CS 3214 Final Exam

This is a closed-book, closed-internet, closed-cell phone and closed-computer exam. However, you may refer to your prepared notes on 1 double-sided page. **Your exam should have 11 pages with 4 topics totaling 150 points. You have 120 minutes**. Please write your answers in the space provided on the exam paper. If you unstaple your exam, please put your initials on all pages. You may use the back of pages if necessary, but please indicate if you do so. Answers will be graded on correctness and clarity. You will lose points if your solution is more complicated than necessary or if you provide extraneous, but incorrect information along with a correct solution.

To be considerate to your fellow students, if you leave early, do so with the least amount of noise.

Name (printed) ____

I accept the letter and the spirit of the Virginia Tech undergraduate honor code – I have not given or received aid on this exam.

(signed) ______________________________________

#	Problem	Points	Score
1	Concurrency and Synchronization	25	
2	Memory	45	
3	Communication and Networking	60	
4	Essay Question technical content writing	 14 6	
	Total	150	

1. Concurrency and Synchronization (25 points)

c) (25 points) Threads in a given application exert different loads on the system's database. The load that a given thread exerts on the database is measured by a simple positive integer value. If the total load exerted by all threads on the database exceeds MAX_LOAD the overall system performance declines precipitously.

The threads must be synchronized so that the total load on the database is never more than MAX_LOAD. Complete the two procedures shown below with the code needed to achieve the correct synchronization. Assume that each thread uses `start_DB` before using the database and `end_DB` afterwards.

Declare any global variables needed. Be sure to show all initializations.

```
void start_DB(int  load);
void end_DB (int load);
```

answer

```
int current_load = 0;
pthread_mutex_t lock     = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  load_low = PTHREAD_COND_INITIALIZER;

void start_DB(int load) {
  pthread_mutex_lock(lock);
  while (load + current_load > MAX_LOAD)
    pthread_cond_wait(load_low, lock);
 current_ load = current_load + load;
 pthread_mutex_unlock(lock)
}

void end_DB(int load){
  pthread_mutex_lock(lock);
  current_load = current_load- load;
  pthread_cond_broadcast(load_low);
  pthread_mutex_unlock(lock);
}
```

2. Memory (45 points)

a) (10 points) One of the code samples below has a memory management issue and one has an optimization blocker. Identify which code example has which problem and specifically describe how the problem occurs. You do not have to fix the problem.

```
(1)  struct mags {
        int sum;
        int prod;
     }

     struct mags* find(int x, int y) {
        struct mags m;
        m.sum = x + y;
        m.prod = x*y;
        return &m;
     }
```

```
(2)  void find(int x, int y, int* sum, int* prod) {
        *sum  = x + y;
        *prod = (*sum) * (*sum);
     }
```

```
(3)  int find(int x, int y) {
        int m = 0;
        for(i=0; i< x; i++)
          for(j=0; j< y; j++)
             m = m + j*bigger(x,y);  // bigger returns the
                                     // larger of x and y
        return m;
     }
```

answer:

(1) has a memory management problem because it returns a pointer to a variable allocated in the current stack frame which is de-allocated when the procedure returns

(3) has an optimization blocker because the compiler cannot know that bigger can be moved out of the inner loop

b) (12 points) Examine each of for the four code examples shown below. Each example involves the code in two files named one.c and two.c. In each case state whether the code will or will not compile. In those cases where the code does compile state what, if any, concerns there is about the correctness of the code's execution. You do NOT have to fix the problems. Use the back of the page for additional space if needed.

Example	**one.c**	**two.c**

1	int x;
	void one(int y) {
	  double a;
	  …
	}
		int x = 0;
		void two(double z) {
		  float y;
		  …
		}

2	int x;
	void one(int y) {
	  double a;
	  …
	}
		extern int x;
		void two(double z) {
		  float y;
		  …
		}

3	int x=0;
	void one(int y) {
	  double a;
	  …
	}
		double x = 0;
		void two(double z) {
		  float y;
		  …
		}

4	int x;
	void one(int y) {
	  double a;
	  …
	}
		double x = 0;
		void two(double z) {
		  float y;
		  …
		}

answers:

(1) will compile but may not be what is intended
(2) will compile and execute
(3) will not compile because of conflict between two strong global symbols
(4) will compile but is almost certainly wrong because of mismatched type for symbol x

c) (9 points) Examine each of for the three code examples shown below. In each case state whether the code is or is not thread safe and why the code is or is not thread safe.

Example	**Code**
1	int safe(int x) { int y; y = x *2 + x; return y; }
2	int* safe(int x) { int y*; y = malloc(sizeof(int)); y* = x *2 + x; return y; }
3	int y; void safe(int x) { y = x *2 + x; }

answer:

(1) thread safe – data allocated on stack specific to each thread
(2) thread safe – data allocated in memory for each thread
(3) NOT thread safe because of global variable that is unprotected by synchronization

d) Indicate the role that virtual memory plays in each of these situations. If the virtual memory plays no role write NONE.

i. (4 points) What the system does in response to the fork() system call.

answer: create a duplicate page table mapping in child; use copy-on write to avoid creating new physical pages

ii. (3 points) Preventing code stored on the stack from executing, thus preventing some forms of buffer overflow attacks.

