Princeton University
COS 217: Introduction to Programming Systems
Spring 2006 Final Exam Answers

The exam was a three-hour, open-book, open-notes exam.

Question 1

(a) TRUE - getting the data from a register will be generally faster than fetching it from memory.

(b) FALSE - stack frame size depends on the local variables, etc.

(c) FALSE - race conditions are generally unintended, and make programs incorrect.

(d) TRUE - assuming it's a 4-byte int, etc. It has the effect of also initializing it to zero, but given that subl will make the value uninitalized, this isn't a correctness problem.

(e) FALSE - const variables can be overridden by force. If the OS protects roData, the compiler won't be able to force a write to data that resides there.

Question 2

(a) A relocation record is an entry in a table generated by the assembler that contains the global declarations for this particular file, as well as the locations of uses of globals from other files. These entries indicate patches that the linker must perform.

(b) Some parts of the code cannot be resolved by just the assembler, and must wait until all of the files are seen by the linker. Once the linker lays out the program, it uses the relocation record to fix up the unresolved references.

(c) Any scenario where the CPU may be spending lots of time doing other things. For example, if you want to profile a CPU-intensive program running on a machine with other CPU-intensive processes, you should use virtual time.

(d) Scenarios where CPU consumption of one process doesn't tell the whole story - for example, a process that's slow due to large amounts of disk access may show low virtual time, but users may care about wall-clock instead.

(e) So that a user overrunning a buffer is less likely to corrupt the library's bookkeeping structures. Lots of people said that this makes allocation easier or lowers fragmentation. Those are independent of this decision - modern allocators may use power-of-two allocation sizes for the reasons above, and they may split bookkeeping and buffers, but that's just because each have their own good points.

Question 3

(a) ebx

(b) Checking to see if it's hit the terminating NUL character, and ending the loop if so.

(c) It fails to take the branch if the character being compared was originally between A and Z inclusive. More precisely, it checks to see if the actual value, after subtractions, is greater than 25, since anything before 'A' becomes a large positive, and anything after Z is a small number after 25.

(d) Prints all of the uppercase letters in the string.

Question 4

(a) Hit ctrl-z and kill it via the kill command, or hit ctrl-backslash.

(b)

1. Should re-set the signal handler, or use something other than signal().

[Dondero note: With some old-style signal handling implementations, when a signal of type X is received and its handler function begins to execute, the handler function installation for signals of type X is cancelled. Thus the next signal of type X causes the default action to occur. So, when using such old-style implementations, the handler function must be reinstalled within the handler function itself. On the other hand, if the program contains the preprocessor directive "#define _GNU_SOURCE" before including any .h file, then the program will use the new-style signal handling implementation. With the new-style implementation, when a signal of type X is received, the handler function installation for signals of type X is **not** cancelled. So, when using new-style implementations, the handler function need not be reinstalled within the handler function itself.]

2 Either use write() instead of printf(), or flush(stdout), since the output of this may be fed to something else.

[Dondero note: When stdout is bound to the /dev/tty device (that is, the terminal), then inserting a newline character into the stdout buffer will flush the buffer automatically. That's not necessarily the case when stdout is bound to some other device.]

3. Get rid of the semicolon after while, since it's preferable to sleep rather than spin wildly and eat all the free CPU.

4. Should check if signal got installed correctly by examining return value.

5. Should check to see if we're already ignoring SIGINT before overriding it.

[Dondero note: That subtle point was covered during the Spring 2006 semester. It is not covered in some other semesters.]

6. Should block signals in handler to avoid race condition. Just making the number of signals local, or just increasing numSigs does not fix the race condition.

[Dondero note: With new-style signal handling implementations, a signal of type X automatically is blocked when the handler for signals of type X is executing. With some old-style implementations, that is not the case. With such implementations the possibility of a race condition exists, and so signals explicitly should be blocked in the handler. From that point of view, item 6 is an acceptable answer.]

Note that not returning a value in main is not necessarily a problem - a good compiler should complain about unreachable code if you try to add a return value. Likewise, by not including a return value, if someone comes along and changes the program incorrectly, they'll get a warning about the missing return value, which might tip them off that they did something wrong.

Question 5

(a)

Code Pro: can customize it however you want, profile it easily, etc.

Code Con: less likely to be tested, harder for anyone else to use.

Library Pro: easy for other people to use it, pressure to make it more general, likely to be better tested if in widespread use.

Library Con: may not be perfectly tailored for application, generality may miss optimization opportunities, use might stomp library data, causing problems.

OS Pro: protection from applications, ability to perform privileged operations if needed.

OS Con: higher per-call overhead, any security flaw can harm entire system

(b)

Example of bad invocation: maximum = MAX(a++, b--)

The arguments to a macro don't have the same type constraints that functions do, so you'd have to have versions of the function that handle ints, floats, etc. Also, in any place where MAX was used in a performance-critical way, you should make sure the higher cost of a function call won't be a problem.

(c)

This is straight from lecture, with one field name changed. The method of allocation is malloc(sizeof(Entry) + strlen(name)), and the name is the strcpy'd into e name, which is going beyond the defined space of one character. This is valid C, because even though the precise layout in memory of fields within a structure is not guaranteed, the order is preserved. As a result, the extra allocated space will come after e_name, and going beyond the end of the e_name array will not cause a problem.

The reason for this unusual allocation process is that it can conserve memory for small allocations, and even improve performance. Assume most names are in the vicinity of 10-16 characters. If the malloc library rounds all allocs up to 32 bytes minimum, then doing the alloc in this manner will use only one minimum-sized alloc for both the name and the other fields. In contrast, making e_name a char pointer and using a separate alloc for it will yield two minimum-sized allocs in most cases. Additionally, this mechanism also requires a separate pointer dereference to get the name, which may be at some other region in memory.

(d)

Any answer is correct for this question.