CSCI 4210 — Operating Systems
Final Exam Prep and Sample Questions (document version 1.2)
SELECTED SOLUTIONS

- **(v1.1)** The Registrar has scheduled our Final Exam for 6:30-9:30PM ET on Monday 5/10; however, the exam will be available to take during the extended window that starts at 4:30PM ET on Monday 5/10 and ends at 4:30AM ET on Tuesday 5/11
- The Final Exam will be a 120-minute exam, but you will have a full **four-hour window of time** to complete and submit your exam solutions
- Submitty will start the four-hour "clock" for you when you first download the exam, so please plan accordingly by avoiding any distractions or interruptions; and note that **you must start your exam by 11:59PM ET on Monday 5/10 to have the full four-hour window**
- On the exam, there will be one auto-graded question worth 20 points, the 1 point for submitting a PDF, and 11 free response questions worth the remaining 79 points; for the auto-graded questions, you will submit your code directly and have 20 penalty-free submissions; for free response questions, all of your answers must be collated into **a single PDF file called** `upload.pdf`
- The Final Exam is open book(s), open notes; given that you are working remotely, you may use any and all of the posted course materials, including all previous questions and answers posted in the Discussion Forum
- Exam conflicts are resolved according to RPI policy: `https://tinyurl.com/4m2ycb3y`
- Make-up exams are given only with an official excused absence (`http://bit.ly/rpiabsence`); also re-read the syllabus
- The Final Exam is comprehensive, covering everything we have done this semester, including all assignments due through Monday 5/3; however, the auto-graded question will not cover network programming
- To best prepare for the exam, focus on the previous exams, the previous "exam prep" PDFs, coding examples, and suggested "to do" items, as well as other code modifications you can think of; review `man` pages and behavior for all system calls and library functions we have covered
- For free response questions, be as concise as you can in your answers; long answers are difficult to grade
- Pretend that you are taking this exam in West Hall Auditorium; therefore, all work on the exam **must** be your own; **do not even consider copying from other sources or communicating with others**
- **Any copying or collaborating with others will result in a grade of zero on the exam;** this includes posting in the Discussion Forum or other public or private forums

Sample problems

Use the previous sample "exam prep" questions provided for Exams 1 and 2. A few additional practice problems are on the pages that follow. Feel free to post your solutions in the Discussion Forum; and reply to posts if you agree or disagree with the proposed approaches/solutions. Some selected hints and solutions are shown on the next few pages.

1. What do the following system calls related to UDP do? What are their input arguments and return values?

   - `socket()`
   - `bind()`
   - `getsockname()`
   - `recvfrom()`
   - `sendto()`

   SOLUTION: Refer to the `man` pages and `udp-server.c`.

2. What do the following system calls related to TCP do? What are their input arguments and return values?

   - `socket()`
   - `bind()`
   - `listen()`
   - `accept()  <== (v1.2) added this important system call!`
   - `gethostbyname()`
   - `connect()`
   - `recv()`
   - `send()`

   SOLUTION: Refer to the `man` pages, `tcp-client.c`, and the various `tcp-server-*.c` examples.

3. Describe the different types of servers (e.g., iterative) and where they might be best used.

   SOLUTION: Refer to the `04-23-notes.txt` and the `04-23` lecture video.

4. Related to Question 3 above, what does the `select()` system call do? What are its input arguments and return value? How is `select()` useful?

   SOLUTION: Refer to the `man` page and `tcp-server-multiplex.c` example. In brief, `select()` allows you to monitor multiple file descriptors simultaneously, which therefore avoids blocking calls to `accept()`, `read()`, `recv()`, etc.

5. What are the key differences between UDP and TCP? Which one is more reliable?

   SOLUTION: UDP is connection-less, unreliable, and has very low overhead; conversely, TCP is connection-oriented with error-checking, resequencing, etc., which therefore adds overhead. UDP traffic consists of datagrams, while TCP traffic consists of packets sent over an established connection.

6. Related to Question 5, how does each endpoint "know" it has received all of the data sent?

   SOLUTION: Each layer of the seven-layer OSI Reference Model effectively communicates with the same layer at the remote endpoint. Within the various protocols, datagram/packet lengths and "checksums" help to ensure that all data is correctly received.

7. Why might `bind()` fail?

   SOLUTION: If a socket is already bound to a specific port number, calling `bind()` on another socket using the same port number will fail.

8. What is data marshalling and why is it important?

   SOLUTION: Data marshalling ensures that all data is sent in a uniform network format, which helps to ensure that endpoints properly interpret received data. Key examples here are integer data types `int` and `short`; regardless of the endianness on a given hardware architecture, integers sent across the network are sent in big endian format.

   Another example is a date, which could be sent in many formats (e.g., `MM/DD/YYYY`, `DD/MM/YYYY`, `YYYY/MM/DD`, etc.); the application protocol must have a well-defined date format.

9. Assume that the server program below has just started running. What is the **exact** terminal output of this server code if a client sends a UDP datagram containing the string "MEME" and then a few minutes later a different client sends a UDP datagram containing the string "ABCDEFGHIJKLMNOPQRSTUVWXYZ"?