answer: each page table entry contains access control bits; turn off the execute bit in entries that map to pages containing the stack

iii. (3 points) Improving the locality of programs

answer: NONE

iv. (4 points) Supporting dynamic shared libraries.

answer: create different mappings of virtual to physical memory in each of the processes sharing the library code

3. Communication/Networking (60 points)

a) (10 points) Identify three similarities and three differences between pipes and sockets.

answer:

Similarities: both are accessed via file descriptors
both are reliable
both are ordered
both have short-read issues

Difference: pipes are not across machines while sockets are
pipes are unidirectional while sockets are bidirectional
a pipe has two file descriptors while a socket has only one
a socket has an address attached to it while a pipe does not.

note: other reasonable similarities/differences accepted

b) You are asked to implement a server which, for each accepted connection from a client, launches a child process to execute the code in the executable file "serve-client". The child process is configured so that data sent by the client is received on the standard input of the child and the data produced by the child process on its standard output is sent to the client. A utility function, `socket_setup`, creates a protocol-independent socket on which incoming client requests can be accepted and returns the file descriptor for that socket.

i. (5 points) The `socket_setup` function uses four socket-related system calls. Name and briefly describe each one. You do NOT need to show detailed code.

answer:

- socket (creates socket and identifies general protocol characteristics)
- get_addrinfo (returns the Internet address for this socket)
- bind (attaches the Internet address to the socket)
- listen (specifies the queue length for incoming requests and allows incoming requests to be accepted)

ii. (15 points) Use the `socket_setup` function to write the code for the server that accepts incoming client connections and launches the child process as described above. You do not need to worry about error conditions in this code.

answer:

```
int sockfd = socket_setup();
while (true) {                                    // condition does not matter
   int clientfd = accept(sockfd, NULL, NULL);   // NULL parameters OK
   if(fork() == 0) {   /* child /*
      dup2(clientfd, 0);
      dup2(clientfd, 1);
      close(clientfd);   // child closes original file descriptor after dups
      execlp("serve-client", "serve-client", (char*)0)
    }
   close(clientid);  // parent also closes the file descriptor
}
```

c) HTTP allows a "non-persistent connection". Similarly, the TCP/IP protocols provide through UDP a "connectionless-oriented" service. These two seem well matched.

i. (3 points) Explain what is meant by a non-persistent connection.

answer: a connection that is closed after the current request has been satisfied

ii. (3 points) Explain what is meant by a connectionless-oriented service.

answer: one where each packet is delivered independently of any other packet between the same sender and receiver.

iii. (4 points) If you are implementing a non-persistent connection would you use UDP as the underlying protocol? Explain why or why not.

answer: No, because reliable ordered information is needed to send the response properly to the client.

d) (20 points) You are asked to serve on a standards committee to draft a new version of the HTTP protocol for periodic data streams. Examples meant to be covered by the protocol include stock quotes, news feeds, weather updates, etc. The model for periodic data streams is that, once a stream is opened by a client, the client will receive data periodically until the client sends a request to close the stream. The names of the periodic data streams available on a given server are denoted by the names of files in the root directory `stream`. For each periodic data stream the client selects a port number that it will use to receive the data sent by the server. A client may have several streams open at one time to the same server. The client should be able to specify what protocol is used to send the stream data.

Show and briefly describe an example illustrating at least **FIVE** extensions that you would make to the HTTP protocol.

answer:

add new version number to the first line of a request, example HTTP/1.2
add definition for data streams (e.g., /stream/newsfeed)
add two new verbs for OPEN and CLOSE
add new header line to specify client port
add new header line to specify protocol

example:

OPEN /stream/newsfeed HTTP/1.2
Receive-port: 8010
Protocol: UDP

CLOSE /stream/newsfeed HTTP/1.2
Receive-port: 8010

4. Essay Question (20 points)

Recall the LRU memory management algorithm that is typically used with virtual memory management, where the least recently used page is evicted to make room for a page this is being brought in as part of the page-fault servicing. The theoretical LRU algorithm works by moving a page that experiences a hit to the head of a list, and when needed select a page on the tail of the list for eviction. Identify a key challenge in implementing the LRU algorithm in actual virtual memory management. Discuss the design of a practical system that implements the LRU for virtual memory management.

Note: *This question will be graded both for technical content of your arguments (14 points) and for your ability to communicate effectively in writing (6 points). Your answer should be well-written, organized, and clear.* ***Your answer must be legible – points will be deducted for parts that cannot be read with normal effort. Use the back of this page as needed.***

The writing should focus on details specific to LRU, to the LRU implementation (not the LRU algorithm), and concern system-level factors (not simply data structure concerns when these are not directly tied to system factors).

Technical content:

Identify inability to know about page hits in a VM-based system, i.e., while a page fault is routed to the kernel for processing, the kernel is not informed on a page hit. This renders the ability to maintain a recency-ordered list of pages impossible.(4)

Describe a clock-based or similar system to implement the LRU. Identify that hardware support is needed to identify pages that have been accessed between page faults and that special clock-based data structures are required to approximate least recently accessed pages. (6)

Describe the complete design with dirty page management (2-bit clock). (4)

Writing:

Clear and complete sentences (2)
Logically composed sentences (4)

Technical Content: __________ Writing: __________