CS241 Final Exam Fall 2019 (3 hours)

- You may not leave with any exam materials. You must hand in this question set, your Scantron form, and your Free-Response sheet before you leave. Do not disrupt others if you wish to leave early.
- The instructor/TAs/proctors will not answer any questions about questions in this exam. Exam typos or ambiguities will be reviewed after the exam has finished. Write down your interpretation or assumptions and answer accordingly.
- This exam contains 5 coding questions that you will answer on the Free-Response sheet (not in this booklet) and 46 Scantron multiple choice problems. Plan your time accordingly.
- A list of common C library calls and system call declarations and a man page are included at the end of this exam. You may tear off these pages but must turn them in when you leave. You may write shortened versions of types, calls and initializers (e.g. `p_m_lock`), provided the grader can unambiguously interpret your code as a typical programming item studied in CS241.
- Read the given code carefully and be aware it may contain deliberate errors that are common in C programs. You may use C99 syntax (e.g. declaring variables inside a `for` loop).
- For given code, you may assume code examples are wrapped inside a main() function when necessary and the necessary header file includes are written but not shown. You do not need to specify include files for your own written code.
- Specific questions may be dropped from scoring based on a statistical analysis *after* the student response data has been fully analyzed.
- No books, notes, calculators, or electronic devices (e-watches) are allowed. **These must off and inaccessible.**
- Do not turn this page until instructed to.
- At the end of the exam time, there is a 20% per minute penalty if you do not stop when instructed to do so. If you spoil your scantron you may transcribe your answers from your old scantron to a new sheet at the front of the room after the exam has finished.
- There are several different versions of this exam (this is version 0). Be sure to correctly mark the answer key below on your Scantron form.
- Cheating or appearing to cheat (e.g. attempting to copy from another student, using notes, ignoring proctor instructions) is taken seriously by the University and are grounds for dismissal from UIUC. Proctors will ensure that the exam is fair for everyone.
- This is a closed book and closed notes exam.

1. Fill in your information:

Full Name (first last): _____ _____

UIN (Student Number 6...): _____

NetID: _____ **@illinois.edu**

2. Use a pencil to bubble in the course, your name, NetID and UIN on Scantron form PLUS the following answer key. Verify that you have copied the key into the correct scantron responses.

Zone 1

Free-Response FR1 (12 points)

Write your answers on the Free Response Sheet,"FR1" to implement the function `readall`. Responses on this sheet will not be graded. You may use C99 syntax. Implement the function,

```
void *readall(int fd, size_t N, size_t *bytes);
```

that reads up to N bytes from the given file descriptor `fd`. Return a pointer to the start of the buffer of size `N`, which your code will allocate on the heap. You may assume your heap allocation succeeds. Your code will set the memory at `bytes` to be the actual total number of bytes read.

Use `ssize_t read(int fd, void *buf, size_t count)` to read bytes from the file descriptor. Implement the following conditions:

- `read` may return 0 (end of file) before N bytes are read. In this case, return the buffer and set the value pointed to by `bytes` to be the total number of bytes read into the buffer.
- `read` may return -1. If the error is `EINTR` then restart the previous read. For all other errors, 1) unallocate the buffer memory and 2) return `NULL`.
- `read` may return a value less than N. Use pointer arithmetic and repeated `read` calls to ensure all `N` bytes are read.

Zone 2

Free-Response FR2 (15 points)

Fill in the blanks! Write your answers on the Free Response Sheet, "FR2" to complete the following code - Written responses on this sheet will not be graded. The completed code will perform the following:

Run the executable `"./foobar"` with no additional arguments, The executable's `stdin` will originate from the file `"mydata"`, and its `stdout` will read by the parent process (your code). You may assume the given `read` call below succeeds.

```
int fds[  a._____   ];

b._____ ;  // create communication mechanism between child and parent

int pid =  c._____;    // start a new process

if (pid  d._____   ) {

  printf("Failed to create a new process\n");

} else if (pid  e._____   ) {

  // This is the parent process.

  // read up to 100 bytes of output from the child process
  char buf[  f._____  ];
  size_t b = read(  g._____  , buf, 100); // assume b>=0
  buf[b] = '\0';    // terminate the string
  printf("foobar output: %s\n", buf);

  // Wait for the child to finish.
  h._____;

} else {

  // open the data file for reading
  int file_fd = i._____;

  // set stdin using one or two function calls
  j._____;


  // set stdout using one or two function calls
  k._____;


  // run "./foobar" with argv[0] as "./foobar" and no other arguments
  l._____;


  fprintf(stderr, "failed to run foobar\n");
}
```

Responses on this sheet will not be graded. Use the Free-Response sheet.

Zone 3

Free-Response FR3 (12 points)

Complete the following function to read the entire contents of a file (specified by the parameter) into memory. You may assume the arguments are not `NULL`.
Use `stat` to determine the file size and check that (i) the file exists and (ii) is not a directory.
If it is a directory or does not exist, return `NULL` and leave the value at `length` unmodified.
Write the file size into the memory pointed at by the `length` argument.
Return the starting address of the file contents.
You may read the file contents into memory using `mmap`, `read` or `fread`.

For `read` and `fread` reserve exactly the amount of memory you require on the heap. You do not need to implement error handling (other than checking to see if the file is not present or is a directory) or close the file handle / file descriptor.
Hint: See the declarations at the end of this exam for useful calls, macros and struct declarations.

```
void *get_contents(char *filename, off_t *length) {
```

Responses on this sheet will not be graded. Use the Free-Response sheet.

Zone 4

Free-Response FR4 (10 points)

This is not a Scantron question–use the Free Response Sheet "FR4". Written responses here will not be graded. Consider an operating system with a single-level page table. Complete the following function to translate a virtual to a physical memory address. The arguments are `va`, the virtual address, and `pt`, the page table. The page table is an array of page table entries with no status bits at all. That is, the i^{th} entry in the page table is the memory address in physical memory of the start of the i^{th} page of virtual memory. The number of bits required to store an offset is stored in the variable `PAGE_OFFSET_BITS`. For example, if the page size is `4KB`, then `PAGE_OFFSET_BITS = 12`.

For example, if the 5^{th} virtual page is located at physical addresses 0x8000 to 0x8fff, the value contained in the `pt[4]` would be `0x8000` (the start of the physical address).

```
void *translate_address(void *va, void **pt) {
  long pt_index;
  void *pte = NULL;

  (Line 1): _____________________ YOUR ANSWERS MUST BE ON THE

  (Line 2): _____________________ FREE RESPONSE SHEET

  void *offset = va - (pt_index << PAGE_OFFSET_BITS);

  (Line 3): _____________________ YOUR RESPONSES ON THIS

  (Line 4): _____________________ PAGE WILL NOT

  (Line 5): _____________________ BE GRADED
}
```

From the following lines of code, choose the line of code that best completes the functionality described above. Note that it may be necessary to use one line multiple times. Choose option `X` if no code is needed. Enter the letter corresponding to your answers on the Free-Response sheet.

```
A. pt_index = va + pt;
B. pt_index = va - pt;
C. pt_index = va & pt; /* Bit-wise and  e.g. 3 & 6 == 2 */
D. pt_index = va ^ pt; /* Bit-wise exclusive-or  e.g. 3 ^ 6 == 5 */
E. pt_index = va / PAGE_OFFSET_BITS;
F. pt_index = va * PAGE_OFFSET_BITS;
G. pt_index = va >> PAGE_OFFSET_BITS; /* Shift right e.g. 0x123 >> 4 == 0x12 */
H. pt_index = va << PAGE_OFFSET_BITS; /* Shift left e.g. 0x123 << 4 == 0x1230 */
I. pte = pt[PAGE_OFFSET_BITS];
J. pte = pt[pt_index];
K. pte = pt[va];
L. pte = pt[offset];
M. pte++;
N. pte--;
O. pte = pt[pt_index * PAGE_OFFSET_BITS];
P. pte = pt[va * PAGE_OFFSET_BITS];
Q. pte = pt[offset * PAGE_OFFSET_BITS];
R. return pt_index;
S. return pte;
T. return offset;
U. return pte + pt_index;
V. return pte + offset;
W. return pt_index + offset;
X. /* No code needed on this line. */
```

Zone 5

Free-Response FR5 (12 points)

Write your answers on the Free Response Sheet "FR5". Responses on this page will not be graded. This is the last free response question. Use the Scantron for the remaining 46 questions.

To test the security of the state's electronic voting machine, you attempt to install a tiny, long running single-threaded server program named `secret` that can secretly execute arbitrary commands. Two functions are already provided (you may call them, and do not have to implement them):

```
/* Returns the file descriptor of a new passive server socket that is already
   bound to a local TCP port, has a backlog of 10 and is ready to be used to
   accept new client connections. Only call init_server one time. */
int init_server(char **argv);

/* Parse cmdline and call execvp. This function will never return:
   it will call exit(1) if execvp fails. */
void myexec(const char *cmdline);
```

Your server will open a socket and wait for TCP connections. Each client that connects will send one command to your server, and your server will execute the command if it is in the valid format described below.

Do not start up a new thread to handle each connection; process them one at a time in the current thread. Your code should not perform any other actions other than ones specified here e.g. unspecified error handling and logging are unnecessary.

You may assume that any command sent to the server can be successfully read with a single call to `read()`. When a new client connection is accepted, perform a blocking read. Store the incoming data in a buffer of size 2048 bytes. The client will not send a terminating 0 byte. Always close the client connection file descriptor before processing the command.

Use the return value of `read()` to determine the number of bytes that have arrived. If the request is at least 3 bytes and the first byte is a `[` ASCII character and the last byte is a `]` ASCII character then the request is valid and the characters between `[` and `]` represent the command line to be executed.

Invalid commands also include the case where `read` returns a non-positive value. Invalid commands must be safely ignored; your program will continue to accept and process new client connections.

An example valid client request is "`[rm /etc/hosts]`" (with no terminating 0 byte), and it will cause the command `rm /etc/hosts` to be run.

Your code will run forever – after handling a client request, closing the client connection (in all cases), loop back and accept another client.

Zone 6

Multiple choice section

Fill in your answers for this section on the Scantron form.

1/1. (3 points) Choose the best response. The signal `SIGCHLD` is an example of ...

A. A signal that is generated by a process and can be caught by the target process's chosen signal handler
B. A signal that is typically generated by a process, delivered to the kernel, and cannot be caught by a signal handler
C. A signal that is generated by the kernel and can be caught by the target process's chosen signal handler
D. A signal that is used internally by the kernel and never delivered to a target process
E. A signal that is delivered to a network interface when there is a connection error

2/1. (3 points) Choose the best response. The signal `SIGKILL` is an example of ...

A. A signal that is generated by a process and can be caught by the target process's chosen signal handler
B. A signal that is typically generated by a process, delivered to the kernel, and cannot be caught by a signal handler
C. A signal that is generated by the kernel and can be caught by the target process's chosen signal handler
D. A signal that is used internally by the kernel and never delivered to a target process
E. A signal that is delivered to a network interface when there is a connection error

3/1. (3 points) Riddle me this! I find new memory for you but won't stop until I find the smallest piece that's at least as large as the size you're looking for! I am:

A. Impossible fit allocator
B. First fit allocator
C. Buddy allocator
D. Best fit allocator
E. Worst fit allocator

4/1. (3 points) I have a file `good.txt`. How do I change the permissions of my file such that:

```
The owner (me) has full permission: Read, write, and execute
The group has read and write permissions
Everyone else has read permission
```

A. chmod 764 good.txt
B. chown rwx-rw–r good.txt
C. chown 671 good.txt
D. perm good.txt 317
E. perm good.txt rwx-rw–r

5/1. (3 points) Which of the following lines is most likely to crash at runtime or cause a compile-time error?

```
1. int* f(void) {
2.     int* x = (int*) malloc (10000 * sizeof(int));
3.     return & x[5];
4. }
5. int main(int argc, char** argv) {
6.     int* p = f();
7.     int y = *p;
8.     free(p);
9. }
```

A. Line 2
B. Line 3
C. Line 6
D. Line 7
E. Line 8

6/1. (3 points) In the code below, suppose that when malloc is called, the *system has no more memory to allocate.* In this case, which of the lines is most likely to crash the program (terminating the program, not just returning an error)?

```
1. int* f(void) {
2.     int* x = (int*) malloc (100000 * sizeof(int));
```

```
3.     return x + 5;
4. }
5. int main(int argc, char** argv) {
6.     int* p = f();
7.     int y = *p;
8.     free(p);
9. }
```

A. Line 2
B. Line 3
C. Line 6
D. Line 7
E. Line 8

7/1. (3 points) Suppose Program A prints out the virtual memory address of one of its variables, and Program B reads this address value. Typically, well-written programs do not share their virtual memory addresses in this way. Why not?

A. Program B can use this address to illegally access private data of Program A.
B. Program B can use this address to access data of Program A, which may cause a problem if the data is not protected with a synchronization primitive.
C. The virtual memory address has no meaning in Program B's memory address space and is essentially a garbage pointer.
D. This is not possible to implement: when Program A tries to print out an address, it will be translated to a physical address, and is not a virtual address.

8/1. (3 points) A memory address is split up into a page number and a page offset. On a 32-bit byte-addressable machine where each page contains 4 KB of data, how many bits are used for the page number?

A. 8
B. 12
C. 16
D. 20
E. 28

8/2. (3 points) A memory address is split up into a page number and a page offset. On a 32-bit byte-addressable machine where each page contains 2 KB of data, how many bits are used for the page offset?

A. 10
B. 12
C. 21
D. 11
E. 16

8/3. (3 points) A memory address is split up into a page number and a page offset. On a 64-bit byte-addressable machine where each page contains 8 KB of data, how many bits are used for the page number?

A. 13
B. 8
C. 56
D. 51
E. 64

9/1. (3 points) Consider the following functions. Recall that `strcpy(dest, s2)` copies the string `s2` into `dest`.

```
char *func_1(char *fname) {
  char * buffer = (char *)malloc(strlen(fname));
  strcpy(buffer, fname);
  return buffer;
}
char *func_2(char *fname) {
  char buffer[1024];
  strcpy(buffer, fname);
```

```
  return buffer;
}
char *func_3(char *fname) {
  char * buffer = (char *)malloc(sizeof(fname) + 1);
  strcpy(buffer, fname);
  return buffer;
}
```

When used as part of a larger program, which of the functions above use memory correctly?

