15 Review

A non-comprehensive list of topics is below.

15.1 C

15.1.1 Memory and Strings

1. In the example below, which variables are guaranteed to print the value of zero?

```
int a;
static int b;

void func() {
  static int c;
  int d;
  printf("%d %d %d %d\n",a,b,c,d);
}
```

2. In the example below, which variables are guaranteed to print the value of zero?

```
void func() {
  int* ptr1 = malloc(sizeof(int));
  int* ptr2 = realloc(NULL, sizeof(int));
  int* ptr3 = calloc(1, sizeof(int));
  int* ptr4 = calloc(sizeof(int), 1);

  printf("%d %d %d %d\n",*ptr1,*ptr2,*ptr3,*ptr4);
}
```

3. Explain the error in the following attempt to copy a string.

```
char* copy(char*src) {
  char*result = malloc( strlen(src) );
  strcpy(result, src);
  return result;
}
```

4. Why does the following attempt to copy a string sometimes work and sometimes fail?

```
char* copy(char*src) {
  char*result = malloc( strlen(src) +1 );
  strcat(result, src);
  return result;
}
```

5. Explain the two errors in the following code that attempts to copy a string.

```
char* copy(char*src) {
  char result[sizeof(src)];
  strcpy(result, src);
  return result;
}
```

6. Which of the following is legal?

```
char a[] = "Hello"; strcpy(a, "World");
char b[] = "Hello"; strcpy(b, "World12345", b);
char* c = "Hello"; strcpy(c, "World");
```

7. Complete the function pointer typedef to declare a pointer to a function that takes a void* argument and returns a void*. Name your type 'pthread_callback'

```
typedef ______________________;
```

8. In addition to the function arguments what else is stored on a thread's stack?

9. Implement a version of `char* strcat(char*dest, const char*src)` using only `strcpy` `strlen` and pointer arithmetic

```
char* mystrcat(char*dest, const char*src) {

  ? Use strcpy strlen here

  return dest;
}
```

10. Implement version of size_t strlen(const char*) using a loop and no function calls.

```
size_t mystrlen(const char*s) {

}
```

11. Identify the three bugs in the following implementation of `strcpy`.

```
char* strcpy(const char* dest, const char* src) {
  while(*src) {*dest++ = *src++; }
  return dest;
}
```

15.1.2 Printing

1. Spot the two errors!

```
fprintf("You scored 100%");
```

2. Complete the following code to print to a file. Print the name, a comma and the score to the file 'result.txt'

```
char* name = .....;
int score = ......
FILE *f = fopen("result.txt",_____);
if(f) {
  _____
```

```
}
fclose(f);
```

3. How would you print the values of the variables `a`, `mesg`, `val` and `ptr` to a string? Print a as an integer, mesg as C string, val as a double val and ptr as a hexadecimal pointer. You may assume the mesg points to a short C string(<50 characters). Bonus: How would you make this code more robust or able to cope with?

```
char* toString(int a, char*mesg, double val, void* ptr) {
  char* result = malloc( strlen(mesg) + 50);
  -----
  return result;
}
```

15.1.3 Input parsing

1. Why should you check the return value of sscanf and scanf? ## Q 5.2 Why is 'gets' dangerous?
2. Write a complete program that uses `getline`. Ensure your program has no memory leaks.
3. When would you use calloc instead of malloc? When would realloc be useful?
4. What mistake did the programmer make in the following code? Is it possible to fix it
   i) using heap memory? ii) using global (static) memory?

```
static int id;

char* next_ticket() {
  id ++;
  char result[20];
  sprintf(result,"%d",id);
  return result;
}
```

15.2 Processes

1. What is a process?
2. What attributes are carried over from a process on fork? How about on a successful exec call?

3. What is a fork bomb? How can we avoid one?
4. What is the wait system call used for?
5. What is a zombie? How do we avoid them?
6. What is an orphan? What happens to them?
7. How do we check the status of a process that has exited?
8. What is a common pattern of processes?

15.3 Memory

1. What are the calls in C to allocate memory?
2. What must malloc memory be aligned to? Why is it important?
3. What is Knuth's Allocation Scheme?
4. How would you handle a request in a buddy allocation scheme?
5. What is a free list?
6. What are some different ways of inserting into a free list?
7. What are the benefits and drawbacks to first fit, worst fit, best fit?
8. When would a trivial malloc implementation

```
void *malloc(int size) {
  return (void *)sbrk(size);
}
```

   Be acceptable?

15.4 Threading and Synchronization

1. What is a thread? What do threads share?
2. How does one create a thread?
3. Where are the stacks for a thread located in memory?
4. What is a mutex? What problem does it solve?
5. What is a condition variable? What problem does it solve?

6. Write a thread safe linked list that supports insert front, back, pop front, and pop back. Make sure it doesn't busy wait!