   Assume all library function and system calls complete successfully. Note that this server does not terminate. Also note that in UDP, when a datagram is received, if the given buffer is not large enough, datagrams are simply truncated (i.e., bytes in the datagram beyond the size of the buffer are ignored).

```
#define MAXBUFFER 9

int main()
{
  char buffer[MAXBUFFER];
  int sd = socket( PF_INET, SOCK_DGRAM, 0 );
  struct sockaddr_in server;
  server.sin_family = PF_INET;
  server.sin_addr.s_addr = htonl( INADDR_ANY );
  server.sin_port = htons( 8128 );
  bind( sd, (struct sockaddr *) &server, sizeof( server ) );
  struct sockaddr_in client;
  int n, len = sizeof( client );

  while ( 1 )
  {
    n = recvfrom( sd, buffer, MAXBUFFER - 1, 0, (struct sockaddr *) &client,
                  (socklen_t *) &len );
    if ( n > 0 )
    {
      buffer[n] = '\0';
      fprintf( stderr, "Rcvd %d bytes: \"%s\"", n, buffer );
    }
    else return EXIT_FAILURE;
  }

  return EXIT_SUCCESS;
}
```

   SOLUTION: Exact terminal output of the server code is as follows:

```
Rcvd 4 bytes: "MEME"Rcvd 8 bytes: "ABCDEFGH"
```

   In this case, the string "IJKLMNOPQRSTUVWXYZ" is lost since it was in the UDP datagram but not read in the `recvfrom()` call.

Expand the code above to send a response datagram back to the client. The response datagram must contain an `unsigned short` value with the number of bytes received followed by the third character received. If the number of bytes originally received was less than three, then send `'\0'` instead.

SOLUTION: The `if` clause within the `while` loop is expanded as shown below in the `Q9_PART_B` section of the code.

```
  while ( 1 )
  {
    n = recvfrom( sd, buffer, MAXBUFFER - 1, 0, (struct sockaddr *) &client,
                  (socklen_t *) &len );
    if ( n > 0 )
    {
      buffer[n] = '\0';
      fprintf( stderr, "Rcvd %d bytes: \"%s\"", n, buffer );

#ifdef Q9_PART_B
      int datagram_size = sizeof( unsigned short ) + sizeof( char );
      void * data = calloc( 1, datagram_size );
      *(unsigned short *)data = htons( n );
      *(char *)( data + sizeof( unsigned short ) ) = ( n < 3 ? '\0' : buffer[2] );
      n = sendto( sd, data, datagram_size, 0, (struct sockaddr *) &client, len );
      if ( n == -1 ) perror( "sendto() failed" );
      free( data );
#endif
    }
    else return EXIT_FAILURE;
  }
```

10. Assume that the server program below has just started running. What is the **exact** terminal output of this server code if a client connects and sends two TCP packets, the first containing "`MEME`" and the second containing "`ABCDEFGHIJKLMNOPQRSTUVWXYZ`" (and then disconnects)? Assume all library function and system calls complete successfully.

```
#define MAXBUFFER 32

int main()
{
  int listener = socket( PF_INET, SOCK_STREAM, 0 );
  struct sockaddr_in server;
  server.sin_family = PF_INET;
  server.sin_addr.s_addr = htonl( INADDR_ANY );
  unsigned short port = 8123;
  server.sin_port = htons( port );
  int n, len = sizeof( server );
  bind( listener, (struct sockaddr *)&server, len );
  listen( listener, 5 );
  struct sockaddr_in client;
  int fromlen = sizeof( client );
  int newsd = accept( listener, (struct sockaddr *)&client,
                      (socklen_t *)&fromlen );

  do
  {
    char * buffer = calloc( MAXBUFFER, sizeof( char ) );
    n = recv( newsd, buffer, MAXBUFFER - 1, 0 );

    if ( n > 0 )
    {
      buffer[n] = buffer[4] = '\0';
      fprintf( stderr, "Rcvd %s%c-%s%s", buffer + 2, *(buffer + 1),
               buffer + 21, buffer + 30 );
    }
    free( buffer );
  }
  while ( n > 0 );

  close( newsd );
  return EXIT_SUCCESS;
}
```

SOLUTION: If we assume that the two packets are entirely read by two subsequent `recv()` calls, the output is shown below. (For the final exam and generally in practice, we assume this to be true for reasonably sized packets.)

```
Rcvd MEE-Rcvd CDB-VWXYZ
```

What happens if `MAXBUFFER` is set to 16 instead of 32?

SOLUTION: In this case, the entirety of each packet is read by the server through multiple `recv()` calls. Unlike UDP, for TCP, we are essentially guaranteed that all data is received on the remote end.

```
Rcvd MEE-Rcvd CDB-Rcvd RSQ-
```

(v1.2) Also, since the `calloc()` call only allocates 16 bytes, there could be garbage data after the `CDB-` and `RSQ-` output shown above.