CSE 410 Final Exam 6/05/12 Sample Solution

Question 1. (10 points) (bits) An array of boolean (0/1) values can be stored very compactly if we store 8 bits of the array in each byte. For instance here is an array of 24 bits stored in 3 bytes.

0	0	0	0	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	0	0	0	1
23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				2							1									0			

In this array, bits 0, 7, 11-14, and 17 are 1 and all the rest are 0. The bits of the array are numbered from right to left. The bytes holding the bits are also numbered right to left, as shown by the numbers in the last row.

For this problem, complete the following C function so it returns the value of the specified bit in the array. You may assume that parameter `index` has an appropriate value (i.e., you don't need to worry about what to do if it is negative, or larger than the array size, or too large to be stored in a signed `int` variable). The array pointer `b` has type `char *`, which means that it points to an array of bytes.

Hint: shifts, `&`, etc. Suggestion: break your code into a few small assignment statements to make it easier to follow.

```
/* return the value of the bit whose location in */
/* bit array b is given by the variable index.   */
int getBit(char * b, int index) {
   int byteOffset = index >> 3;
   int bitOffset = index & 0x7;
   return 1 & (b[byteoffset] >> bitoffset);
}
```

Notes:

- **This question was intended to be an exercise in bit manipulation, as in Lab 1. We should have restricted the available operations to shifts, logical operators like `&` and `|`, and so forth, but since we forgot to say so explicitly we allowed credit for solutions that used things like division and mod.**

- **Obviously the solution can be written as a 1-liner, which is fine.**

Question 2. (10 points) (some mystery code, or the ghosts of midterms past)

Once again one of the interns has lost the source code to an important function. We have been able to discover that the function starts and ends like this:

```
int g(int x, int y) {
  int ans;
   …
  return ans;
}
```

But beyond that, all we've been able to find is some output produced using gdb. When we disassemble the code, here's what we get:

```
(gdb) disas g
Dump of assembler code for function g:
   0x0000000000400474 <+0>:      push   %rbp
   0x0000000000400475 <+1>:      mov    %rsp,%rbp
   0x0000000000400478 <+4>:      mov    %edi,-0x14(%rbp)
   0x000000000040047b <+7>:      mov    %esi,-0x18(%rbp)
   0x000000000040047e <+10>:     cmpl   $0x6,-0x14(%rbp)
   0x0000000000400482 <+14>:     ja     0x4004b4 <g+64>
   0x0000000000400484 <+16>:     mov    -0x14(%rbp),%eax
   0x0000000000400487 <+19>:     mov    0x4005d8(,%rax,8),%rax
   0x000000000040048f <+27>:     jmpq   *%rax
   0x0000000000400491 <+29>:     movl   $0x1,-0x4(%rbp)
   0x0000000000400498 <+36>:     jmp    0x4004bb <g+71>
   0x000000000040049a <+38>:     mov    -0x18(%rbp),%eax
   0x000000000040049d <+41>:     mov    %eax,-0x4(%rbp)
   0x00000000004004a0 <+44>:     jmp    0x4004bb <g+71>
   0x00000000004004a2 <+46>:     movl   $0x7,-0x4(%rbp)
   0x00000000004004a9 <+53>:     jmp    0x4004bb <g+71>
   0x00000000004004ab <+55>:     movl   $0x6,-0x4(%rbp)
   0x00000000004004b2 <+62>:     jmp    0x4004bb <g+71>
   0x00000000004004b4 <+64>:     movl   $0x0,-0x4(%rbp)
   0x00000000004004bb <+71>:     mov    -0x4(%rbp),%eax
   0x00000000004004be <+74>:     pop    %rbp
   0x00000000004004bf <+75>:     retq
```

And a look at the memory locations apparently referenced in the code shows this:

```
(gdb) x /10gx 0x4005d8
0x4005d8:  0x00000000004004b4    0x000000000040049a
0x4005e8:  0x000000000040049a    0x0000000000400491
0x4005f8:  0x00000000004004a2    0x00000000004004ab
0x400608:  0x00000000004004a2    0x0000002c3b031b01
0x400618:  0xfffffe6400000004    0xfffffeb000000048
```

(continued next page – feel free to remove this page for reference)