7. What is Peterson's Solution to the critical section problem? How about Dekker's?

8. Is the following code thread-safe? Redesign the following code to be thread-safe. Hint: A mutex is unnecessary if the message memory is unique to each call.

```
static char message[20];
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *format(int v) {
  pthread_mutex_lock(&mutex);
  sprintf(message, ":%d:" ,v);
  pthread_mutex_unlock(&mutex);
  return message;
}
```

9. Which one of the following may leave a process in running?

   (a) Returning from the pthread's starting function in the last running thread.

   (b) The original thread returning from main.

   (c) Any thread causing a segmentation fault.

   (d) Any thread calling **exit**.

   (e) Calling **pthread_exit** in the main thread with other threads still running.

10. Write a mathematical expression for the number of "W" characters that will be printed by the following program. Assume a,b,c,d are small positive integers. Your answer may use a 'min' function that returns its lowest valued argument.

```
unsigned int a=...,b=...,c=...,d=...;

void* func(void* ptr) {
  char m = * (char*)ptr;
  if(m == 'P') sem_post(s);
  if(m == 'W') sem_wait(s);
  putchar(m);
  return NULL;
}

int main(int argv, char** argc) {
  sem_init(s,0, a);
  while(b--) pthread_create(&tid, NULL, func, "W");
  while(c--) pthread_create(&tid, NULL, func, "P");
  while(d--) pthread_create(&tid, NULL, func, "W");
```

```
  pthread_exit(NULL);
  /*Process will finish when all threads have exited */
}
```

11. Complete the following code. The following code is supposed to print alternating `A` and `B`. It represents two threads that take turns to execute. Add condition variable calls to `func` so that the waiting thread need not to continually check the `turn` variable. Q: Is pthread_cond_broadcast necessary or is pthread_cond_signal sufficient?

```
pthread_cond_t cv = PTHREAD_COND_INITIALIZER;
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;

void* turn;

void* func(void* mesg) {
  while(1) {
    // Add mutex lock and condition variable calls ...

    while(turn == mesg) {
      /* poll again ... Change me - This busy loop burns CPU time!
          */
    }

    /* Do stuff on this thread */
    puts( (char*) mesg);
    turn = mesg;

  }
  return 0;
}

int main(int argc, char** argv){
  pthread_t tid1;
  pthread_create(&tid1, NULL, func, "A");
  func("B"); // no need to create another thread - use the main
      thread
  return 0;
}
```

12. Identify the critical sections in the given code. Add mutex locking to make the code thread safe. Add condition variable calls so that `total` never becomes negative or above 1000. Instead the call should block until it is safe to proceed. Explain why `pthread_cond_broadcast` is necessary.

```
int total;
void add(int value) {
  if(value < 1) return;
  total += value;
}
void sub(int value) {
  if(value < 1) return;
  total -= value;
}
```

13. An thread unsafe data structure has **`size()`** **`enq`** and **`deq`** methods. Use condition variable and mutex lock to complete the thread-safe, blocking versions.

```
void enqueue(void* data) {
  // should block if the size() would become greater than 256
  enq(data);
}
void* dequeue() {
  // should block if size() is 0
  return deq();
}
```

14. Your startup offers path planning using the latest traffic information. Your overpaid intern has created a thread unsafe data structure with two functions: **`shortest`** (which uses but does not modify the graph) and **`set_edge`** (which modifies the graph).

```
graph_t* create_graph(char* filename); // called once

// returns a new heap object that is the shortest path from vertex
    i to j
path_t* shortest(graph_t* graph, int i, int j);

// updates edge from vertex i to j
void set_edge(graph_t* graph, int i, int j, double time);
```

For performance, multiple threads must be able to call **`shortest`** at the same time but the graph can only be modified by one thread when no threads other are executing inside **`shortest`** or **`set_edge`**.

15. Use mutex lock and condition variables to implement a reader-writer solution. An incomplete attempt is shown below. Though this attempt is thread safe (thus sufficient for demo day!), it does not allow multiple threads to calculate **`shortest`** path at the same time and will not have sufficient throughput.

```
path_t* shortest_safe(graph_t* graph, int i, int j) {
  pthread_mutex_lock(&m);
  path_t* path = shortest(graph, i, j);
  pthread_mutex_unlock(&m);
  return path;
}
void set_edge_safe(graph_t* graph, int i, int j, double dist) {
  pthread_mutex_lock(&m);
  set_edge(graph, i, j, dist);
  pthread_mutex_unlock(&m);
}
```

16. How many of the following statements are true for the reader-writer problem?

    - There can be multiple active readers
    - There can be multiple active writers
    - When there is an active writer the number of active readers must be zero
    - If there is an active reader the number of active writers must be zero
    - A writer must wait until the current active readers have finished