A. Only `func_1`

B. Only `func_2`

C. Only `func_3`

D. Only `func_1` and `func_2`

E. All three functions use memory incorrectly

10/1. (3 points) Solve my poetic riddle! There once was a process named Thor who tore through big data galore. Now one day an `alarm()` sent him possible harm

A. But he blocked it with a semaphore

B. But he'd signal()'d a handler before

C. He segfaulted and promptly dumped core

D. So he utilized the standard POSIX syscall to return a process to running state after it is terminated, `fork_restore()`

11/1. (3 points) A worker thread with id `tid` calls `pthread_exit(x)` where `x` is a pointer to a buffer allocated on the heap. Which one of the following is true?

A. The main thread can get the buffer pointer using `pthread_join(tid,&ptr)`.

B. When the thread exits, all heap memory is reclaimed; the buffer is invalid.

C. Other threads need to use `pthread_get_attr()` to read from the buffer.

D. The process will exit if the x value is `NULL`.

E. Only the main thread may call `pthread_exit`.

12/1. (3 points) In the code below, function `foo()` has a stack variable `x` and calls the function worker with argument &x as the entry point to a new thread. Once `worker()` starts running, is its argument `arg` a valid memory pointer?

```
void* worker(void* arg) {
    int y;
    ...
    return arg;
}
void foo(void){
    char x = 'A';
    void* result;
    pthread_t tid;
    pthread_create(&tid, 0, worker, (void*)&x);
    pthread_join(tid, (void*) &result);
}
```

A. Yes

B. No

C. Maybe: may depend on nondeterministic events at runtime

D. African or European swallow?

E. This is not the response you are looking for

12/2. (3 points) Function `worker()` has a stack variable `y` and returns &y. After `foo()` starts the thread, it calls `pthread_join()` to collect the return value of `worker()`. At this point, is the return value of `worker()` a valid memory pointer?

```
void* worker(void* arg) {
    int y;
    ...
```

```
    return &y;
}
void foo(void){
    char x='A';
    void* result;
    pthread_t tid;
    pthread_create(&tid, 0, worker, (void*)&x);
    pthread_join(tid,&result);
}
```

A. Yes
B. No
C. Maybe: may depend on nondeterministic events at runtime
D. Only when compiled using a C99 compiler
E. Only when compiled using a C89 compiler

13/1. (3 points) Choose the most complete answer: When reassigning a CPU from one process to another, the OS must always...

A. change some register values
B. change some register values and update the page table in the MMU
C. change some register values, update the page table in the MMU, and write pages to disk
D. change some register values, update the page table in the MMU, write pages to disk, and put the old process on the "blocked" queue

14/1. (3 points) What is "C.I.A." from a secure development practices point of view?

A. Protect the Confidentiality, Integrity and Availability of the data and service.
B. Security-aware code reviews should be Complete, Intelligent and Audacious.
C. Security automated tests should be Comprehensive, Intensive and Accuate.
D. roduction features - Complete, Debug features - Inoperative, and Documentation - Accurate.

15/1. (3 points) Consider pseudo-code executed potentially multiple times by two different threads, Thread 1 and Thread 2, around a critical section. In all cases, assume that all variables are initialized to zero when the threads start. In answering the question, you should consider all possible scenarios including scenarios where a thread executes the code more than once (sequentially) and scenarios where only one of the two threads remains, while the other has terminated. Which of the following statements is true about the code segment below?

```
Thread 1 :
   while (!done) {
      while (x > 0) { /*repeat*/ }
      atomic increment x;
      CRITICAL SECTION
      atomic decrement x;

      // Do other computation for a random amount of time
   }

Thread 2 :
   while (!done) {
      while (x > 0) { /*repeat*/ }
      atomic increment x;
      CRITICAL SECTION
      atomic decrement x;

      // Do other computation for a random amount of time
   }
```

A. Mutual Exclusion and Progress are both NOT guaranteed
B. Mutual Exclusion is guaranteed but Progress is NOT guaranteed

C. Mutual Exclusion is NOT guaranteed but Progress is guaranteed

D. Mutual Exclusion and Progress are both guaranteed

16/1. (3 points) What are Meltdown, Spectre and Heartbleed?

A. These are vulnerabilities that allow an unauthorized access to confidential information.

B. These are all examples of ssl protocol vulnerabilities.

C. These are all examples of CPU vulnerabilities that affect the integrity of the data.

D. These are all examples of network vulnerabilities that affect the availability of the data.

E. None of the other responses are correct.

17/1. (3 points) A resource allocation graph is typically used for which purpose by the operating system?

A. Create deadlocks for test purposes

B. Find or minimize deadlocks

C. Allocate TCP server graphs

D. Allocate UDP server graphs

E. Allocate shared memory

18/1. (3 points) What term is used to describe an error condition particular to concurrent programming where the results of a multi-threaded application change as the relative schedule (sequence of operations) of different threads varies?

A. Memory Leak

B. Race Condition

C. Loose Semaphore

D. Dining Philosopher Problem

E. Deadlock

19/1. (3 points) Suppose, for some system, an ordering of resources is defined and all processes/threads acquire resources in the defined order. Would deadlock be avoided?

A. Yes

B. No

20/1. (3 points) Choose the most complete answer: When reassigning a CPU from one thread to another thread in the same process, the OS must always...

A. change some register values

B. change some register values and update the page table in the memory management unit (MMU)

C. change some register values, update the page table in the MMU, and write pages to disk

D. change some register values, update the page table in the MMU, write pages to disk, and put the old thread on the "blocked" queue

21/1. (3 points) Two threads call `pthread_mutex_lock` on the same mutex. Which one of the following best describes what happens next?

A. The result is undefined

B. The mutex lock is increased by two

C. The mutex lock is decreased by two

D. One thread will continue, the other thread must wait until the mutex is unlocked

22/1. (3 points) Assume in the Dining Philosophers problem, each philosopher repeatedly thinks for some length of time and then decides to eat. To eat, they attempt to lock the left chopstick, then lock the right chopstick, eat, then unlock in the reverse order (right, then left). With N philosophers and N plates sitting around a circular table this can cause deadlock. Which of the following changes are sufficient to eliminate **deadlock** ?

```
1. Having N seats (each including a chopstick) and N/2 philosophers.
2. Having N seats (each including a chopstick) and N-1 philosophers.
3. If the right chopstick is locked, put the left chopstick down and go back to thinking
```

A. Only change 1 would prevent deadlock

B. Only change 2 would prevent deadlock

C. Only change 3 would prevent deadlock

D. Any of the changes would prevent deadlock

E. Only changes 1 and 3 would prevent deadlock

23/1. (3 points) Which of the following scheduling policies is likely to achieve the best response time?

A. Non-preemptive First Come First Serve (FCFS)

B. Non-preemptive priority scheduling

C. Non-preemptive shortest job first

D. Round Robin with a large quantum

E. Round Robin with a small quantum

24/1. (3 points) Which response best describes stream interface(s) (similar to a TCP connection) that can be used for IPC?

A. pipes

B. mmap

C. signals

D. fstreams

E. All of the other responses are correct

25/1. (3 points) When the code is stuck in the infinite loop at Line 7, the user presses `Ctrl+C` three times in attempt to kill the program. What best describes what happens?

```
01 void handler(int sig) {
02     write(1,"No!",3);
03 }
04
05 void main() {
06     signal(SIGINT, handler);
07     while (1) { }
08 }
```

A. The program does nothing.

B. The program exits due to a segmentation fault caused by `Ctrl+C`.

C. The program exits normally, the `Ctrl+C` breaks the code out of an infinite loop.

D. The program prints out `No!` exactly three times, and keeps looping.

E. The program prints out `No!` one to three times, and keeps looping.

26/1. (3 points) Each machine only has a limited number of network port numbers that are shared globally between all users of the machine. Which one of the following is true about ports?

```
1. The port numbers are limited to 65,536 on EWS machines only because they are 64-bit machines.
2. All ports are equivalent and any user on any machine may access any port.
3. Ports below 1024 are reserved for well-established protocols run by users with root privileges.
```

A. Only statements 1 and 2 are true

B. Only statements 1 and 3 are true

C. Only statement 3 is true

D. Only statement 2 is true

E. Only statement 1 is true

27/1. (3 points) Which of the following statements about HTTP is FALSE?

A. HTTP can be used on top of a TCP connection.

B. Only one HTTP request can be sent per connection using non-persistent HTTP.

C. An HTTP client can store user-specific information in the form of cookies.

D. GET is a valid method of an HTTP request.

E. HTTP/1.0 headers are binary encoded and not human readable.

28/1. (3 points) Which one of the following features is NOT a feature provided by TCP?

A. Retransmission of lost packets

B. Duplicate packet detection

C. Encryption of application data

D. Flow control

E. Ordered delivery

29/1. (3 points) A file system has 4-KB blocks and 4-byte disk addresses. The inode of a file uses 8 direct entries and a single-indirect table which is half full. How large is the file in blocks?

A. 16 blocks

B. 264 blocks

C. 520 blocks

D. 1032 blocks

E. 5004 blocks

30/1. (3 points) Which response best describes the purpose of `/sys` and `/proc` virtual filesystems?

A. To provide a filesystem view of kernel objects and resources currently used by the kernel and user processes

B. To provide standard area to mount temporary devices such as USB keys and DVDs

C. To provide networked filesystem control

D. None of the other responses are correct

E. To provide thread and synchronization control of kernel tasks

31/1. (3 points) The manual page for `strpbrk()` can be found after the function prototypes at the end of the exam. What would the following code print?

```
char *s = "hello";
char *r = strpbrk(s, NULL);
printf("%d\n", r == NULL ? -1 : (int)(r-s));
```

A. -1

B. 0

C. 5

D. The behavior is undefined, but is guaranteed not to cause a segfault.

E. The behavior is undefined, and it may cause a segfault.

31/2. (3 points) The manual page for `strpbrk()` can be found after the function prototypes at the end of the exam. What would the following code print?

```
char *s = "asteroids";
char *r = strpbrk(s, "ids");
printf("%d\n", r == NULL ? -1 : (int)(r-s));
```

A. -1

B. 0

C. Undefined

D. 6

E. 1

31/3. (3 points) The manual page for `strpbrk()` can be found after the function prototypes at the end of the exam. What would the following code print?

```
char *s = "galaga";
char *r = strpbrk(s, "tempest");
printf("%d\n", r == NULL ? -1 : (int)(r-s));
```

A. 1

B. 0

C. Undefined

D. 6

E. -1

32/1. (3 points) Which one of the following is IMPOSSIBLE?

A. sizeof(char) is 2

B. sizeof(int) is 8

C. sizeof(int*) is 4

D. sizeof(char*) is 8

E. sizeof(void*) is 4

32/2. (3 points) Which one of the following does not depend on the computer architecture?

A. sizeof(char)

B. sizeof(int)

C. sizeof(int*)

D. sizeof(char*)

E. sizeof(void*)

32/3. (3 points) Which of the following best describes the C code below? Assume this is part of a C main method and malloc returns a non-NULL value.

```
1  int* ptr = (int*) malloc(4);
2  *ptr = 42;
3  free(ptr);
4  ptr = (int*) 42;
```

A. May crash at line 2 if an integer requires more than 4 bytes of storage

B. C uses 'new' and 'delete' not 'malloc' and 'free'

C. Allocates 4 bytes of memory on the stack

D. Will always crash at line 3

E. Will always crash at line 4

32/4. (3 points) Which of the following best describes the C code below? Assume this is part of a C main method and malloc returns a non-NULL value.

```
1  int* ptr = (int*) malloc(sizeof(int));
2  *ptr = 42;
3  free(ptr);
4  ptr = (int*) 42;
5  free(ptr);
```

A. May crash at line 2 if an integer requires more than 4 bytes of storage

B. C uses 'new' and 'delete' not 'malloc' and 'free'

C. Allocates 4 bytes of memory on the stack

D. Will always crash at line 3

E. Will always crash at line 5

32/5. (3 points) Which of the following best describes the C code below? Assume this is part of a C main method and malloc returns a non-NULL value.

```
1  void* v = malloc(4);
2  free(v);
3  free(v);
```

A. Is a memory allocation error described as "double free"

B. Is a memory allocation error described as "free after malloc"

C. Allocates 4 bytes of memory on the stack

D. Is valid and error-free

E. To be error free line 1 requires a cast to an int or character pointer

32/6. (3 points) Which one of the following best describes the `free` call in the following code example?

```
1  int* v = NULL;
2  free(v);
```

A. The above `free` call has no effect and is error free

B. Frees up all previously allocated memory

C. Is invalid and commonly described as a 'NULL-free' error

D. Is invalid and commonly described as a 'free-on-null' error

32/7. (3 points) The following expression uses `sizeof` and `strlen` function. What is the value of result?

```
int result = 1 + sizeof("abc") + ( sizeof("abc") * strlen("abc") );
```

A. 17
B. 16
C. 13
D. 21
E. None of the other responses are correct

32/8. (3 points) The following C code is executed as part of a main method. Which line, if any, will likely cause the program to crash?

```
1  char * ptr = (char*) rand(); /* rand() returns an random integer value */
2  int * b = (int*) ptr;
3  b = b + 1;
4  ptr = (char*) rand();
5  *ptr = (char) rand();
```

A. 5
B. 4
C. 3
D. 2
E. None of the other responses are correct

32/9. (3 points) Which one of the following best describes how to find the length of a C string?

A. Requires O(N) search to find the terminating null character `\0`
B. Requires O(1) lookup to read the length byte
C. Is compiler dependent and not part of the C specification
D. Requires O(N) reverse linear search
E. None of the other responses are correct

32/10. (3 points) Which one of the following correctly allocates enough bytes on the heap to copy an existing string pointed to by a character pointer, `char* src`?

A. `malloc( strlen(src) + 1);`
B. `malloc( sizeof(src) + 1);`
C. `new string( sizeof(src) + 1);`
D. `char array[ strlen(src) ];`
E. None of the other responses are correct

32/11. (3 points) Which one of the following best describes `malloc`?