Question 2 (cont.) In the space below, translate the assembly language function given on the previous page into C. The function heading and return statement at the end are written for you.

```
int g(int x, int y) {
  int ans;

  switch(x) {

  case 3:

    ans = 1;

    break;

  case 1: case 2:

    ans = y;

    break;

  case 4: case 6:

    ans = 7;

    break;

  case 5:

    ans = 6;

    break;

  default:

    ans = 0;

  }

  return ans;
}
```

Question 3. (10 points) (buffers) Consider the following function, which calls the same `Gets` function used in the buffer overflow lab to read a sequence of bytes.

```
int f() {
  int x, y;
  char s[2];
  x = 1;
  y = x+1;
  Gets(s);
  return y;
}
```

When this function was assembled on an x86-64 machine, it produced the following assembly code:

```
f: pushq   %rbp
   movq    %rsp, %rbp
   subq    $16, %rsp
   movl    $1, -4(%rbp)
   movl    -4(%rbp), %eax
   addl    $1, %eax
   movl    %eax, -8(%rbp)
   leaq    -16(%rbp), %rax
   movq    %rax, %rdi
   call    Gets
   movl    -8(%rbp), %eax
   leave
   ret
```

Your job is to create a string of bytes to be read by `Gets` that will cause this function to return the value 5 instead of the value it ordinarily returns, which is 2.

You should give your answer as a string of hex digits giving the byte values for the input in the same format used as input to `sendstring` in lab 3, i.e., a pair of hex digits for each byte, like `31 32 33`.

Hint: you will find it useful to sketch the layout of the stack frame and the variables in it to decide how many bytes your exploit string needs and what the contents should be.

Here's an exploit string that will work:

```
01 02 03 04 05 06 07 08 05
```

[The exploit string needs 8 bytes of padding, which can contain any values, followed by a single byte containing `0x05`. That's enough, or the string can also contain up to 6 trailing bytes, but no more. If additional bytes are supplied after the `0x05`, the first ones need to be `0x00` so the value of y is set to `0x00000005`. (Also, remember that a trailing `0x00` byte will be provided by `Gets`).]

Question 4. (10 points) (caches) You are not required to show your work on these questions, but showing some work might help if it is necessary to award partial credit.

(a) Suppose we have a 512 byte, 2-way set associative cache with a block size of 32 bytes. How many sets (rows) are there in this cache?

8

[Each set contains two blocks of 32 bytes. That's 64 bytes per row.]

(b) Suppose we have a memory system with a single level cache. The cache has an access time of 2ns and main memory has an access time of 200ns. The average hit rate is 98%. What is the effective access time of this memory system?

6 ns (= 2 + 0.02*200)

[Every reference costs 2ns for the cache access; misses have an additional 200ns cost to fetch the data from memory.]

(c) True or false: Increasing the associativity of a cache is likely to reduce conflict misses.

True

(d) True or false: Even a small decrease in the miss rate of a cache can result in a dramatic performance increase.

True

Question 5. (8 points) (Hit or miss?)

Suppose we have a direct-mapped cache containing 128 bytes total with 16-byte cache blocks. What is the miss rate of the following code?

```
int x[2][32];
int i;
int sum = 0;

for (i = 0; i < 32; i++) {
   sum += x[0][i] * x[1][i];
}
```

Assumptions:

- The cache is initially empty.
- Array `x` begins at memory address `0x0` and is stored in row-major order.
- All variables and code other than the array `x` itself are stored in registers (i.e., they do not affect the cache).
- Integers occupy 4 bytes each.

Miss rate is 100%.

Reason (not required): the cache has 8 rows that hold 4 `int` values each. That is sufficient to hold exactly one row of the array, but not both. Variables `x[0][i]` and `x[1][i]` map to the same location in the cache for each possible value of i. Every reference to an array element will be either a cold miss or a capacity miss.

Question 6. (8 points) (Virtual memory) Suppose we have a system with 32-bit virtual addresses, 20-bit physical (real memory) addresses, and 8K pages.

(a) How many bits are included in the virtual page number and page offset parts of a virtual address on this system?

Virtual page number bits ________**19**__________

Page offset bits ________**13**____________

(b) How many bits are included in the physical page number part of a physical (real memory) address?

Physical page number bits ________**7**__________

(c) How many page table entries (PTEs) are needed for this virtual memory system?

Number of PTEs ____________**2^{19} = 512K**_____________

Question 7. (3 points) (VM again) One of the reasons a virtual memory system may perform poorly is if it starts *thrashing*. **Very** briefly, what does this mean?

