGUSR1, sigusr) == SIG_ERR)
      printf("cannot catch signal SIGUSR1!\n");

  for( ; ; )
     pause();
}
```

What happens if you remove the for(; ;) statement? Why?

6b. The Aggie Network

Some of you decide to form a startup company. First point of business is to make certain that you are able to connect to all your clients.

(6b.i) (5 points) Since you're currently a super small startup, all of your servers reside in a single room and are directly connected to the same gateway router. All incoming and outgoing traffic pass through this router. **To make routing faster, you add an extra tag to each request** (both client and server side). You then upgrade your router with your own custom firmware that forwards the packets by reading this application-level tag. You can think of this as clients sending requests with serverIDs as the tag and your router forwarding on these serverIDs directly.

Is this design feasible? Why or why not? (hint: think of the OSI model)

(6b.ii) (5 points) Your startup wants to make sure that you move fast and break things. Instead of using and understanding TCP, you have designed our own transport layer protocol (Rudder Datagram Protocol, RDP) that gives no guarantees since you want to fail fast. You decide to assign it a random protocol number, 42, since the OS needs this protocol within the IP field to determine how to handle it. Unless you receive a success reply back from your Server, you assume the transaction failed. Your protocol only includes the two port numbers (source and destination) within the transport layer. This is even less overhead than UDP!

Is this design feasible? Why or why not? (hint: think of the OSI model)