A. `malloc` will return `NULL` if it cannot reserve sufficient heap memory
B. `malloc` will return -1 if it cannot reserve sufficient stack memory
C. `malloc` will throw an exception if there is insufficient free ram
D. `malloc` will always successfully allocate heap memory
E. None of the other responses are correct

32/12. (3 points) My C program prints `Hello 42 3.14`. Which response is the best choice for the next line?

```
1 char* ptr = "Hello";
2 int x = 84 >>1;
3 double d = 3.14159265;
3 ?
```

A. `printf("%s %d %.2f",ptr,x,d);`
B. `write(ptr,5);write(x,2); write(&d,3);`
C. `cout <<ptr<<" "<<x<<" "<<&d;`
D. `printf("$1s $2d $1f",ptr,x,d);`
E. `printf("${ptr} ${x} ${d}");`

32/13. (3 points) Which one of the following is true for typical layout of a process's memory?

A. Program constants are stored in the stack

B. Writing to read-only memory is ignored by the operating system

C. Program code is not stored in the process's memory

D. All of the process's memory address maps to physical RAM address

E. Program constants are read-only

32/14. (3 points) Which one of the following best describes the following C code?

```
1 char array[] = "ABC";
2 char x = array[3];
3 char y = array[4];
4 x = y;
```

A. Buffer overflow at line 2. `x` may contain data from another variable

B. Buffer overflow at line 3. `y` may contain data from another variable

C. The program will not compile

D. The program will crash at line 2

E. The program will throw an exception

32/15. (3 points) Which one of the following best describes for the following C code?

```
1 char array[] = "ABCD";
2 char x = array[5];
3 char y = array[0];
4 x = y;
```

A. Buffer overflow at line 2. `x` may contain data from another variable

B. Buffer overflow at line 3. `y` may contain data from another variable

C. The program will not compile

D. The program will crash at line 4

E. None of the other responses are correct

32/16. (3 points) Which one of the following is correct?

A. `write` always calls `printf` when it is called

B. `printf` uses a buffer so may not call `write` every time it is called

C. `printf` always calls `write` when it is called with more than one argument

D. `write` and `printf` are identical and have the same function prototype

E. `printf` is a system call, `write` is a C library call

32/17. (3 points) The `printf` function declaration can be included in your C program by writing...

A. `#include <stdio.h>`

B. `#include <iostream>`

C. `#define iostream.h(printf)`

D. `#define "sys/printf.h"`

E. None of the other responses are correct

32/18. (3 points) Carefully read the following C code and determine how often it will print `lucky`.

```
int a = rand(); /* returns a random int */
if( a = 4) printf("You're lucky!");
```

A. You are always lucky

B. You are never lucky

C. You have a small chance of being lucky

32/19. (3 points) Carefully read the following C code and determine how often it will print `lucky`.

```
int a = rand(); /* returns a random int */
if( a = 0) printf("You're lucky!");
```

A. You are always lucky

B. You are never lucky

C. You have a small chance of being lucky

32/20. (3 points) If `sizeof(int)` is 2 what will be the expected output of the following C code?

```
char* ptr = "ABCDEF";
int * x = (int*) ptr;
printf("%s", x + 1 );
```

A. `CDEF`

B. `ABCDEF1`

C. `EF`

D. `BCDEF`

E. `Segmentation Fault`

32/21. (3 points) Which response best describes the following code? Assume `ptr` holds the address `0x8400`.

```
1 void* ptr =   /* code not shown */
2 char* ptr2 = (char*)ptr;
3 void* x = & ptr2 + 1;
4 int result = *(ptr2 +1);
```

A. One byte of memory at address 0x8401 is read at line 4

B. One byte of memory at address 0x8401 is read at line 3

C. One byte of memory at address 0x8400 is read at line 2

D. None of the other responses are correct

E. Line 4 has a syntax error

32/22. (3 points) Which of the following best describes the design goal(s) of an operating system?

A. All of the other responses are correct

B. An operating system provides security and guards against malfunctioning user programs

C. An operating system provides a level of abstraction above low-level hardware interfaces

D. An operating system provides a set of services to user programs that can be accessed by system calls

E. An operating system must efficiently manage scarce resources (CPU cores, RAM,...)

32/23. (3 points) In the Linux operating system, which is based on the POSIX standard, which one of the following is true?

A. Each process is isolated and runs in its own virtual memory space

B. Processes can write directly into another processes memory to easily crash the other process

C. A program can only be run by a single user at a time

D. Shell utilities (e.g. `cat ls make`) are written in assembler

E. The overhead of a system calls is the same as a C library call

32/24. (3 points) Which response best describes the behavior of the following code?

```
int mystery(char *start) {
  if( start == NULL) return NULL;
  char* p= start;
  while (*p !='q') p++;
  return p - start;
}
```

A. `mystery("ABC")` is undefined (and may crash)

B. `mystery("q")` returns 1

C. `mystery("q")` returns 2

D. `mystery(NULL)` is undefined (and may crash)

E. `mystery(NULL)` returns 1

32/25. (3 points) Which response best describes the following student code that attempts to implement string copy?

```
1 void mystery(char *dest, char *src) {
2  if( src == NULL || dest==NULL) return;
3  while (src) {
4    *dest = *src;
5    src ++; dest++;
6  }
7  *dest = (char)0;
8 }
```

A. The function will be correct by changing a small error at line 3
B. The function will be correct by changing a small error at line 7
C. The function will be correct by changing a small error at line 4
D. The function will be correct by changing a small error at line 5
E. The function will be correct by changing two small errors at line 4 and 5

32/26. (3 points) Which response best describes the following student code that attempts to implement string copy?

```
1 void mystery(char *dest, char *src) {
2  if (src == NULL || dest==NULL) return;
3  while (*src) {
4    dest = src;
5    src++; dest++;
6  }
7  *dest = (char)0;
8 }
```

A. The function will be correct by changing a small error at line 4
B. The function will be correct by changing a small error at line 7
C. The function will be correct by changing a small error at line 3
D. The function will be correct by changing a small error at line 5
E. The function will be correct by changing two small errors at line 4 and 5

32/27. (3 points) Which response best describes the following student code that attempts to implement string copy?

```
1 void mystery(char *dest, char *src) {
2  if (src == NULL || dest==NULL) return;
3  while (*src) {
4    *dest = *src;
5    src ++; dest++;
6  }
7  *src = (char)0;
8 }
```

A. The function will be correct by changing a small error at line 3
B. The function will be correct by changing a small error at line 7
C. The function will be correct by changing a small error at line 4
D. The function will be correct by changing a small error at line 5
E. The function will be correct by changing two small errors at line 4 and 5

32/28. (3 points) Which one of the following best describes the code below?

```
int *p = (int*) malloc(sizeof(int));
p = NULL;
free(p);
```

A. Compiler error: `free` cant be applied on NULL pointer
B. Memory leak
C. Dangling pointer
D. The program may crash as `free()` is called on a NULL pointer

E. There are no errors in the code

The original allocation is never freed,so the code has a memory leak. Note, `free(NULL)` does nothing.

32/29. (3 points) Which one of the following best describes the correct line 4 to read the line and store the result in the buffer and score variables?

```
1  char* buffer = (char*) malloc(16);
2  int score,res;
3  char* line = "Pointers 123";
4  _______  ?
```

A. `res = sscanf(line, "%15s %d",buffer, &score);`

B. `res = sscanf(line, "%15s %d",buffer, score);`

C. `res = sscanf(line, "%15s %d", &buffer, &score);`

D. `res = sscanf(line, "%15s %d", &buffer, score);`

E. `res = sscanf(line, "%15s %d", *buffer, *score);`

The `buffer` variable points to the memory that we wish `sscanf` to write into, whereas `score` is the integer that we wish to change, so we need its address.

32/30. (3 points) Which one of the following is most likely to print `Hi` only once?

A. `fork(); write(1,"Hi",2);`

B. `execl("nosuchfile","nss",(char*)NULL); write(1,"Hi",2);`

C. `execl("/bin/ls","ls",(char*)NULL); write(1,"Hi",2);`

D. `puts("Hi");if(1) puts("Hi");`

We need `exec` to fail! The incorrect response `execl("/bin/ls","ls",(char*)NULL);` will replace the running process with the `/bin/ls` program and the `write` call will never be executed.

32/31. (3 points) When will `fork()` return a positive integer (a process id) ?

A. In the parent process

B. In the child process

C. If an error occurs

D. When a child needs to be restarted

E. When the parent needs to be signalled

`fork` returns -1 (fail! No fork for you!) or: 0 in the child and a positive integer in the parent - so the parent can store the process id the newly created child.

32/32. (3 points) When will `fork()` return 0 ?

A. In the parent process

B. In the child process

C. If an error occurs

D. When a child needs to be restarted

E. When the parent is the first process

`fork` returns -1 (fail! No fork for you!) or: 0 in the child and a positive integer in the parent - so the parent can store the process id the newly created child.

32/33. (3 points) When will `fork()` return -1 ?

A. In the parent process

B. In the child process

C. If `fork` failed

D. When a child needs to be restarted

E. When the parent is the first process

`fork` returns -1 (fail! No fork for you!) or: 0 in the child and a positive integer in the parent - so the parent can store the process id the newly created child.

32/34. (3 points) `puts(ptr)` is equivalent to

A. `printf("%s\n",ptr)`

B. `scanf("%s\n",ptr)`

C. `ptr=getchar()`

D. `fprintf(stderr,"%s\n",ptr)`

E. `signal(SIGINT,ptr)`

32/35. (3 points) Which one of the following is NOT true for a child process created by `fork`?

A. Gets its own complete copy of the parent's process memory

B. Inherits (shares) the parent's open file streams

C. Inherits signal handlers defined using `signal`

D. Cannot start until the parent process has finished

32/36. (3 points) Which one of the following is true for `gets`?

A. The function `gets` is the recommended function to read lines of input into a buffer

B. Allows a buffer overflow if the input line is longer than the buffer

C. Can only read text data from a file

D. Can only read binary data from a file

E. Returns an integer pointer

32/37. (3 points) How many times will `!` be printed?

```
fork(); printf("!\n"); fork();
```

A. 1

B. 2

C. 3

D. 4

E. 5 or more

The first `fork` creates two processes. Then we print `!` and flush the output to the terminal. The second `fork` is executed by both processes, so now there are four processes, however there is no more output processing at this point.

32/38. (3 points) How many times will `!` be printed?

```
fork();  fork(); printf("!\n"); exit(0);
```

A. 1

B. 4

C. 3

D. 2

E. 5 or more

The first `fork` creates two processes. The second `fork` is executed by both processes, so now there are four processes. Then all four processes print `!` and flush the output to the terminal.

32/39. (3 points) How many times will `!` be printed?

```
printf("!"); fork();  fork(); exit(0);
```

A. 1

B. 4

C. 3

D. 2

E. 5 or more

The printf does not flush the buffer, so the process memory contains `!` in its buffer. The first `fork` creates two processes. The second `fork` is executed by both processes, so now there are four processes. At exit all four processes flush their buffer to the terminal.

32/40. (3 points) A zombie is created when

A. A parent doesn't wait on a finished child

B. A child doesn't wait on a parent

C. A child calls `exec`

D. The parent finishes before the child process

E. A parent sends the child process a `SIGZOMB` signal

If the parent finishes first then the child's parent will become process 1 (init) which will correctly wait on processes.

32/41. (3 points) Which one of the following is NOT true for the following line?

```
waitpid(id,&status,0); // Assume waitpid is successful
```

A. The status variable is an `int; &status` means the address of the integer

B. If the child exited normally then `status` it includes the lowest 8 bits of the child's exit value

C. The `status` variable can be queried with macros `WIFEXITED,WIFSIGNALED,WEXITSTATUS` to extract the status of the child process

D. Is used to cleanup zombies from the kernel's process table

E. Is used to prevent child processes from starting immediately

32/42. (3 points) Which one of the following is NOT true?

A. Dead processes (zombies) still take up space in the system's process table. If the system table is full no new processes can be created.

B. `alarm(5)` will send an asynchronous signal `SIGALRM` to the process in 5 seconds

C. The default SIGALRM handler does nothing

D. Standard error stream (std err, file descriptor 2) is not buffered

E. If `open` succeeds it will return the smallest unused non-negative integer

The default action of SIGALRM is to kill the process.

32/43. (3 points) Which response bests describes the common system programming pattern to run another program and wait for it to finish?

A. Parent process forks then child execs and parent waits

B. Parent process forks then parent execs and child waits

C. Parent process execs then child forks and child waits

D. Child process execs then child forks and child waits

E. Child process waits then parent exec and child forks

Note, this common system programming pattern has the insightful name "fork-exec-wait."

32/44. (3 points) Which one of the following is true for `fork()` ?

A. Creates a new process by reloading the program - the child starts at `main()`

B. Creates a new process by cloning the existing process; the child starts by `fork()` returning 0

C. Creates a new file handle for standard out

D. Creates a new file handle for standard int

E. Installs a new fork handler

The cloning of the process happens at the moment of calling `fork`. The process is not restarted. Looking backwards in time, it is as if two processes showed the same history. Looking forwards in time, both processes have their own address space - changes to variables in one process will not affect the other.

32/45. (3 points) Which response best describes the following buggy code that, when executed on a 32 bit machine, is suppose to create a 16x16 2D character array?

```
1  char ** array;
2  array = (char**) malloc(16 * sizeof(char));
3  if(!array) return 1; // malloc failed
4  int i= 0;
5  for(; i < 16;i++)
6    array[i] = malloc(16 * sizeof(16));
```

A. Insufficient memory allocated (line 2) causes a buffer overflow when i is 0

B. Insufficient memory allocated (line 2) causes a buffer overflow when i is 4 and higher

C. Insufficient memory allocated (line 2) causes a buffer overflow when i is 8 and higher

D. Insufficient memory allocated (line 2) causes a buffer overflow when i is 12 and higher

E. Insufficient memory allocated (line 2) causes a buffer overflow when i is 16 and higher

16 bytes are allocated on the heap. On a 32 bit machine each pointer uses 4 bytes, thus there is sufficient space for 4 pointers. The fifth pointer (i=4) and higher, will be stored beyond the edge of the buffer.

