# CS 351 Sample Final Exam, Spring 2015

Before starting the exam, **make sure that you write and bubble in your student ID number** (omitting the leading 'A') on the answer sheet in the field labeled "Student Number".

For each question, choose the **single best answer** and fill in the corresponding bubble on the answer sheet. Feel free to scribble on this question paper, but be sure to turn it in at the end of the exam!

1. Which system call may result in the deallocation of an open file description object?
   (a) `open`
   (b) `read`
   **(c) `exit`**
   (d) `dup`

2. Which piece of information is *not* maintained within the v-node structure?
   (a) file permissions
   (b) data location (e.g., on disk)
   (c) file size
   **(d) current file position**

3. What is a limitation of unnamed pipes?
   (a) a hard limit of 2 processes that can read/write them at any time
   (b) their lack of synchronization when it comes to IPC
   (c) the loss of data stored in them after the writing process terminates
   **(d) not being able to use one for IPC with a process that doesn't already have a reading/writing end**

4. What is a primary incentive for implementing user-level buffered I/O (on top of the I/O system calls)?
   **(a) reducing the number of I/O system calls**
   (b) eliminating the possibility of short counts
   (c) supporting the standard input/output/error abstraction
   (d) allowing processes to share files without going through the kernel

5. What is a strong argument against using regular files for dynamic IPC?

   (a) separate processes cannot simultaneously read/write regular files

   (b) regular files are more prone to short counts (than purpose-built IPC mechanisms)

   **(c) coordinating separate (e.g., read/write) positions in regular files is tricky**

   (d) data stored in regular files does not persist after a process exits

6. For this and the next question, consider the following function, `read_write`, which makes use of the buffered stdio function `fread`:

```
void read_write(FILE *stream, int n) {
    char buf[100];
    /* read n bytes from stream into buf */
    int nread = fread(buf, 1, n, stream);
    /* print bytes read to stdout */
    write(1, buf, nread);
}
```

Note that `fread` takes three arguments in addition to the destination buffer:

- the size of each item to read (1 byte, in the given invocation)
- the number of items to read (`n`)
- the stream from which to read the items

Assume that stream buffers are 4KB large.

Now consider the following program, which contains two separate calls to read_write:

```
main() {
   FILE *infile = fopen("foo.txt", "r");
   if (fork() == 0) {
       read_write(infile, 6);
   } else {
       wait(NULL);
       read_write(infile, 6);
   }
}
```

Given that the file "foo.txt" contains the single line of text:

```
Lorem ipsum dolor sit amet
```

What would be the output of the above program (from parent and child processes combined)?

(a) **`Lorem`**

(b) `Lorem Lorem`

(c) `Lorem ipsum`

(d) No output is produced

7. Based on the same `read_write` function and "foo.txt" file from the previous problem, we make a minor modification to the program as show below, "priming" the stream with yet another call to `read_write`:

```
main() {
    FILE *infile = fopen("foo.txt", "r");
    read_write(infile, 6); /* initial read/write */
    if (fork() == 0) {
        read_write(infile, 6);
    } else {
        wait(NULL);
        read_write(infile, 6);
    }
 }
```

What is the output of this program?

(a) `Lorem ipsum`

(b) `Lorem ipsum dolor`

(c) **`Lorem ipsum ipsum`**

(d) `Lorem ipsum Lorem`

8. The following program makes use of an unnamed pipe to facilitate communication between two processes.

```
int main() {
    int i, n, pfds[2];
    char c, buf[80];
    pipe(pfds);
    if (fork() == 0) {
        for (c='1'; c<='5'; c++) {
            write(pfds[1], &c, 1);
        }
    } else {
        close(pfds[1]);
        while ((n = read(pfds[0], buf, sizeof(buf))) > 0) {
            write(1, buf, n);
            write(1, "-", 1);
        }
    }
}
```

Which of the following is *not* a possible output of the program?

(a) 12345-

(b) **34512-**

(c) 1-2-3-4-5-

(d) All the above are possible

9. Which of the following caching policies requires the addition of a *dirty bit* to each cache line?

(a) write-through

(b) write-around

(c) write-allocate

(d) **write-back**

10. Which of the following combinations of write caching policies allow for *write absorption* at the cache level?

    (a) write-through + write-around

    (b) write-through + write-back

    (c) **write-back + write-allocate**

    (d) write-back + write-around

11. For this and the next three questions, consider the following function which takes pointers to two non-overlapping arrays (`arr1`, `arr2`) of random, word-sized elements, and the number (`n`) of elements in each to be processed:

```
int foo (int *arr1, int *arr2, int n)
{
    int i, accum;

    for (i=0; i<n; i++)   /* loop #1 */
        accum += arr1[i];

    for (i=0; i<n; i++)   /* loop #2 */
        accum += arr2[(2*i) % n];

    for (i=0; i<n; i++)   /* loop #3 */
        accum += arr1[i] * arr2[i]

    return accum;
}
```

    Make the following assumptions:

    - the cache is 2-way set associative with 2-word blocks
    - all local variables (excluding arrays) are mapped to registers by the compiler
    - data in `arr1` and `arr2` are uncached before `foo` is called

    What is the approximate *best case* hit rate for loop #1?

    (a) 0%

    (b) 25% (¼)

    (c) **50% (½)**

    (d) 100%

12. What is the approximate *best case* hit rate for loop #2?

    (a) 0%

    (b) 25% (¼)

    (c) **50% (½)**

    (d) 100%

13. What is the approximate *best case* hit rate for loop #3 (given that it was preceded by loops #1 and #2)?

    (a) 0%

    (b) 25% (¼)

    (c) 50% (½)

    (d) **100%**

14. What is the approximate *worst case* hit rate for loop #3 (given that it was preceded by loops #1 and #2)?

    (a) 0%

    (b) 25% (¼)

    (c) **50% (½)**

    (d) 100%

15. What is a primary justification for enforcing alignment of data in memory?

    (a) **improving cache utilization**

    (b) reducing memory fragmentation

    (c) improving memory (DRAM) utilization

    (d) simplifying compilation

16. What is a primary justification for making use of the lower bits of an address to compute the cache index?

    (a) improving cache utilization

    (b) to better leverage good temporal locality

    (c) to better leverage good spatial locality

    (d) **to reduce the likelihood of cache collisions**

17. A relatively large miss penalty at a given level of caching is good justification for which of the following?

    (a) implementing the write-around policy

    (b) **increasing associativity to improve hit rate**

    (c) using a simple cache design to optimize the hit time

    (d) implementing the write-through policy

18. Given a fixed cache (data) size, what strategy best deals with the problem of *cache thrashing*?

    (a) decreasing the block size

    (b) increasing the block size

    (c) decreasing associativity

    (d) **increasing associativity**

19. (See slide 86 of Memory Hierarchy & Caching slides for examples of direct-mapped cache lookup exercises)

20. (See slide 109 of Memory Hierarchy & Caching slides for examples of associative cache lookup exercises)