Thrashing occurs when the combined working sets (active memory pages) of all active processes are too large to fit in physical memory. That means that page faults are likely to occur frequently, and each page fault will evict a page that is actively being used, which is almost certain to cause another page fault quickly. The amount of paging I/O increases dramatically and the system gets little work done because each process spends most of its time waiting for pages to be brought in from disk.

Question 8. (4 points) (something exceptional) After the operating system handles an exception, the system can either resume execution of the interrupted process at the instruction following the one that caused the exception, resume execution by re-executing the instruction that caused the exception, or terminate the process. Below is a list of possible exceptions. For each one, indicate by writing a, b, or c what the OS would normally do after handling the exception.

(a) Continue execution at the instruction following the one that generated the exception.

(b) Continue execution by re-executing the instruction that caused the exception.

(c) Terminate the process

___**c**___ Division by 0

___**b**___ Process causes a page fault

___**c**___ Memory reference out of bounds (e.g., segfault)

___**a**___ System call (`int`) instruction executed by process to read data

Question 9. (3 points) In the classic Unix file system, a directory is a special file containing one entry for each file in that directory. What information is stored in the directory entry itself along with the file name? Place an X next to all of the following that are found in the directory entry, and only check items that are included in the directory file itself as opposed to elsewhere on the disk. Leave all the answers blank if none of them apply.

_____ disk address of first file data block

__**X**__ inode number of the file

_____ full path name of the file (e.g., `/a/b/c`)

Question 10. (6 points) (OS hardware support) Some of the features included in modern processors were introduced because they were needed to implement multi-processing operating systems. For each of the following, put an X in the blank space if the feature is included to provide support for operating systems. Leave the entry blank if it is an ordinary feature of the instruction set not specific to operating system support.

_____ condition code registers

__X__ system call (`int`) instruction

__X__ privileged execution mode

_____ call instruction

_____ stack pointer (sp) register

__X__ interval timer (can be set to cause an interrupt/exception after a short time)

Question 11. (6 points) (OS data structures) Which of the following are items that are likely to be found in the Process Control Block (PCB) used by the operating system to describe a process? Place an X in the blank space if it is something that would appear in a PCB, leave the entry blank if it is not.

__X__ Current process state (running, waiting, etc.)

_____ Current time of day

__X__ CPU time used by process

__X__ Link to next PCB on the same linked list (ready or waiting queue, for example)

_____ Contents of the process' virtual memory page tables

__X__ Process program counter if the process is not running.

Question 12. (12 points) (Compare and contrast☺) For each of the following pairs of terms, give a ***very* brief** explanation of the key difference between the two terms or concepts. "Very brief" = a sentence or two that shows you understand the key ideas.

(a) program, process

A process is the active, dynamic execution of a program, which is a static entity.

(b) write-through, write-back

Write-through: changes to a cache block or page frame are immediately written to the backing store (main memory or disk)

Write-back: changes are made only to the cached copy of the data, and not written to the backing store until the cache block or physical page frame is reused for other data.

[These concepts apply equally well to caches, virtual memory, and other memory heirarchies, but it's fine if your answer only referred to a specific one.]

(c) temporal locality, spatial locality

Temporal locality: a referenced location is likely to be referenced again soon.

Spatial locality: data near to a referenced location is likely to be referenced soon.

(d) process, thread

Process: basic unit of execution, including an address space, environment data like variables and open files, and at least one thread.

Thread: a sequential execution in a process, including registers, a stack, and a program counter. A process may support one or more threads.

Question 13. (10 points) (finis) Consider the following program:

```
int main() {
  if (fork() != 0) {
    printf("good ");
  } else {
    fork();
    printf("bye ");
    printf("y'all ");
  }
  return 0;
}
```

We'd like to know what output can be produced when this program is run. If it is possible for the program to produce different output if it is run more than once, give two of the possible results. If it always produces the same output, give that output and indicate that it is unique.

There are many possible outputs. The original process prints "good" and then terminates. The child and grandchild processes each print "bye" and "y'all" in that order. These output operations can be interleaved in any sequence as long as the order of each "bye"/"y'all" pair printed by a single process is not reversed.

Some possible output sequences (which is all you were asked to write):

```
bye bye y'all y'all good

good bye y'all bye y'all
```

Have a great summer!!!