32/46. (3 points) Which of the following is FALSE for `getline`?

A. `getline` arguments include a pointer to an int and a pointer to a pointer to char, so it can modify their contents.

B. Th character pointer is typically set to `NULL` before the first call to `getline`

C. To avoid a memory leak, call `free` on the buffer after the last call to `getline`

D. `getline` returns the number of characters read (possibly including a newline character at the end)

E. Is used to convert a character array into integer and floating point values

32/47. (3 points) Which one of the following is the best description of POSIX process control? When a child process finishes (or temporarily stops) ...

A. The init (process 1) is sent a `SIGUSR1` signal

B. The parent process is sent a `SIGCHLD` signal
C. The child process is re-assigned a new parent process
D. All siblings are notified with a `SIGQUIT` signal
E. The process is automatically restarted

32/48. (3 points) Which one of the following is NOT true?
A. The default action of `SIGCHLD` is to do nothing
B. The default action of `SIGALRM` is to quit the process
C. Pressing `CTRL-C` will send a `SIGINT` signal to the process
D. Signals are software interrupts - they are handled concurrently by signal handlers.
E. Signals can be sent to other processes using the 'signal' function

Signals are sent to other processes using `kill`

32/49. (3 points) What will be the last thing printed by the following program?

```
1 int main() {
2    int c = fork();
3    printf("c=%d : pid=%d ppid=%d\n",c, getpid(),getppid() );
4    if(c>0) return 97;
5    sleep(4); printf("Answer: %d\n",getppid());
6    return 80;
7  }
OUTPUT:
c=0 : pid=97 ppid=90
c=97 : pid=90 ppid=80
____ ?
```

A. `Answer: 1`
B. `Answer: 80`
C. `Answer: 90`
D. `Answer: 97`
E. None of the other responses are correct

When a process is orphaned because its parent has already finished it is adopted by `init` (process 1). `init` ensures there are no zombies (i.e. it will call `wait` or `waited` for every SIGCHLD signal).

32/50. (3 points) If `malloc` fails (returns NULL) will the following program crash (seg fault)? If so, where?

```
1 void * ptr1 = (void*) malloc(16);
2 int ** ptr2 = (int**) ptr1;
3 int *** ptr3 = & ptr2;
4 void* ptr4= (void*) &ptr1;
```

A. Line 1
B. Line 2
C. Line 3
D. Line 4
E. None of the other responses are correct

`ptr1` is cast but never de-referenced: It is never used to attempt to read/write memory at address 0. The other lines of code get the address of the variable but do not read/write address held by of ptr1.

32/51. (3 points) Which one of the following changes the process's current directory to the user's home directory?
A. `chdir( getenv("HOME") )`
B. `pwd( environ[0] )`
C. `chdir( environ[ getenv("HOME")] )`
D. `pwd( environ[UHOME] )`
E. None of the other responses are correct

32/52. (3 points) Which one of the following is NOT true?
A. `static` variables are automatic

B. The last entry of string arrays `argv` and `environ` is always `NULL`

C. `malloc` allocates memory on the heap

D. `char** environ` should be declared `extern`

E. `argv[1]` is the first argument because `argv[0]` is the program name

`static` variables are stored in the data segment, not the stack or heap. They are valid for the lifetime of the process.

32/53. (3 points) Which one of the following is the best choice for the missing code? Choose the correct snippet so that the program uses a `string.h` function to display a help message when the program is started with `"-h"` option

```
1  int main(int argc, char*argv[]) {
2    // If no arguments or just -h, then show a message and quit:
3    if(argc ==1 ||   _______    ) help_message_and_quit();
```

A. `0==strcmp(argv[1],"-h")`

B. `0==strcmp(argv[0],"-h")`

C. `argv[1] == "-h"`

D. `argv[0] == "-h"`

E. `0==streq(argv[0],"-h")`

`argv[0]` holds the program name. `strcmp` returns 0 if the two arguments are equal.

32/54. (3 points) Four students were asked to write four alternative ways to print `Hello World!` to standard output. Carefully read the four functions below and for each one, decide if it will print `Hello World!` without error. Choose the most accurate response below.

```
void A() { char *s=(char*)malloc(100); strcpy(s,"Hello World!\n"); puts(s); free(s); }
void B() { char s[100]; *s=0; strcat(s, "Hello World!\n"); puts(s); }
void C() { static char s[100]; sprintf(s,"Hello "); strcat(s,"World!\n"); write(1,s,strlen(s)); }
void D() { char *s = "Hello "; strcat(s, "World!"); printf("%s\n", s); }
```

A. None of the functions are correct

B. Only 1 function is correct

C. 2 functions are correct

D. 3 functions are correct

E. All 4 functions are correct

`D()` will segfault at strcat because `s` points to readonly memory (the string constant).

32/55. (3 points) Which one of the following prints `H` to the standard output stream?

```
1 char* ptr = "H";
2 ________?
```

A. `write(1,ptr,strlen(ptr));`

B. `printf("%p",ptr);`

C. `write(sizeof(ptr), ptr, stdout);`

D. `fprintf(stderr,"%s",ptr);`

E. `puts(* ptr);`

33/1. (3 points) Which one of the following is NOT true for the Buddy allocator compared to other memory allocators?

A. Uses a hierarchy of allocation blocks of size 2^n

B. Minimizes fragmentation

C. Optimizes for performance

D. Can be used as a heap allocator

33/2. (3 points) Which one of the following is NOT true for `calloc`?

A. Memory allocated by `calloc` will be initialized to zero

B. `calloc(4,4)` is identical to `calloc(1,16)`

C. Returns `NULL` if memory allocation failed

D. Use `free` to release (de-allocate) memory reserved by `calloc`.

E. Allocates memory in a character stack

33/3. (3 points) Which expression is the best choice to set the value of `result` to 5.0 ?

```
double* ptrA = malloc(sizeof(double));
*ptrA= 5.0;
double* ptrB = (double*) realloc(ptrA, sizeof(double)*2);
double result = ______;
```

A. `ptrB[0]`
B. `ptrA[0]`
C. None of other responses are correct;
D. `*(ptrA + 0)`
E. `ptrA[1]`

`realloc` may return a new address if the original allocation cannot be expanded.

33/4. (3 points) Which line,if any, will likely crash the program?

```
1 int main() {
2  int a = 10,**c, *d;
3  c = &d;
4  *c = &a;
5  **c= 5;
6  d = NULL;
7  return 0;
}
```

A. 5
B. 6
C. The program will not crash
D. 3
E. 4

33/5. (3 points) Which of the following will NOT reserve enough memory for 10 character pointers. Assume a character pointer requires 8 bytes of storage

A. `malloc(80);`
B. `calloc(10,8);`
C. `malloc(sizeof(char)* 10);`
D. `calloc(10, sizeof(char*));`
E. All of the other responses reserve sufficient memory

`malloc(sizeof(char)* 10);` which only reserves 10 bytes because `sizeof(char) != sizeof(char*)`.

33/6. (3 points) Which of the following is FALSE for pthreads?

A. Include `pthread.h` to get declarations for `pthread_create` etc
B. Add the `"-pthread"` gcc compiler option to build multi-threaded programs
C. A running thread can be transferred from a child to its parent process
D. Multi-threaded programs can still be executed on machines with a single CPU

33/7. (3 points) Which of the following will NOT cause a multi-threaded process (with multiple threads currently running) to terminate?

A. The process is delivered a `SIGTERM` signal
B. A background thread writes to address zero
C. A background thread calls `exit`
D. The original thread calls `pthread_exit` from `main`
E. The original thread `return`s from `main`

`pthread_exit` never returns; it terminates the calling thread. Only if all threads have now exited will the process exit (with value 0). However in this question, there are other threads running so the process will remain.

33/8. (3 points) Complete the following code to print `Hello`. What is the missing line at line 8?

```
1   void* func(void*p) {
```

```
2      return p;
3   }
4   int main() {
5    pthread_t id;
6    pthread_create(&id,NULL, func,"Hello");
7    void *r;
8    ????
9    puts( r); return 0;
10 }
```

A. `pthread_join(id, & r);`
B. `pthread_exit(&r);`
C. `r = pthread_exit(id);`
D. `pthread_join(&id, r);`
E. `pthread_wait(NULL,r);`

Use `pthread_join` to wait for a thread to finish and to find its return value (or the value it passed to `pthread_exit`). Note the correct answer must be pthread_join(id, & r); because we need to pass the *address of* r; If you just passed "r" then you are passing whatever address r happens to contain (which is an arbitrary value).

33/9. (3 points) Which of the following is NOT true?

A. Some C library functions e.g. `asctime`, `strtok` are not thread-safe
B. Two threads can use the function at the same time if it is "thread-safe"
C. A function that uses static (global) variable to hold a result value is not thread-safe
D. If a function is documented as "not thread-safe" then it must not be used in multi-threaded programs

33/10. (3 points) Which one of the following items is NOT stored on the stack?

A. A value for the CPU's Program Counter (return address)
B. Parameter values of the function
C. Automatic (local) variables
D. static variables

static variables are stored in the data segment (just below the heap).

33/11. (3 points) Which response best fills in the three blanks: When a program calls `pthread_create`, a new ? and ? are created ? the current process.

A. thread ; stack ; inside
B. thread ; heap ; inside
C. process ; stack ; outside
D. process ; thread ; by forking
E. thread ; heap ; outside

33/12. (3 points) In a POSIX system (such as LINUX) which one of the following is TRUE by default?

A. Processes are isolated and run inside their own virtual memory space
B. Parent processes can write into the child process
C. Child processes can write into the parent process
D. While a process is running it is allocated the entire physical memory (RAM)
E. A process can only use a single CPU at a time

33/13. (3 points) For a linked-list heap allocator which response best describes the following statements about explicit free lists compared to implicit free lists?

```
1 Can decrease allocation time
2 Find-first allocation algorithm can be mapped onto different placement strategies
3 Require separate storage outside of the heap for the linked list
4 Require additional operating system support to manage the heap
```

A. Only 1 and 2 are correct
B. Only 1 is correct
C. Only 2 is correct
D. Only 3 and 4 are correct

E. Only 1 3 and 4 are correct

33/14. (3 points) Which response does NOT describe a Boundary Tags -based allocator described by Donald Knuth?

A. Traverse allocated blocks by using their size

B. Store size of block at the beginning and end of the block.

C. Is an implicit linked list implementation

D. Requires a buddy allocator to coalesce adjacent blocks

E. Coalesces blocks to prevent false-fragmentation

33/15. (3 points) When a heap allocator requires more heap memory it calls

A. `Ghostbusters`. Just kidding. Hint this response is incorrect.

B. `sbrk`

C. `malloc`

D. No system call is required; heap space is allocated automatically by the MMU

E. `heap_alloc`

33/16. (3 points) Which one of the following is FALSE?

A. `pthread_exit` waits until all other threads finish before returning

B. Each thread requires its own stack space

C. pthreads are peers; there is no hierarchy of threads in the same process

D. Creating threads is faster than forking process

E. pthreads in the same process share the same heap and the same virtual memory address space

`pthread_exit` never returns.

33/17. (3 points) Using an initial heap size of 2^{10} bytes (1KB) and a binary buddy-allocator, how many memory allocation requests of 68 bytes can be completed before the allocator requires additional heap memory?

A. 9 or fewer

B. 14

C. 13

D. 15

E. 16 or greater

Round allocation requests up to nearest 2^n i.e. 128 bytes (2^7)

33/18. (3 points) The statement "`pthread_mutex_lock` is an atomic operation" ...

A. is a warning that the lock may explode if the lock is not correctly initialized

B. means the locking operation behaves as if it is a single uninterruptible operation

C. is a warning that the mutex can only be locked once

D. means that only one lock can be locked at at time

E. means it can only be used on single CPU systems

33/19. (3 points) Which one of the following is TRUE for a correctly written multi-threaded program that has locked a mutex of type `pthread_mutex_t`?

A. The thread that initialized the mutex must have also locked it

B. The lock can now be destroyed using `pthread_mutex_destroy`

C. The lock was initialized with a positive count

D. The same thread that locked the mutex must unlock it

E. A call to `pthread_mutex_lock` on the same mutex by other threads will return with an error

33/20. (3 points) Complete the following. Which best describes two well known solutions to the *The Critical Section Problem*?

A. Dekker published the first correct solution. Later, Peterson published a simple solution.

B. Peterson published the first correct solution. Later, Dekker published a simple solution.

C. Turing published the first correct solution. Later, Dijkstra published a simple solution.

D. Hopcroft published the first correct solution. Later, Dijkstra published a simple solution.

E. von Neumann published the first correct solution. Later, Ullman published a simple solution.

33/21. (3 points) Which response best describes "Bounded Wait"?

A. If a thread is waiting to enter the critical section (CS), then other threads may only enter the CS first, a finite number of times.

B. If a thread is waiting to enter the critical section (CS), then atomic exchange assures waiting time is limited to less then N CPU instructions.

C. A thread inside the critical section may only sleep for a finite number of milliseconds before continuing.

D. Multi-threaded performance is only guaranteed if threads do not sleep inside the critical section.

E. Before sleeping or performing slow I/O during a critical section, threads must preemptively unlock the mutex.

33/22. (3 points) Which response best describes "Mutual Exclusion"?

A. Only one thread may be executing code inside the critical section at a time.

B. Two threads may not be executing the same line of code at the same time.

C. Two threads may not perform I/O at the same time.

D. Multi-threaded performance is only guaranteed if one thread sleeps during I/O actions of the second thread.

E. Before sleeping or performing slow I/O during a critical section, threads must preemptively unlock the mutex.

33/23. (3 points) Which response best describes "Progress"?

A. If there are no threads inside the critical section, a thread should be able to enter immediately.

B. The CPU clock may not be reset to an earlier time.

C. Two threads may not perform terminal I/O at the same time.

D. Only one thread may lock a mutex lock a time.

E. In a multi-threaded program, a thread must preemptively unlock the mutex if it is required by another thread.

33/24. (3 points) Which response is an example of "Deadlock"?

