Computer Science 61

Final Exam Solutions

December 12, 2015

180 minutes/180 points

Instructions

The points allocated to each problem correspond to how much time we think the problem should take.

You will submit the exam via your code.seas repository and the grading server. If you do not have a current pset repository, you will need to pull the cs61-psets repository so you have the final directory. As with the problem sets, you should verify that you have set up the handout remote:

```
% git remote show handout
```

If this reports an error, run:

```
git remote add handout git://code.seas.harvard.edu/cs61/cs61-psets.git
```

Next, you should pull the exam from our repository:

```
git pull handout master
```

This will merge the final directory with your previous work; you should not run into any conflicts. After you've done this, run git push to save this merge back to your personal repository on code.seas.

Please place all answers for the exam in the file named exam.txt in the final directory. Be sure to label each question clearly, so there is no possible way we can be confused about which question you are answering. Please do **not** place your name anywhere in the exam.txt file. **When you have completed the exam, edit the file policy.txt, and attest that you have followed the rules for the exam (listed below) by writing, "I have neither given nor received help on this exam and have followed the rules as stated." Then type your name beneath it, indicating compliance.** The exam is open book, open notes, open computer. You may access the book and your own notes in paper form. You may also use a computer or equivalent to access class materials. However, you may not access non-class materials except as explicitly allowed below, and you may not write programs to answer questions. Specifically:

- You may access a browser and a PDF reader.
- You may access your own notes electronically.

- You may access an Internet site on which your own notes are stored.
- You may access system manual pages (man).
- You may access the cs61.seas.harvard.edu/wiki/2015 site (but not any prior years, even though they are linked off the site).

But, you may not access Google or Wikipedia or anything else except as directly linked from the cs61.seas.harvard.edu/wiki/2015 site. In particular:

- You may not access Piazza.
- You may not access course videos.
- You may not run a C compiler, assembler, on-line disassembler, calculator, or anything similar. Simple reading/writing applications only.
- You may not access other students' notes.
- You absolutely may not contact other humans via IM or anything like it.
- You may not access solutions from any previous exam, by paper or computer.

Any violations of this policy, or the spirit of this policy, are breaches of academic honesty and will be treated accordingly. Please appreciate our flexibility and behave honestly and honorably.

Unless otherwise stated, please make the following assumptions:

- A 32-bit little endian architecture (i.e., the appliance).
- The terms kilobyte (KB), megabyte (MB), and gigabyte (GB) refer to the binary values (i.e., 2 to some integral power) not decimal (i.e., 10 to some integral power).

---

1. Data Representation (10 points)

Write the value of the variable or expression in each problem -- use signed decimal representation.

For example, if we gave you:

A. int i = 0xA;
B. int j = 0xFFFFFFFF

you would write A) 10 B) -1

A. int i = 0xFFFF; (You may write this either in decimal or as an expression including a power of 2.) That's 2^16 - 1 or 65535
B. short s = 0xFFFF; That's -1 for a short
C. unsigned u = 1 << 10; That's 1024; I also gave credit for 2^10
D. From assignment 6: unsigned long l = PTE_P | PTE_U; 5
E. int j = ~0; That would be all 1's (binary) or all F's (hex) which is -1
F. From assignment 6: sizeof(x86_pagetable);4096 (4KB)

G.
```
struct s {
        char c;
        short s;
        long l;
};
struct s *ps;
sizeof(ps);
```

It's a pointer! Sizeof = 4

H. Using the structure above: sizeof(*ps); 8

I. unsigned char u = 0xABC; This is unsigned, so 0xBC = 188

J. char c = 0xABC; The char holds only 0xBC; the high order bit is set, so this is the negative number -0x44 or -68

2. Program Layout (10 points)

For the following questions, select the part(s) of memory from the list below that best describes where you will find the object.