15.5 Deadlock

1. What do each of the Coffman conditions and what do they mean? Can you provide a definition of each one and an example of breaking them using mutexes?
2. Give a real life example of breaking each Coffman condition in turn. A situation to consider: Painters, paint and paint brushes.
   (a) Hold and wait
   (b) Circular wait
   (c) No preemption
   (d) Mutual exclusion
3. Identify when Dining Philosophers code causes a deadlock (or not). For example, if you saw the following code snippet which Coffman condition is not satisfied?

```
// Get both locks or none.
pthread_mutex_lock( a );
if( pthread_mutex_trylock( b ) ) {/*failed*/
  pthread_mutex_unlock( a );
  ...
}
```

4. How many processes are blocked?

   - P1 acquires R1
   - P2 acquires R2
   - P1 acquires R3
   - P2 waits for R3
   - P3 acquires R5
   - P1 acquires R4
   - P3 waits for R1
   - P4 waits for R5
   - P5 waits for R1

5. What are the pros and cons for the following solutions to dining philosophers

   (a) Arbitrator

   (b) Dijkstra

   (c) Stalling's

   (d) Trylock

15.6 IPC

1. What are the following and what is their purpose?

   (a) Translation Lookaside Buffer

   (b) Physical Address

   (c) Memory Management Unit

   (d) The dirty bit

2. How do you determine how many bits are used in the page offset?

3. 20 ms after a context switch the TLB contains all logical addresses used by your numerical code which performs main memory access 100% of the time. What is the overhead (slowdown) of a two-level page table compared to a single-level page table?

4. Explain why the TLB must be flushed when a context switch occurs (i.e. the CPU is assigned to work on a different process).

5. Fill in the blanks to make the following program print 123456789. If `cat` is given no arguments it simply prints its input until EOF. Bonus: Explain why the `close` call below is necessary.

```
int main() {
  int i = 0;
  while(++i < 10) {
    pid_t pid = fork();
```

```
    if(pid == 0) {/* child */
      char buffer[16];
      sprintf(buffer, ______,i);
      int fds[ ______];
      pipe(fds);
      write(fds[1], ______,______ ); // Write the buffer into the
          pipe
      close(______);
      dup2(fds[0], ______);
      execlp("cat", "cat", ______);
      perror("exec"); exit(1);
    }
    waitpid(pid, NULL, 0);
  }
  return 0;
}
```

6. Use POSIX calls `fork` `pipe` `dup2` and `close` to implement an autograding program. Capture the standard output of a child process into a pipe. The child process should `exec` the program `./test` with no additional arguments (other than the process name). In the parent process read from the pipe: Exit the parent process as soon as the captured output contains the ! character. Before exiting the parent process send SIGKILL to the child process. Exit 0 if the output contained a !. Otherwise if the child process exits causing the pipe write end to be closed, then exit with a value of 1. Be sure to close the unused ends of the pipe in the parent and child process

7. This advanced challenge uses pipes to get an "AI player" to play itself until the game is complete. The program `tic tac toe` accepts a line of input - the sequence of turns made so far, prints the same sequence followed by another turn, and then exits. A turn is specified using two characters. For example "A1" and "C3" are two opposite corner positions. The string `B2A1A3` is a game of 3 turns/plys. A valid response is `B2A1A3C1` (the C1 response blocks the diagonal B2 A3 threat). The output line may also include a suffix `-I win` `-You win` `-invalid` or `-draw` Use pipes to control the input and output of each child process created. When the output contains a `-`, print the final output line (the entire game sequence and the result) and exit.

8. Write a function that uses fseek and ftell to replace the middle character of a file with an 'X'

```
void xout(char* filename) {
  FILE *f = fopen(filename, ____ );

  // Your code here ...
}
```

9. What is an MMU? What are the drawbacks to using it versus a direct memory system?

10. What is a pipe?

11. What are the pros and cons between named and unnamed pipes?

15.7 Filesystems

1. What is the file API?
2. Where are the names of the files stored?
3. What is contained in an inode?
4. What are the two special file names in every directory
5. How do you resolve the following path `a/../b/./c/../../c`
6. What are the rwx groups?
7. What is an UID? GID? What is the difference between UID and Effective UID?
8. What is umask?
9. What is the sticky bit?
10. What is a virtual file system?
11. What is RAID?
12. In an `ext2` filesystem how many inodes are read from disk to access the first byte of the file `/dir1/subdirA/notes.txt` ? Assume the directory names and inode numbers in the root directory (but not the inodes themselves) are already in memory.
13. In an `ext2` filesystem what is the minimum number of disk blocks that must be read from disk to access the first byte of the file `/dir1/subdirA/notes.txt` ? Assume the directory names and inode numbers in the root directory and all inodes are already in memory.
14. In an `ext2` filesystem with 32 bit addresses and 4KiB disk blocks, an inode can store 10 direct disk block numbers. What is the minimum file size required to require a single indirection table? ii) a double direction table?
15. Fix the shell command `chmod` below to set the permission of a file `secret.txt` so that the owner can read,write,and execute permissions the group can read and everyone else has no access.

```
$ chmod 000 secret.txt
```