A. When two threads cannot continue because they are both waiting for the other one to finish.

B. When a mutex is destroyed but another thread calls `pthread_mutex_lock` on the same mutex.

C. When a mutex cannot be unlocked because it was locked from another thread.

D. When a mutex is transformed into an inconsistent state because it was initialized twice.

E. When a mutex is transformed into an inconsistent state because it was destroyed and then re-initialized.

33/25. (3 points) Complete the following by choosing the best response. On modern processors, implementations of mutex locks require hardware support. For example x86 (Intel) processors implement the XCHG CPU instruction. The relevant characteristics of this instruction are that ...

A. it exchanges the contents of a register and memory and is atomic.

B. it exchanges the contents of two data registers and is non-atomic.

C. it exchanges the contents of a data register and PC register and satisfies bounded waiting.

D. it exchanges the contents of a data register and stack pointer and satisfies progress.

E. it inverts the bit pattern stored in one byte of memory and will never deadlock.

33/26. (3 points) Complete the following by choosing the best response. Simple implementations of correct solutions to the critical section problem may fail on some architectures because ...

A. For performance, the CPU and compiler may re-order instructions and cache reads may be stale.

B. Memory reads can deadlock.

C. CPU speeds (instructions per second) are now faster than main-memory read-write access.

D. Some programs are only single threaded.

E. It is not possible to implement critical section problem solutions in real software.

33/27. (3 points) Solve my riddle! Four threads call my X function but only two may continue; the other two threads must wait! Later my Y function is called once more and one of the two waiting threads is allowed to continue. What am I and what is X and Y?

A. I am counting semaphore, X is wait and Y is post.

B. I am a mutex, X is lock and Y is unlock.

C. I am a critical section, X is start and Y is end.

D. I am a condition variable, X is signal and Y is wait.

E. I am a mutex, X is signal and Y is waitpid.

33/28. (3 points) Solve my riddle! Whenever a thread calls my X function it must always wait in jail! Later when my Y function is called then one waiting thread (if there is one) is released and allowed to continue. What am I and what is X and Y? Hint: I occasionally release jailed threads for no reason!

A. I am counting semaphore, X is wait and Y is post.

B. I am a mutex, X is lock and Y is unlock.

C. I am a critical section, X is start and Y is end.

D. I am a condition variable, X is wait and Y is signal.

E. I am a mutex, X is signal and Y is waitpid.

33/29. (3 points) Which response best describes the following code?

```
int b = 0;  /* shared between threads */

void lock() {
  while( b ) {/* busy wait*/};
  b = 1;
}
void unlock() { b = 0; }
```

A. A correct implementation of a mutex lock

B. Incorrect lock implementation (suffers from a race condition) and does not satisfy Progress (may deadlock)

C. Incorrect lock implementation (suffers from a race condition) and does not satisfy Mutual Exclusion

D. Incorrect lock implementation (suffers from a race condition) and does not satisfy Bounded Wait

E. This implementation is equivalent to Peterson's solution

Consider two threads calling `lock()` at the same time. Both would see a b value of zero (0). Both would continue.

33/30. (3 points) Which of the following is FALSE for condition variables?

A. Occasionally a call to `pthread_cond_wait()` may return even without any corresponding `pthread_cond_signal()` or `pthread_cond_broadcast()` call

B. Condition variables use a helper mutex lock which must be locked before calling `pthread_cond_wait()`

C. During `pthread_cond_wait()` the mutex is automatically unlocked and later relocked before returning

D. A thread can wake up one or all threads that are waiting on a condition variable

E. Condition variables are initialized with a user-supplied condition callback function that returns 0(wait) or 1(continue).

33/31. (3 points) Which response best describes the code below? Each process or thread has it's own flag plus there is a shared-variable named `turn`. Identify the two missing pieces to complete Dekker's N=2 solution.

```
raise my flag
while your flag is raised :
   if it's your turn to win :
      lower my flag
      wait while your turn
      raise my flag
// Do Critical Section stuff
set your turn to win
lower my flag
```

A. This is Dekker's solution for 2 processes

B. Does not satisfy mutual exclusion

C. Does not satisfy progress but satisfies mutual exclusion

D. Does not satisfy bounded wait but satisfies mutual exclusion

E. This is Peterson's N=2 solution

33/32. (3 points) Which response best describes the code below? Each process or thread has it's own flag plus there is a shared-variable named `turn`.

```
raise my flag
Set turn to you
wait while (your flag is raised and it's your turn)
// Do Critical Section stuff
lower my flag
```

A. This is Peterson's solution for 2 processes

B. Does not satisfy mutual exclusion

C. Does not satisfy progress but satisfies mutual exclusion

D. Does not satisfy bounded wait but satisfies mutual exclusion

E. This is Dekker's solution for 2 processes

33/33. (3 points) Solve my riddle! Whenever a thread calls my X function it should later call my Y function. If two or more threads call X then I shall declare a winner and the other(s) will have to wait! What am I and what is X and Y?

A. I am counting semaphore, X is wait and Y is post.

B. I am a mutex, X is lock and Y is unlock.

C. I am a critical section, X is start and Y is end.

D. I am a condition variable, X is wait and Y is signal.

E. I am a mutex, X is signal and Y is waitpid.

33/34. (3 points) Which response best describes the following code to 'solve' the Critical Section Problem? Assume both flags are initially down.

```
raise your flag
lower my flag
wait until my flag is up
// Perform critical section activities
raise my flag
lower your flag
```

A. This is Turing's solution

B. This is correct only for multi-threaded processes

C. Does not satisfy mutual exclusion

D. Does not satisfy progress but mutual exclusion is satisfied

E. Does not satisfy bounded wait but mutual exclusion is satisfied

33/35. (3 points) Which response best describes the following attempt to solve the Critical Section Problem for two processes (or threads)? Assume both flags are initially down.

```
wait while my flag is up
raise your flag
// Perform critical section activities
lower your flag
```

A. Does not satisfy mutual exclusion

B. Does not satisfy progress but mutual exclusion is satisfied

C. Does not satisfy bounded wait but mutual exclusion is satisfied

33/36. (3 points) Which one of the following is NOT TRUE?

A. A pthread mutex lock can be easily replaced with a counting semaphore (albeit with a slight loss of performance)

B. A condition variable is initialized with an integer counter.

C. A counting semaphore can be implemented with a mutex lock and condition variable

D. Underflow and overflow of a queue data structure can be prevented using counting semaphores

E. Waiting on a condition variable should be wrapped in a loop (in part due to spurious wake ups)

33/37. (3 points) Which one of the following is NOT TRUE?

A. `PTHREAD_MUTEX_INITIALIZER` can be used on memory allocated from the heap

B. `pthread_mutex_init` is an alternative function to initialize a mutex

C. Not calling `pthread_mutex_destroy` can lead to resource leaks because the mutex may include a pointer to a system-based synchronization primitive

D. Programs should not use the contents of pthread_mutex_t directly

E. A program may `fork()` after initializing a mutex but by default the mutex is not shared between processes

`PTHREAD_MUTEX_INITIALIZER` can only be used with static (global) variables.

```
pthread_mutex_t m =  PTHREAD_MUTEX_INITIALIZER;
int dostuff() {
  pthread_mutex_lock(&m);
  pthread_mutex_unlock(&m);
}
```

```
int dostuff2() {
  pthread_mutex_t m2;
  pthread_mutex_init(&m2, NULL);
  pthread_mutex_lock(&m2);
  // Do Critical Section stuff here
  pthread_mutex_unlock(&m2);
  pthread_mutex_lock(&m2);
}
```

34/1. (3 points) Analyze the following synchronization code carefully. Which response best describes the following student implementation of a multithreaded stack push and pop function when used with ONE producer thread and ONE consumer thread? Assume `sem_post` and `sem_wait` are never interrupted by a POSIX signal, and the mutex and semaphores are initialized to reasonable values when possible.

```
void push(double value) {
  pthread_mutex_lock(&m);
  sem_wait(&s1);
  stack[ count++ ] = value;
  sem_post(&s2);
  pthread_mutex_unlock(&m);
}
```

```
double pop() {
  pthread_mutex_lock(&m);
  sem_wait(&s2);
  double result = stack[ --count ];
  sem_post(&s1);
  pthread_mutex_unlock(&m);
  return result;
}
```

A. Is valid implementation but only if there is one producer and one consumer thread
B. Has a race condition except when the stack is full or empty, that can cause data corruption
C. `pop` and `push` are implemented correctly (will block until the stack is non-empty or non-full)
D. Will deadlock but only when the stack is full or empty
E. Has a race condition but only when the stack is full or empty that can cause data corruption

`sem_wait` is called *after* the mutex is locked, but the other thread will then deadlock because it is waiting (forever) to lock the mutex.

34/2. (3 points) What are the correct initial values for the three counting semaphores sA,sB,SC so that the following are correct multithreaded stack push and pop functions? Assume `sem_post` and `sem_wait` are never interrupted by a POSIX signal, and N = maximum capacity of array.

```
void push(double value) {
  sem_wait(&sA);
  sem_wait(&sC);
  stack[ count++ ] = value;
  sem_post(&sC);
  sem_post(&sB);
}
```

```
double pop() {
  sem_wait(&sB);
  sem_wait(&sC)
  double result = stack[ --count ];
  sem_post(&sC);
  sem_post(&sA);
  return result;
}
```

A. `sA(1), sB(N) sC(N)`
B. `sA(0), sB(N) sC(1)`
C. `sA(0), sB(N) sC(0)`
D. `sA(N), sB(0) sC(0)`
E. `sA(N), sB(0) sC(1)`

`sA` represents the number of free slots, `sB` represents the number of items, `sC` is being used to ensure mutual exclusion

34/3. (3 points) Which one of the following is NOT true for the Reader Writer Problem
A. There can be multiple active readers
B. There can be multiple active writers
C. When there is an active writer the number of active readers must be zero
D. If there is an active reader the number of active writers must be zero
E. A writer must wait until current active readers have finished

34/4. (3 points) Which one of the following is not one of the Deadlock Coffman conditions?
A. No Pre-emption

B. Mutual Exclusion

C. Circular Wait

D. Hold and Wait

E. No Starvation

34/5. (3 points) Identify the following definition: "The resources can not be shareable between processes at the same time"

A. No Pre-emption

B. Mutual Exclusion

C. Circular Wait

D. Hold and Wait

E. Livelock

34/6. (3 points) Identify the following definition: "There exists a set of process P1,P2,... such that there is a process dependency cycle in the Resource Allocation Graph"

A. No Pre-emption

B. Mutual Exclusion

C. Circular Wait

D. Hold and Wait

E. Livelock

34/7. (3 points) Identify the following definition: "Once a process acquires a resource, it will retain the resource and wait for the next resource"

A. No Pre-emption

B. Mutual Exclusion

C. Circular Wait

D. Hold and Wait

E. Livelock

34/8. (3 points) Identify the following definition: "Once a process acquires a resource, another process cannot force the original process to release it"

A. No Pre-emption

B. Mutual Exclusion

C. Circular Wait

D. Hold and Wait

E. Livelock

34/9. (3 points) Identify the following definition "A process is not deadlocked but is never able to acquire all necessary resources to make progress"

A. No Pre-emption

B. Mutual Exclusion

C. Circular Wait

D. Hold and Wait

E. Livelock

34/10. (3 points) Two singers each need a drummer and keyboard player to complete their bands for a concert. The first singer has enrolled the best drummer, while the second singer has enrolled the best keyboard player. To avoid deadlock the first singer steals the keyboard player from the second singer. Deadlock was avoided by breaking which condition? (Choose the best response)

A. No Pre-emption

B. Mutual Exclusion

C. Circular Wait

D. Hold and Wait

E. None of the other responses are correct.

34/11. (3 points) Two singers each need a drummer and keyboard player to complete their bands for a concert. The drummer and keyboard player decide to work for both. Deadlock is avoided by breaking which condition? (Choose the best response)

A. No Pre-emption
B. Mutual Exclusion
C. Circular Wait
D. Hold and Wait
E. None of the other responses are correct.

34/12. (3 points) Two singers each need a drummer and keyboard player to complete their bands for a concert. Both singers hire a drummer first,then hire a keyboard player second. Deadlock is avoided by breaking which condition? (Choose the best response)

A. No Pre-emption
B. Mutual Exclusion
C. Circular Wait
D. Hold and Wait
E. None of the other responses are correct.

34/13. (3 points) Two singers each need a drummer and keyboard player to complete their bands for a concert. The singers wait until both resources are available before hiring. Deadlock was avoided by breaking which condition? (Choose the best response)

A. No Pre-emption
B. Mutual Exclusion
C. Circular Wait
D. Hold and Wait
E. None of the other responses are correct.

34/14. (3 points) Review the multi-threaded code below for synchronization errors. The two methods are used to increase or decrease `money`. The `withdraw` method should block until there are sufficient funds in the account `(money >= amount)`, Note `PTHREAD_COND_INITIALIZER` is equivalent to `pthread_cond_init`, and the argument `amount` will always contain a positive value.

```
01  int money = 100; /* Must be positive */
02  pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
03  pthread_cond_t cv = PTHREAD_COND_INITIALIZER;
04
05  void deposit(int amount) { /* add money */
06   pthread_mutex_lock(&m);
07   money =  money + amount;
08   pthread_mutex_unlock(&m);
09  }
10  // withdraw() will block;
11  // It will return only when there is sufficient money in the account!
12  void withdraw(int amount) { /* reduce money */
13    if(money < amount) pthread_cond_wait(&cv,m);
14    else pthread_cond_signal(&cv);
15    money = money - amount;
16  }
```

Which one of the following is a true statement about the synchronization used in above functions?

A. The `deposit` method needs to call `pthread_cond_wait`.
B. The `withdraw` method has no synchronization errors.
C. If `pthread_cond_signal` is wrapped inside a while loop then the code is correct.
D. The `withdraw` method must call `pthread_mutex_lock` on the mutex after `pthread_cond_wait` returns.
E. None of the ofter responses are correct.

The above code is horribly broken. The `deposit` call should call broadcast after incrementing money, but before unlocking the mutex The `withdraw` function should lock the mutex for the whole function and use `while(money<amount)` not `if` and the signal call is unnecessary.