1. heap
2. stack
3. between the heap and the stack
4. in a read-only data segment
5. in a text segment starting at address 0x08048000
6. in a read/write data segment
7. in a register

Assume the following code, **compiled without optimization**.

```
#include <errno.h>
#include <getopt.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

// The following is copied from stdio.h for your reference
#define EOF (-1)

1       unsigned long
2       fib (unsigned long n)
3       {
4           if (n < 2)
5               return (n);
6           return (fib(n - 1) + fib(n - 2));
7       }
8
```

```
 9      int
10      main(int argc, char *argv[])
11      {
12          extern int optind;
13          char ch;
14          unsigned long f, n;
15
16          /* Command line processing. */
17          while ((ch = getopt(argc, argv, "h")) != EOF)
18              switch (ch) {
19                  case 'h':
20                  case '?':
21                  default:
22                      return (usage());
23              }
24
25          argc -= optind;
26          argv += optind;
27
28          if (argc != 1)
29              return (usage());
30
31          n = strtoul(strdup(argv[0]), NULL, 10);
32          if (n == 0 && errno == EINVAL)
33              return (usage());
34
35          /* Now call one of the fib routines. */
36          f = fib(n);
37          printf("fib(%lu) = %lu\n", n, f);
38
39          return (0);
40      }
```

A. The string "fib(%lu) = %lu\n" (line 37).
B. optind (line 25).
C. When executing at line 4, where you will find the address to which fib returns.
D. Where will you find the value of EOF that is compared to the return value of getopt in line 17.
E. getopt (line 17)
F. fib (lines 1-7)
G. the variable f (line 36)
H. the string being passed to strtoul (line 31)
I. strdup (line 31)
J. The value of the fib function when we return from fib (line 6).

1 point each

A = 4 read-only data segment

```
B = 6 R/W data segment
C = 2 the stack
D = This one is tricky. EOF is a macro, which means the value -1
shows up in some instruction somewhere, which means the right answer
is 5 (text segment); it *may* by placed in a register, but has never
been in any of the assembly code we've read.
E = 3 between the heap and the stack
F = 5 text segment
G = 2 the stack
H = 1 the heap (The address of the string is one the stack,
but the string itself is on the heap -- the man page for strdup
tells you that it mallocs space.)
I = 3 between the heap and the stack
J = 7 register
```

3. Pot Pourri (20 points)

Questions A-D pertain to the data structures and hexdump output shown here.

```
        struct x {
                unsigned long ul;
                unsigned short us;
                unsigned char uc;
        } *sp;

// Hexdump output of some program running on the appliance.
08c1b008  e9 11 cf d0 0d d0 3f f3  63 61 74 00 0d f0 fe ca  |......?.cat.....|
08c1b018  5e ea 15 0d de c0 ad de                           |^.......|
```

You are told that sp = 0x08c1b008

A. What is the value (in hex) of sp->ul?
B. What is the value (in hex) of sp->uc?
C. At what address will you find the string "cat"?
D. You think that the bytes after the string "cat" comprise an array of 3 integers; what is the value (in hex) of the middle one of those?
E. What is the following binary value expressed in hexadecimal: 01011010?
F. What is the value of the hex number 0x7FF in decimal?
G. Is 0x98765432 a valid return from malloc?
H. What is the minimum number of x86 instruction bytes you need to write an infinite loop?
I. True or False: A declaration allocates space for an object in C?
J. True or False: Processes cannot share memory.

For questions K-O, assume we are running on the appliance and we initialize ival, p, and q as shown below. Write the value of the expression -- you may express the values in hex if that's simpler, just be sure to prefix them with 0x to make it clear that you are doing so. For True/False questions, there is no need to correct or provide a counterexample for any statements that are false.

```
int     ival[4] = {0x12345678, 0x9ABCDEF0, 0x13579BDF,0x2468ACE0};
int     *p = &ival[0];
int     *q = &ival[3];
```

K. q - p
L. ((char *)q - (char *)p)
M. int *x = p + 1; what is value of: x - p;
N. (Assuming x as set above) *((short *)((char *)x+2))
O. char *cp = (char *)(q - 2); what is the value of: *cp;
P. What system call allows you to block on a collection of file descriptors?
Q. What system call creates a communication channel that can only be used among processes with a common ancestor?
R. What system call lets you change the attributes of a file descriptor so you can poll it rather than blocking?
S. What system call produces a file descriptor on which a server can exchange messages with a client?
T. True or False: A program and a process are the same thing.

1 point each

A. 0xd0cf11e9
B. 0x3f
C. 0x08c1b010
D. 0x0d15ea5e
E. 0x5A
F. 2047
G. No, it is not a multiple of 16, which is the ALIGNOF for our 32 bit architecture. (It's not even divisible by 4). (It's also not in the heap region.)
H. 2 bytes: 0xfeeb;
I. False - the definition allocates the space; the declaration just tells you about the variable.
J. False - they can map the same physical page!