15.8 Networking

1. What is a socket?

2. What are the different layers of the internet?
3. What is IP? What is an IP address?
4. What is TCP? What is UDP? What are the differences?
5. Create a TCP client that send “Hello” to a server.
6. Create a simple TCP echo server. This is a server that reads bytes from a client until it closes and echoes the bytes back to the client.
7. Create a UDP client that would send a flood of packets to a hostname at argv[1].
8. What is HTTP?
9. What is DNS?
10. Why do we use non-blocking IO for networking?
11. What is an RPC?
12. What is special about listening on port 1000 vs port 2000?
    - Port 2000 is twice as slow as port 1000
    - Port 2000 is twice as fast as port 1000
    - Port 1000 requires root privileges
    - Nothing
13. Describe one significant difference between IPv4 and IPv6?
14. When and why would you use ntohs?
15. If a host address is 32 bits which IP scheme am I most likely using? 128 bits?
16. Which common network protocol is packet based and may not successfully deliver the data?
17. Which common protocol is stream-based and will resend data if packets are lost?
18. What is the SYN ACK ACK-SYN handshake?
19. Which one of the following is NOT a feature of TCP?
    (a) Packet reordering
    (b) Flow control
    (c) Packet retranmission
    (d) Simple error detection
    (e) Encryption
20. What protocol uses sequence numbers? What is their initial value? And why?
21. What are the minimum network calls are required to build a TCP server? What is their correct order?
22. What are the minimum network calls are required to build a TCP client? What is their correct order?
23. When would you call bind on a TCP client?

24. What is the purpose of socket bind listen accept ?
25. Which of the above calls can block, waiting for a new client to connect?
26. What is DNS? What does it do for you? Which of the CS241 network calls will use it for you?
27. For getaddrinfo, how do you specify a server socket?
28. Why may getaddrinfo generate network packets?
29. Which network call specifies the size of the allowed backlog?
30. Which network call returns a new file descriptor?
31. When are passive sockets used?
32. When is epoll a better choice than select? When is select a better choice than epoll?
33. Will `write(fd, data, 5000)` always send 5000 bytes of data? When can it fail?
34. How does Network Address Translation (NAT) work?
35. Assuming a network has a 20ms One Way Transit Time between Client and Server, how much time would it take to establish a TCP Connection?
    (a) 20ms
    (b) 40ms
    (c) 100ms
    (d) 60ms
36. What are some of the differences between HTTP 1.0 and HTTP 1.1? How many ms will it take to transmit 3 files from server to client if the network has a 20ms transmit time? How does the time taken differ between HTTP 1.0 and HTTP 1.1?
37. Writing to a network socket may not send all of the bytes and may be interrupted due to a signal. Check the return value of `write` to implement `write_all` that will repeatedly call `write` with any remaining data. If `write` returns -1 then immediately return -1 unless the `errno` is `EINTR` - in which case repeat the last `write` attempt. You will need to use pointer arithmetic.

```
// Returns -1 if write fails (unless EINTR in which case it recalls
    write
// Repeated calls write until all of the buffer is written.
ssize_t write_all(int fd, const char *buf, size_t nbyte) {
  ssize_t nb = write(fd, buf, nbyte);
  return nb;
}
```

38. Implement a multithreaded TCP server that listens on port 2000. Each thread should read 128 bytes from the client file descriptor and echo it back to the client, before closing the connection and ending the thread.
39. Implement a UDP server that listens on port 2000. Reserve a buffer of 200 bytes. Listen for an arriving packet. Valid packets are 200 bytes or less and start with four bytes 0x65 0x66 0x67 0x68. Ignore invalid packets. For valid packets add the value of the fifth byte as an unsigned value to a running total and print the total so far. If the running total is greater than 255 then exit.

15.9 Security

1. What are the three measures for data security?
2. What is stack smashing?
3. What is buffer overflows?
4. How does an operating system provide security? What are some examples from Networking and Filesystems?
5. What security features does TCP provide?
6. Is DNS secure?

15.10 Signals

1. Give the names of two signals that are normally generated by the kernel
2. Give the name of a signal that can not be caught by a signal
3. Why is it unsafe to call any function (something that it is not signal handler safe) in a signal handler?
4. Write brief code that uses SIGACTION and a SIGNALSET to create a SIGALRM handler.
5. What is the difference between a disposition, mask, and pending signal set?
6. What attributes are passed over to process children? How about exececuted processes?