34/15. (3 points) Spot the error(s)! 10 threads will call `barrier` once. The first 9 threads should sleep until the 10^{th} thread calls `barrier`, then all 10 threads should continue. Review the multi-threaded code below for synchronization errors. Note `PTHREAD_COND_INITIALIZER` is equivalent to `pthread_cond_init`

```
01  int count=10;
02  pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
03  pthread_cond_t cv = PTHREAD_COND_INITIALIZER;
04
05  void barrier() {
06    pthread_mutex_lock(&m);
07    count--;
08    if(count >0) {
09        while(count>0) pthread_cond_wait(&m,&cv);
10    } else {
11         pthread_cond_signal(&cv);
12    }
13    pthread_mutex_unlock(&m);
14  }
```

Which one of the following is true for the code above?

A. When the 10^{th} thread calls `barrier`, 2 threads will continue executing code after the `barrier` call

B. When the 10^{th} thread calls `barrier`, 10 threads will continue executing code after the `barrier` call

C. Two or more threads can continue executing code after the `barrier` call before the 10^{th} thread calls `barrier`

D. One thread can continue executing code after the barrier call before the 10^{th} thread calls `barrier`

E. When the 10^{th} thread calls `barrier`, no threads will continue executing code after the `barrier` call

34/16. (3 points) Solve the Condition Variables riddle. Once upon a time The FAR-PAR chef tweeted 'Wakeup students the food is ready!' But at that precise moment another student had just checked their messages (None!) and decided to nap just at the moment the chef sent his tweet! Which answer best explains how Condition Variables prevent this race condition?

A. Condition Variables require a locked mutex before `signal` and `wait` calls to ensure the signal is not lost

B. Condition Variables do not prevent this race condition. The signal should be sent twice or more (use a loop around signal)

C. Condition Variables require a counting semaphore to ensure the signal is received by all the waiting threads

D. Condition Variables use a Reader Writer pattern to ensure the signal is queued and received by all the waiting threads

E. Condition Variables must be correctly initialized

Note: Also the first two steps `pthread_cond_wait` of unlocking the mutex and sleeping (until a signal) is performed atomically.

34/17. (3 points)

Five philosophers are invited to sit at a circular table. Each philosopher's brain has only two states: Either thinking or hungry! Around the table I place five chopsticks (one chopstick between each philosopher). A philosopher can eat once they are exclusively holding the two chopsticks that are nearest to them. If a philosopher is unable to pick up the other chopstick after 60 seconds, then they will put their first chopstick down and think for 60 seconds before trying again. Unfortunately being too smart they all use the timers on their smart-watches and all attempt to pick up one chopstick on their left at the same time, then wait for another chopstick, give up, release their first chopstick, and try again, and again and again for ever... This scenarios is best described as

A. Starvation due to livelock

B. Deadlock due to hold and wait

C. Deadlock due to pre-emption

D. A solution to the Dining Philosophers problem

E. Starvation due to deadlock

35/1. (3 points) Spot the error! When run, the `go` function causes a segfault during the `qsort` call. Assume `comp_fn` is correctly written. Which response best describes the bug that caused the segfault?

```
1    pthread_t tid;
2    void* result;
3    void* func(void*m) {
4       qsort(m, 100000, sizeof(int), comp_fn);
5       return NULL;
6    }
```

```
 7    void go() {
 8       void* mem=calloc(100000, sizeof(int));
 9       pthread_create(&tid,NULL,func,mem);
10       free(mem);
11       pthread_join(tid,&result);
12    }
```

A. `qsort` can not be used with heap memory

B. Line 11: `pthread_join` should be `pthread_exit`

C. Line 10 and 11 need to be swapped

D. `qsort` must not be called in a second thread

E. Line 8 and 9 need to be swapped

35/2. (3 points) Which response best describes the following code segment?

```
int main() {
  FILE*fh=fopen("results.txt","w+");
  fprintf(fh, "%d",12345);
  fflush(fh);
  fseek( fh, 0, SEEK_SET);
  pid_t child = fork();
  if(child==0) { /* I'm the child */
    fseek( fh, 0, SEEK_END);
    fclose(fh);
    exit(0); // does not return
  }
  waitpid(child,NULL,0);

  fprintf(fh, "%d",99);
  fclose(fh);
  return 0;
}
```

A. 99 will be written at the end of the file

B. The parent process will segfault because the file was already closed

C. 99 will be written at the start of the file

D. The parent will never successfully write 99 to the file

E. The child process will truncate the file to zero bytes

35/3. (3 points) Which order of calls can be used to determine a file size (for files $<$ 2GB)?

A. `fseek(fh,0,SEEK_END)` then `ftell(fh)`

B. `fseek(fh,-1,SEEK_APP)` then `fpos(fh)`

C. `fpos(fh)` then `fseek(fh,-1,SEEK_APP)`

D. `fset(fh)` then `fseek(fh,0,SEEK_SET)`

E. `fseekend(fh)` then `flength(fh)`

35/4. (3 points) Which one of the following might be used to re-read the first line of a file? Assume `fh` refers to a valid file handle and the line will be parsed using `fscanf` or `fgets`.

A. `fseek(fh,0,SEEK_SET)`

B. `frepo(fh,-1)`

C. `fpos(fh)`

D. `freread(fh)`

E. `freadat(fh,0)`

35/5. (3 points) Spot the error(s)! 5 threads will call `barrier` once. The first 4 threads should sleep until the 5$^{\text{th}}$ thread calls `barrier`, then all 5 threads should continue. A student wrote the following code and wonders if it will work correctly. Carefully review the multi-threaded code below for synchronization errors. Note `PTHREAD_COND_INITIALIZER` is equivalent to `pthread_cond_init`.

```
01  int count=5;
02  pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;
03  pthread_cond_t cv = PTHREAD_COND_INITIALIZER;
04
05  void barrier() {
06    pthread_mutex_lock(&m);
07    count--;
08    pthread_cond_broadcast(&cv);
09    while(count>0)
10              pthread_cond_wait(&cv, &m);
11    pthread_mutex_unlock(&m);
12  }
```

Decide if each statement is true or false and select the appropriate response.
$S1$: "The code suffers from a race condition if two or more threads call `barrier` at the same time.
$S2$: "It is possible that some threads can continue *before* the 5^{th} thread calls `barrier`"
$S3$: "It is possible that one or more of the first four threads may get stuck inside the barrier function even *after* the 5^{th} thread calls `barrier`."

A. Only $S1$ is true
B. Only $S2$ is true
C. Only $S3$ is true
D. Exactly two statements are true
E. None of the other responses are correct

The code implements a barrier albeit an inefficient one: It's certainly inefficient that all threads unnecessarily call broadcast - even before the count is zero. However this will cause threads to wake up and check the count (not yet zero!) and go back to waiting. A better solution would only broadcast when `count==0`

35/6. (3 points) A pipe will generate a POSIX signal (SIGPIPE) ...

A. When reading and the pipe is empty but not when the pipe is full
B. When writing and the pipe is full but not when the pipe is empty
C. When writing and all listeners (readers) are already closed
D. When all writers are closed and a read is attempted
E. When a reader or writer would block

35/7. (3 points) A pipe is an example of

A. APC
B. PAC
C. IPC
D. TLB
E. MMU

35/8. (3 points) In CS241, IPC stands for

A. Interrupted program counter
B. Interprocess cancelation
C. Interprocess communication
D. Inert pre-emptive Coffman
E. Infinite pre-emptive Condition

35/9. (3 points) Which one of the following is NOT TRUE for a hardware implementation of Virtual Memory?

A. The page table may store how recently a particular page was used
B. The page table does not use the lowest bits of the virtual address
C. The page table converts frame numbers into page numbers
D. The page table is stored in RAM
E. Pages can be missing i.e. they may not have any corresponding physical memory associated with them

35/10. (3 points) Which one of the following is TRUE for a typical 32 bit hardware implementation of Virtual Memory? Assume the machine has 128MB of ram

A. The highest 12 bits of the virtual address are used as an offset

B. A typical page size on a 32 bit linux machine is 32MB

C. A single-level page table is sufficient to fit into main memory

D. The page table converts page numbers into offset numbers

E. The page table converts frame numbers into offset numbers

35/11. (3 points) Solve my riddle! I speed up the conversion of a virtual address to a physical address by caching recent results. I am useless if your memory requests are random (you'll need the page tables for that case) but usually your reads and writes are to recently used pages. My short-term memory is tiny but I am extremely fast! What am I called?

A. Physical Address Cache

B. Memory Management Unit

C. Translation Lookaside Buffer

D. Dynamic Ram Translation

E. Address Conversation Cache

35/12. (3 points) Solve my riddle! I can tell if you the frame is the same as an old copy on disk ("Secondary storage"). You'll find me in the page table. What am I called?

A. The frame bit

B. The frame-disk bit

C. The dirty bit

D. The page-to-disk bit

E. The copy bit

35/13. (3 points) A 64 bit architecture with 16 GB of RAM uses 16 KB pages in a three-level page table. How many bits are used for the offset?

A. 16

B. 10

C. 14

D. 4

E. None of the other responses are correct

35/14. (3 points) Choose the best response to fill in the blank. A computational process performs many writes over it's entire virtual memory space with no predictable pattern. In a machine with virtual memory support disabled the running time is 100 seconds. On an equivalent machine with virtual memory enabled that uses a single-level page table, the process would be expected to complete in approximately _____ seconds.

A. 95 to 105

B. 106 to 139

C. 40 to 60

D. 140 or greater

E. 61 to 94

35/15. (3 points) Which one of the following is NOT an advantage of virtual memory?

A. Processes can share frames using the 'mmap' system call.

B. Virtual memory allows processes to share read-only frames (e.g. C library, program code)

C. To prevent fragmentation, sequential frames are assigned sequentially to pages

D. Stack memory can be set to be non-executable (i.e. only contain data)

E. There can be valid virtual addresses that do not have a physical memory assigned

35/16. (3 points) Which one of the following is NOT true for a multi-level page table?

A. Useful for 64bit because it can be sparse; not all sub-tables need to exist

B. Can identify pages that have been modified compared to the copy on disk

C. Is faster than a single-level page table for virtual address translation

D. Like single-page tables, uses an offset for each frame to calculate the physical address

E. For lookups into the same frame, the TLB will be faster at virtual address translation than a multi-level page table

35/17. (3 points) `"fork(2)" "printf(3)"` is a short-hand to mean `fork` is documented in the system-call section (section #2) of the man pages, while `printf` is documented in the C library (section #3) of the man pages. Where would you expect to find `strcmp` and `pipe`?

A. `strcmp(3) pipe(3)`

B. `strcmp(2) pipe(3)`

C. `strcmp(3) pipe(2)`

D. `strcmp(2) pipe(2)`

E. None of the other responses are correct

35/18. (3 points) Analyze the following cryptic code that claims to solve the Reader Writer solution using a simple while loop. Note mutex function names have been shortened.

```
void X() {
 lock(&m);
 while(y>0) { /* Try again! */}
 x++;
 unlock(&m);
 // Do X stuff
 lock(&m);
 x--;
 unlock(&m);
}
```

```
double Y() {
 lock(&m);
 while(y>0 || x>0) { /* Try again! */}
 y++;
 unlock(&m);
 // Do Y stuff
 lock(&m);
 y--;
 unlock(&m);
}
```

Which response best describes the above code?

A. Reader Writer 'naive solution' (suffers from writer starvation); X is write Y is read

B. Reader Writer 'writers-preference solution' (no writer starvation); X is write Y is read

C. Can livelock because mutex is locked during the while loop

D. Reader Writer 'naive solution' (suffers from writer starvation); Y is write X is read

E. Reader Writer 'writers-preference solution' (no writer starvation); Y is write X is read

Notice that Y waits for X and Y, whereas X only waits for Y. Therefore Y is a writer.

For both X and Y code paths: When busy waiting in the while loop, the thread, unlike the cond_wait version, will not release the lock. This has the following affect: If a writer or reader is waiting for a writer to finish then all incoming reader and writer threads will be blocked by the currently locked mutex. However when an active reader or writer has finished they will need to acquire the mutex lock. This is impossible (the lock is being held by a busy-waiting thread). Therefore the two threads will not be able to proceed. This is deadlock not livelock because one thread will continually check the wait condition in the while loop but will never exit the while loop.

A side note: It is also inefficient to wait on a mutex (mutex is designed to be help only for a very brief period and may unnecessarily tie up a CPU).

36/1. (3 points) My IPv6 server program listens for TCP connections on port 2000 but fails when I try use port 1000 instead. Which of the following is the most likely explanation?

A. Ports numbers below 1024 can only be used by processes with root (admin) privileges

B. Port numbers below 1024 are reserved for system tasks

C. Port numbers below 1024 are reserved for IPv4 connections

D. Port numbers below 1024 cannot be used for incoming connections on IPv6

E. Port numbers below 1024 can only be used for internal socket connections on the same host

36/2. (3 points) Which is NOT true for sockets?

A. Must be super-user (root) to create a socket

B. Can be used for IPv4 streaming connections

C. Can be used for IPv6 streaming connections

D. Can be used for packet-based networking

E. Can be used for stream-based networking

36/3. (3 points) I want to store the client's port number stored in a socket C structure in a text file where I will log (keep a note of) all of the incoming connections. Which C function should I use to correctly read the port data from the struct?

A. `ntohs`

B. `port2n`

C. `htons`

D. `htonl`

E. `portin`

36/4. (3 points) Solve my riddle! I am an modern Internet addressing scheme. My addresses are 128 bits - so I'm perfect for everyone's Internet-connected wearables. There's no place like `::1`

A. IPv6

B. IPv5
C. IPv4
D. UDP
E. TCP

36/5. (3 points) Solve my riddle! You'll find me in most Internet packets today. My addresses are 32 bits as 4 billion addresses should be enough for everyone! There's no place like `127.0.0.1`

A. IPv4
B. IPv6
C. TCP
D. UDP
E. IPv5

36/6. (3 points) Solve my riddle! I am a packet-based method of communication. Send the packet and forget! (I behave more like a letter than a phone call). Sometimes I get lost along the way but I don't care - that's your problem!

A. UDP
B. TCP
C. ODP
D. ADP
E. TDP

36/7. (3 points) Solve my riddle! I am a stream-based method of networking. Before you can send any bytes I will negotiate a reliable channel between you and another host. If my packets get lost I'll resend them automatically - you won't even know!

A. TCP
B. UDP
C. ODP
D. ADP
E. TDP

36/8. (3 points) Which one of the following is NOT a feature of TCP?

A. Encryption
B. Packet re-ordering
C. Flow control
D. Packet re-tranmission
E. Simple error detection

36/9. (3 points) Which one of the following correctly describes the minimum network calls required to build a client to connect to a web server?

A. getaddrinfo, socket, connect
B. getaddrinfo connect
C. getaddrinfo, socket, listen, connect
D. getaddrinfo, listen
E. getaddrinfo, socket

36/10. (3 points) Riddle me this! I am a distributed service that can look up hosts for you. Give me a host name and I'll tell you their IP address(es)!

A. DNS
B. DMS
C. HHS
D. HSS
E. DDS

36/11. (3 points) What is the correct order of the "big 4" networking calls to initialize a server?

A. socket, bind, listen, accept
B. socket, listen, bind, accept
C. socket, listen, accept, bind

D. listen, socket, accept, bind

E. accept, socket, bind, listen

36/12. (3 points) Solve my riddle! I am used to allow a backlog of connections. Call me to specify how many unaccepted new connections should be allowed

A. listen

B. bind

C. accept

D. backlog

E. sockqueue

36/13. (3 points) Solve my riddle. I may block if there's no new connections to your server; but don't fear, I will return as soon as someone connects!

A. accept

B. bind

C. listen

D. socket

E. getaddrinfo

36/14. (3 points) Solve my riddle. Use me to attach a socket to a particular port on the host.

A. bind

B. accept

C. listen

D. socket

E. attach

36/15. (3 points) Which of the following is NOT a reason for a process (or thread) to be moved to the ready queue?

A. The currently executing process calls `exit(0)`

B. The currently executing process calls `fork`

C. A new TCP connection is fully initialized and `accept` can now return

D. A thread waiting on a condition variable is signaled

E. A synchronization primitive (e.g. counting semaphore) is unlocked / released

36/16. (3 points) Pick the best response to complete the following, "Passive sockets are used ..."

A. For listening server connections only

B. For client connections only

C. For both client and server connections

D. For system services

E. For UDP clients

36/17. (3 points) A student claims the following code fails to connect to a remote web server. Which response best describes the major bug in this code that causes this problem?

```
int setup_server() {
    struct addrinfo hints, *result;
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    int sock_fd = socket(hints.ai_family, hints.ai_socktype, 0);
    if( getaddrinfo("www.illinois.edu", "80", &hints, &result) )
        return 0;
    if( connect(sock_fd, result->ai_addr, result->ai_addrlen) )
      return 0;
    return sock_fd;
    }
```

A. The `hints` struct is not properly initialized

B. The `getaddrinfo` call is only used for TCP servers

C. The `connect` call is only used for TCP servers

D. The `hints.ai_socktype` should be set to SOCK_DGRAM

E. The port number is incorrect

36/18. (3 points) You are writing a linux-specific high-performance uiuc-coin-chitchat-wearables-database-caching-search-ai-web server that is designed to concurrently handle 10,000 long-lived connections. For the best performance you recommend ....

A. `epoll`

B. `select`

C. `poll`

D. 10,000 threads (one per connection)

E. 10,000 processes (one per connection)

36/19. (3 points) A good reason to use `select` to handle many open connections is

A. It's cross-platform and supported on all POSIX platforms (including embedded devices and Mac OSX)

B. It has better performance than `epoll` when there are 1000s of connections

C. It can be used with multiple processes but `epoll` cannot

D. It can be used with multiple threads but `epoll` cannot

E. `epoll` requires an O(N) scan `select` does not

36/20. (3 points) Using Round robin scheduling, with a time quanta of 4 ms, determine the average wait time of the following processes.

Process	Arrival time (ms)	Execution time (ms)
P1	0	8
P2	4	8
P3	4	4
P4	8	4

Hint: Wait time = total time a process remains in the ready queue

Thus, Wait time for each process = $(Endtime - Arrivaltime - Executiontime)$

A. None of the other responses are correct

B. 3.00 - 3.49 ms

C. 3.50 - 3.99 ms

D. 4.00 - 4.49 ms

E. 4.50 ms or higher

8ms.

0.4	4.8	8.12	12.16	16.20	20.24
P1	P2	P3	P4	P1	P2

Total Wait = (20-8) + (24-8-4) + (12-4-4) + (16-8-4) = 12+12+4+4=32 ms

36/21. (3 points) Three processes repeatably read a block of data, use the CPU to perform a calculation and then write to disk. While waiting for disk I/O the CPU is re-assigned to another waiting (ready) process. The first two processes require a short amount of CPU processing for each block. The third process requires a large amount of CPU time for each block. Which scheduling algorithm(s) exhibits poor parallelism in this circumstance (i.e. poor use of disk resources) ? Select the best response.

A. First come first served (FCFS) but not round robin (RR)

B. First come first served (FCFS) and round robin (RR)

C. Round robin (RR) but not First come first served (FCFS)

D. Neither! Both Round robin and First come first served exhibit optimal disk I/O

E. What do you mean? An African or a European swallow?

36/22. (3 points) Which scheduler(s) would cause a system to become completely unresponsive ('appear to deadlock/crash') if a background batch process entered an infinite loop?

Hint: FCFS = First come first served; RR = Round robin.

A. FCFS and RR

B. FCFS but not RR

C. RR but not FCFS

D. Neither RR nor FCFS

E. "I refuse to answer that question on the grounds that I don't know the answer"

36/23. (3 points) It takes 20ms for a packet to travel from client `X` to the server `yikyak.com` (and the same amount of

time for a packet to be sent back to the client). How quickly can the client fully initialize a new TCP connection? i.e. How many milliseconds are required (starting from the moment the client sends the first packet) until the server is ready to receive the first `data` packet?

A. Minimum 60 ms (3-way-handshake)
B. Minimum 40 ms (2-way-handshake)
C. Minimum 30 ms (Dual-duplex-handshake)
D. Minimum 20 ms (1-way-handshake)
E. Minimum 50 ms (2.5-way-shake)

36/24. (3 points) Which one of the following is NOT part of the TCP header?

A. IP source address
B. source port
C. SYN bit
D. ACK bit
E. Sequence number

The IP source (and destination) address is part of the IP header, not the TCP header.

The SYN and ACK bits are used to initiate and acknowledge sequence numbers.

36/25. (3 points) Four processes of equal priority, `P1 P2 P3 P4`, enter the ready state at the same time. Process 1 requires 1 second of CPU, Process 2 requires 2 seconds of CPU, 3s of CPU for P3 and 4s of CPU for P4. Process 1 takes approximately 4 (wall-clock) seconds to finish. Which response describes the most likely scheduler? Assume there are no other significant workloads.

A. RR
B. SJF
C. FCFS
D. pre-emptive SJF
E. pre-emptive priority FCFS

37/1. (3 points) Which one of the following is true for the shell command `touch abc` ?

A. Creates an empty file if `abc` does not exist. Updates the last modified time to the current time
B. Truncates the file `abc` to zero bytes and removes any hard links to the original file
C. Updates the user and group information of the file `abc` to be the same as the shell process
D. Creates a hard link to the file `abc` in the same directory
E. Converts a symbolic link of the file `abc` to a physical file by coping the contents of the file

37/2. (3 points) Solve my riddle! I am a virtual file in a virtual filesystem. I'll can generate unpredictable secure random byte values by collecting entropy (noise) from the rest of the system. I'm a useful source of random bytes to seed secure strongly-encrypted connections. Reading bytes from me may block until there is sufficient entropy to provide unpredictable values.

A. /dev/random
B. /dev/block
C. /random/entropy
D. /random/noise
E. /dev/entropy

37/3. (3 points) Solve my riddle! I am a virtual file in a virtual filesystem. I provide a source of random values also based on entropy of the system however I never block and will always return pseudo-random values even when there is little entropy in the system.

A. /dev/urandom
B. /dev/noise
C. /random/noise
D. /dev/source
E. /dev/rnd

37/4. (3 points) Which one of the following shell commands has the "setuid bit" set?

A. sudo

B. touch
C. mkdir
D. dd
E. rm

37/5. (3 points) Which one of the following is NOT true when mounting filesystems using Linux's `mount` shell command?

A. Only read-only filesystems can be mounted
B. File access permissions and ownership can be configured as part of the mounting process
C. `mount` requires admin (root) privileges to mount an arbitrary filesystem in an arbitrary directory
D. `mount` can be used to mount virtual and real filesystems
E. `mount` can mount loop-ed filesystems stored as a single file on an existing filesystem

37/6. (3 points) Which of the following is NOT stored as part of the inode in a standard ext2 linux filesystem?

A. filename
B. created time
C. pointers to direct blocks, indirect blocks, double indirect block and triple indirect block
D. reference count
E. file length

37/7. (3 points) I execute `rm abc` to successfully delete my regular file. The file contents however are still accessible under a different filename in a different directory! What is the best explanation?

A. A hard link to the file must have been created before it was deleted
B. A symbolic link to the file must have been created before it was deleted
C. The file must have had its sticky bit set
D. The file must have had its setuid bit set
E. The file must have been created in the `/etc` `/var` or `/tmp` directory

37/8. (3 points) I want to create a link in my home directory to my favorite python installation directory, what kind of link(s) is/are most reasonable for a non-root user?

A. A symbolic link is a good choice but a hard link is not
B. A hard link is a good choice but a symbolic link is not a good choice
C. Both a symbolic link and hard link are good choices for this task
D. Neither symbolic nor hard links are good choices for this task
E. There is no spoon (nor points for choosing this response)

37/9. (3 points) I create a directory with permissions 500. Which response is a reasonable output of the `ls` command below?

```
mkdir -m 500 stuff
ls -ldi stuff
```

A. `12435 dr-x------.  2 angrave angrave 4096 Nov 16 20:51 stuff`
B. `12435 d------r-x.  2 angrave angrave 4096 Nov 16 20:51 stuff`
C. `12435 dr-sr-sr-s.  2 angrave angrave 4096 Nov 16 20:51 stuff`
D. `12435 dr-xr-xr-x.  2 angrave angrave 4096 Nov 16 20:51 stuff`
E. `12435 dr--r-----.  2 angrave angrave 4096 Nov 16 20:51 stuff`

37/10. (3 points) Choose the best fitting description to complete the following: A directory consists of an inode and data blocks. The directory's data blocks contain ...

A. Filenames and inode numbers of the directory entries
B. Only inode numbers of the directory entries
C. Only filenames of the directory entries
D. Only filenames and permissions of the directory entries
E. Only filenames and hard-link entries

37/11. (3 points) Complete the following code to return 1 if the given path corresponds to a valid directory

```
int isdir(char* path) {
  struct stat s;
```

```
  return  ________;
}
```

A. `0==stat(path, &s ) && 0 != S_ISDIR(s.st_mode)`

B. `0==fstat(path, &s ) && 0 != S_ISDIR(s)`

C. `E_ISDIR==open(path,"d")`

D. `S_ISDIR( lstat(&path, *s) )`

E. `dstat(&path, *s) ==1`

Note stat returns 0 if successful

37/12. (3 points) What is the role of `__LINE __` ?

A. The line number of the source code currently being compiled

B. The line number of the input file currently being read

C. The line number of the last kernel call

D. The number of output lines printed by the process to standard out

E. The total number of output lines printed by the process to stdout and stderr

37/13. (3 points) Before modifying an existing file, the umask is changed to 777. What affect, if any, will this have on the existing file? A reminder that `>` redirects standard output to a file and `>>` appends standard output to a file.

```
echo "Hello" > story.txt
umask 777
echo "Again" >> story.txt
```

A. The story.txt contents cannot be modified or read by other users

B. The story.txt contents can be read and modified by other users

C. The story.txt contents is not readable by the user who created it

D. The story.txt contents is not modifiable by the user who created it

E. Existing files and directories are unaffected by the umask value

37/14. (3 points) My old harddisk has a seek time of 10ms and transfer rate of 50 MB/s. I perform the following benchmark on my system which takes 20 seconds to complete

```
dd if=/dev/zero of=/dev/null bs=1M count=123456
```

I now replace my hard disk with a solid state drive (SSD) that can execute 50,000 I/O 4KB block requests per second and a transfer rate of 250 MB/s. What is the expected completion time now for the above benchmark?

A. 10-30 seconds

B. 1-9.9 seconds

C. 100-999 ms

D. 10-99 ms

E. Less than 9ms

The benchmark should run in approximately the same amount of time: The above dd command is reading bytes from a virtual file `/dev/zero` and writing to a virtual file `/dev/null` from the virtual `/dev` file system - there are no bytes being read or written to a physical disk during the approx. 12GB transfer!

37/15. (3 points) Solve my riddle - What am I? I increase the chances of keeping your data safe from drive failure by storing redundant information - a parity bit for every set of bits written to the other disks. For performance, and to reduce stress on any one single drive, the parity bit is distributed across the disk array. You can replace any single failed drive with a new drive and recalculate the drive's contents. However if two drives fail (e.g. another drive fails during the rebuild process) - well... I hope you had recent backup using alternative storage!

A. RAID 5

B. RAID 1

C. RAID 0

D. Reed-solomon coding

E. Mirror

37/16. (3 points) Which of the following describes how Google manages its distributed file system "Colossus"?

```
1 Multiple copies in different geographic regions
2 Reed-solomon encoded data-blocks to recover from single bit and multiple-bit errors
```

```
3 Resilient to sudden failure of individual disks, servers, racks of servers, and entire data-centers
4 Pro-active warning when free space is under 1 petabyte
```

A. (1) (2) (3) (4)
B. (1) + (2) only
C. (3) + (4) only
D. None of the other responses are correct
E. (4) only

37/17. (3 points) How many lines will be printed by this program and what is the content of each line? Assume all system calls complete successfully.

```
int main(int argc, char**argv) {
  mkdir("dir1",0755);
  mkdir("dir1/dir2",0755);
  symlink("dir1/dir2","dir1/sym");
  struct dirent* dp;
  DIR* dirp = opendir("dir1");
  while ((dp = readdir(dirp)) != NULL) {
      puts(dp->d_name);
  }
  closedir(dirp);
  return 0;
}
```

A. 4 lines are printed: `"."` `".."` `"dir2"` and `"sym"`
B. 2 lines are printed: `"dir2"` and `"sym"`
C. 3 lines are printed: `".."` `"dir2"` and `"sym"`
D. 3 lines are printed: `"."` `"dir2"` and `"sym"`
E. 3 lines are printed: `"."` `".."` and `"dir2"`

38/1. (3 points) Which response best describes when the program below will terminate?

```
10  int main() {
11     sigset_t m, old;
12     sigfillset(&m);
13     sigprocmask(SIG_SETMASK,&m, &old);
14     raise(SIGINT);
15     sigprocmask(SIG_SETMASK,&old, &m);
16     raise(SIGINT);
17     exit(0);
18     return 0;
19  }
```

A. The program will terminate when line 16 is executed
B. The program will terminate when line 15 is executed
C. The program will terminate when line 14 is executed
D. The program will terminate when line 17 is executed
E. None of the other responses are correct

38/2. (3 points) Which response best describes how many of the following implements about `sigaction()` compared to `signal()` are correct?
"`signal` requires less code than `sigaction` to set a signal handler"
"In a multi-threaded program `sigaction` is preferable to `signal`"
"With `sigaction`, additional signals can be blocked for the duration of the signal handler"
"`sigaction()` must be used to set a handler for `SIGKILL`"

A. Three (3) statements are correct
B. Four (4) statements are correct

C. Two (2) statements are correct

D. One statement is correct

E. None of the statements are correct

The last statement is incorrect. All other statements are correct. Neither `sigaction` or `action` can be used to catch `SIGKILL`.

39/1. (3 points) Four students write similar programs but with different code at line 14 (shown below). Decide if any or all of the variations shown below terminate the program.

```
10 int main() {
11  pthread_t tid1, tid2;
12  tid1 = pthread_self();
13  pthread_create(&tid2, NULL, func, NULL);
14  _______

Student A:  pthread_kill( tid2, SIGKILL );
Student B:  kill( getpid(), SIGKILL );
Student C:  raise( SIGKILL );
Student D:  pthread_kill( tid1, SIGKILL );
```

A. All four programs responses will terminate the program

B. Student A's version will not terminate the program

C. Student B's version will not terminate the program

D. Student C's version will not terminate the program

E. Student D's version will not terminate the program

40/1. (3 points) How long will it take to completely download and display a small web page with 3 tiny images from the same web server under the following assumptions?
The round-trip time is 10ms. The multi-threaded web server supports only HTTP1.0 spec (i.e. no persistent connections). The client is multi-threaded and supports up to 4 simultaneous connections. The DNS lookup (IP address of the server) is already cached by the client. Each item can be sent as a small, single network packet. Client and server processing time is minimal.

A. 40 ms

B. 80 ms

C. 20 ms

D. 50 ms

E. 60 ms

41/1. (3 points) Which of the following is NOT shared between threads in the same process ?

A. pending signals

B. signal disposition

C. signal mask

D. Open file descriptors

E. Memory mapped files

42/1. (3 points) Complete the following to create shared memory between a parent and child process that is NOT backed by a file. The shared memory must be large enough to hold one integer and be usable for IPC (Interprocess communication)

```
int* addr = __________;
*addr=42;
pid_t child = fork();
```

A. `mmap (0, sizeof(int), PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANON, -1, 0)`

B. `mmap (0, sizeof(int), PROT_READ , MAP_SHARED , -1, 0)`

C. `mmap (0, 1, 0, MAP_SHARED , -1, 0)`

D. `smem(1, S_CHILD)`

E. `openmem(sizeof(int), S_PARENT|S_CHILD, -1)`

43/1. (3 points) Choose the best response. In an ext-based inode filesystem, a directory entry contains ....

A. filename and corresponding inode numer

B. filename, file size in bytes and modification time

C. filename, number of disk blocks, owner and access information

D. just an inode number (the filename is stored as part of the inode entry)

44/1. (3 points) Select the best response. After a context switch to another process it is likely that ...

A. Physical address calculations will need a full page table lookup

B. Physical address calculation will be satisfied by the TLB

C. Shared memory must be reloaded ("paged-in") from disk

D. Cached disk Inode information must be reloaded ("paged-in") from disk

E. None of the other responses are correct

45/1. (3 points) Our software buys stocks based on analyzing real-time short tweet messages. You modify the software to query the remote server for the most recent tweet using a UDP connection. I use the original software which requires a new TCP connection for each query. The round-trip time is 10ms for all types of packets. Assuming there are no packets lost, who will have an advantage? Select the best response. Assume the remote server does not push information; you must send a request first.

A. Your UDP implementation will get tweet information 10ms earlier than my TCP solution

B. My TCP implementation will get tweet information 10ms earlier than my UDP solution

C. Your UDP implementation will get tweet information 15ms earlier than my TCP solution

D. Your UDP implementation will get tweet information 20ms earlier than my TCP solution

E. None of the other responses are correct

46/1. (3 points) Identify the missing code at positions X,Y, and Z to create an unnamed pipe and write one byte into the pipe.

```
int fd[ _X_ ];
___Y___(fd);
// later...
write( fd[ _Z_ ] , "!",1);
```

A. `X:2 Y:pipe Z:1`

B. `X:2 Y:pipe Z:0`

C. `X:2 Y:mkfifo Z:1`

D. `X:1 Y:open Z:0`

E. None of the other responses are correct

Zone 7

End of Exam Checklist:
1. Did you correctly bubble in your i) 6 letter exam key ii) netid iii) uin ?
2. Did you correctly write on the front page i) Your full name ii) netid iii) uin ?
3. Did you write your solutions to 1-5 on the Free-Response sheet?
4. Did you bubble in all 46 multiple choice responses on your Scantron?
5. Hand in your Scantron with the answer-side on top, and this printed booklet underneath with your name on top.

Zone 8

An incomplete, non-exhaustive list of useful library and system calls covered in CS241. For brevity, const and restrict keywords not shown. In written answers you may safely shorten pthread calls and macros, provided it is unambiguous to the grader. e.g. You may write `p_m_t lock = P_M_I` instead of `pthread_mutex_t lock = PTHREAD_MUTEX_INITALIZER`

```
void *memcpy(void *dest, void *src, size_t n)
void *memset(void *b, int c, size_t len)
char *strcpy(char *dest, char *src)
char *strcat(char *dest,char*src)
char *strncpy(char *dest,char*src, size_t n)
char *strncat(char *dest,char*src, size_t n)
int strcmp(char *s1, char *s2)
int strncmp(char *s1, char *s2, size_t n)
void *calloc(size_t nmemb, size_t size)
void *malloc(size_t size)
void free(void *ptr)
void *realloc(void *ptr, size_t size)
pid_t fork()
char * getenv(char *name)
int execve(char *path, char *argv[], char *envp[])
int execl(char *path, char *arg0, ... ) /* arg0 will be process name. Last arg must be (char*)0 */
pid_t getpid()
pid_t getppid()
int kill(pid_t pid, int sig) /* SIGINT, SIGKILL,SIGALRM ... */
pid_t wait(int *stat_loc)
pid_t waitpid(pid_t pid, int *stat_loc, int options) WIFEXITED ,WIFSIGNALED,WEXITSTATUS. options=WNOHANG
WIFEXITED(status) returns True if the process terminated normally by a call to _exit(2) or exit(3).
WIFSIGNALED(status) returns True if the process terminated due to receipt of a signal.
WEXITSTATUS(status) If WIFEXITED(status) is true, evaluates to the low-order 8 bits of the process's exit value.
int pthread_join(pthread_t thread, void **value_ptr)
int pthread_create(pthread_t * thread,  pthread_attr_t * attr, void *(*start_routine)(void *), void * arg)
int pthread_kill(pthread_t thread, int sig)
void pthread_exit(void *value_ptr)
pthread_mutex_t /* PTHREAD_MUTEX_INITIALIZER */
int pthread_mutex_lock(pthread_mutex_t *mutex)
int pthread_mutex_trylock(pthread_mutex_t *mutex)
int pthread_mutex_unlock(pthread_mutex_t *mutex)
int pthread_mutex_destroy(pthread_mutex_t *mutex)
pthread_cond_t /* PTHREAD_COND_INITIALIZER */
int pthread_cond_init(pthread_cond_t * cond, pthread_condattr_t * attr)
int pthread_cond_wait(pthread_cond_t * cond, pthread_mutex_t * mutex)
int pthread_cond_signal(pthread_cond_t *cond)
int pthread_cond_broadcast(pthread_cond_t *cond)
int pthread_cond_destroy(pthread_cond_t *cond)
int sem_init(sem_t *sem, int pshared, unsigned int value)
int sem_wait(sem_t *sem)
int sem_trywait(sem_t *sem)
int sem_post(sem_t *sem)
int sem_destroy(sem_t *sem)
int stat(char *path, struct stat *buf)  /* S_ISREG(mode), S_ISDIR(mode), S_ISLNK(mode) */
int lstat(char *path, struct stat *buf)
struct stat {
   dev_t     st_dev;     /* ID of device containing file */
   ino_t     st_ino;     /* inode number */
   mode_t    st_mode;    /* protection */
   nlink_t   st_nlink;   /* number of hard links */
   uid_t     st_uid;     /* user ID of owner */
   gid_t     st_gid;     /* group ID of owner */
   off_t     st_size;    /* total size, in bytes */
   ... };
int open(char *pathname, int flags) /*flags = O_RDONLY,O_WRONLY,O_RDWR,O_CREAT */
int open(char *pathname, int flags, mode_t mode) /*mode=octal or S_IWUSR,S_IXGRP,S_IROTH...*/
int pipe(int fds[2]) /* Write to fds[1], read from fds[0]*/
```

```
ssize_t read(int fildes, void *buf, size_t nbyte)
ssize_t write(int fildes,  void *buf, size_t nbyte)
int close(int fd)
int dup2(int oldfd, int newfd) /* An existing fd with value newfd will be closed first */
int accept(int socket, struct sockaddr * address, socklen_t * address_len) /* address,address_len can be null */
int listen(int socket, int backlog)
int socket(int domain, int type, int protocol)
int connect(int socket, const struct sockaddr *address, socklen_t address_len)
int bind(int socket,  struct sockaddr *address, socklen_t address_len)
ssize_t recv(int socket, void *buffer, size_t length, int flags)
ssize_t recvfrom(int socket, void *buffer, size_t length, int flags, struct sockaddr * address, socklen_t * address_len)
ssize_t send(int socket, const void *buffer, size_t length, int flags)
ssize_t sendto(int socket, const void *buffer, size_t length, int flags, struct sockaddr *dest_addr, socklen_t dest_len)
int getaddrinfo( char *hostname,  char *servname, struct addrinfo *hints, struct addrinfo **res)
void freeaddrinfo(struct addrinfo *ai)
struct addrinfo {/* When used as a hint, unused entries should be zero/null */
   int ai_flags;           /* eg. AI_PASSIVE, AI_NUMERICSERV,AI_NUMERICHOST  */
   int ai_family;          /* eg. AF_INET, AF_INET6,PF_UNSPEC... */
   int ai_socktype;        /* eg. SOCK_DGRAM, SOCK_STREAM, SOCK_RAW */
   int ai_protocol;        /* eg. IPPROTO_UDP or IPPROTO_TCP */
   socklen_t ai_addrlen;   /* length of socket-address */
   struct sockaddr *ai_addr; /* socket-address for socket */
   char *ai_canonname;     /* canonical name for service location */
   struct addrinfo *ai_next; /* pointer to next in list */
}
typedef void (*sighandler_t)(int)
sighandler_t signal(int signum, sighandler_t handler)
int sigaction(int signum,    struct sigaction *act, struct sigaction *oldact)
struct sigaction {
  void     (*sa_handler)(int);
  void     (*sa_sigaction)(int, siginfo_t *, void *);
  sigset_t   sa_mask;
  int        sa_flags;}
int sigprocmask(int how, sigset_t *set, sigset_t *oldset) /*how=SIG_BLOCK,SIG_UNBLOCKSIG_SETMASK*/
int pthread_sigmask(int how, sigset_t *set, sigset_t *oldset)
int sigemptyset(sigset_t *set)
int sigfillset(sigset_t *set)
int sigaddset(sigset_t *set, int signo)
FILE *fopen(char *path,char *mode) /*mode=a,a+,w,w+,r,...*/
int feof(FILE *stream) /* e.g. stdin,stdout,stderr */
int ferror(FILE *stream) int fflush(FILE *stream) int fclose(FILE*stream)
size_t fread(void * ptr, size_t size, size_t nitems, FILE * stream)
int fseek(FILE *stream, long offset, int whence) /*whence=SEEK_SET, SEEK_CUR, or SEEK_END*/
int lseek(int fd, off_t offset, int whence) /*whence=SEEK_SET, SEEK_CUR, or SEEK_END*/
long ftell(FILE *stream)
int fgetpos(FILE *stream, fpos_t *pos) int fsetpos(FILE *stream, fpos_t *pos) void rewind(FILE *stream)
int fclose(FILE *fp)
ssize_t getline(char **  linep, size_t *  linecapp, FILE *  stream)
char *fgets(char *s, int size, FILE *stream)
DIR *opendir(char *name)
int closedir(DIR *dirp)
struct dirent *readdir(DIR *dirp);
int readdir_r(DIR *dirp, struct dirent *entry, struct dirent **result)
struct dirent {
  ino_t          d_ino;       /* inode number */
  char           d_name[256]; /* filename */  /* Other entries not shown */
}
void * mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset)
prot= PROT_READ | PROT_WRITE |PROT_EXEC,  flags= MAP_ANON|MAP_PRIVATE|MAP_SHARED
int munmap(void *addr, size_t len)
```

NAME

strpbrk - search a string for any of a set of bytes

SYNOPSIS

```
#include <string.h>

char *strpbrk(const char *s, const char *accept);
```

DESCRIPTION

The strpbrk() function locates the first occurrence in the string s of any of the bytes in the string accept.

RETURN VALUE

The strpbrk() function returns a pointer to the byte in s that matches one of the bytes in accept, or NULL if no such byte is found.