K. 3
L. 12
M. 1
N. 0x9ABC
O. 0xF0
P. select (pselect)
Q. pipe
R. fcntl
S. accept
T. False

4. Assembly and Data Structures (10 points)

Consider the following assembly function.

```
func:
        movl    4(%esp), %edx
        xorl    %eax, %eax
        cmpb    $0, (%edx)
        je      .L4
.L3:
        addl    $1, %eax
        cmpb    $0, (%edx,%eax)
        jne     .L3
        ret
.L4:
        ret
```

A. How many parameters does this function appear to have?
B. What do you suppose the type of that parameter is?
C. Write C code that corresponds to the following assembly function.

2 points for each of A-B 6 points for C

A. 1
B. char * (or unsigned char *)
C. It's strlen

```
int
len(char *c)
{
// 2 points for properly handing iterator: initialize to 0; increment in loop
// 2 points for proper loop exit (*cp = 0)
// 2 points for proper increment of the char pointer (or whatever they thought it was)
    char *cp = c;
    int n = 0;

    while (*cp != '\0') {
        n++;
        cp++;
    }
    return (n);
}
```

5. Memory and Pointers (16 points)

Two processes are mapping a file into their address space. The mapped file contains a linked list of structures, each of which contains an integer value (this list is not sorted). As the processes cannot ensure that the file will be mapped at the same virtual address, they use *relative pointers* between each element in the linked list. So, each element in the linked list is represented by the following structure:

```
struct ll_node {
        int  value;
        off_t  r_next;  /* Relative Pointer. */
};
```

The value -1 is used in the `r_next` field to indicate the end of the list.

Write a function to find an item in the list. The function's prototype is:

```
struct ll_node *find_element(void *region, struct ll_node *list, int value);
```

The `region` parameter is the address of the beginning of the region; the `list` parameter is a pointer to the first node in the list; the `value` parameter is the value for which we are searching. The function should return a pointer to the linked list element if the value appears in the list or NULL if the value is not in the list.

```
struct ll_node *
find_element(void *region, struct ll_node *list, int value)
{
        struct ll_node *lp;

        for (lp = list; lp != NULL; ) {         // 3 points for a valid loop (1 if they don't
                                                //      handle a NULL pointer for list)
                if (lp->value == value)         // 2 points for checking for value
                        break;
                if (lp->r_next == -1) {         // 3 points for handling end of list =
                        lp = NULL;              // 1 for the check; 2 for setting lp
                        break;
                }
                // 8 points total for getting the next calculation correct
                // =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
                // 3 points for understanding that you need to add region to r_next.
                // 3 points for casting region to do the addition
                // 2 points for casting the return value appropriately
                lp = (struct ll_node *)((char *)region + lp->r_next);
        }
        return (lp);
}
```

6. Teensy OS VM System (16 points)

The folks at Teensy Computers, Inc, need your help with their VM system. The hardware team that developed the VM system abruptly left and the folks remaining aren't quite sure how VM works. I volunteered you to help them.

The Teensy machine has a 16-bit virtual address space with 4 KB pages. The Teensy hardware specifies a single-level page table. Each entry in the page table is 16-bits. Eight of those bits are reserved for the physical page number and 8 of the bits are reserved for flag values. Sadly, the hardware designers did not document what the bits do!

A. How many pages are in the Teensy virtual address space?
B. How many bits comprise a physical address?
C. Is the physical address space larger or smaller than the virtual address space?
D. Write, in hex, a PAGE_OFFSET_MASK (the value that when anded with an address returns the offset of the address on a page).
E. Write a C expression that takes a virtual address, in the variable vaddr, and returns the virtual page number.

You are now going to work with the Teensy engineers to figure out what those other bits in the page table entries mean! Fortunately, they have some engineering notes from the hardware team -- they need your help in making sense of them. Each letter below has the contents of a note, state what you can conclude from that note about the lower 8 bits of the page table entries.

F. "Robin, I ran 8 tests using a kernel that did nothing other than loop infinitely -- for each test I set a different bit in all the PTEs of the page table. All of them ended up in the exception handler except for the one where I set bit 4. Any idea what this means?"
G. "Lynn, I'm writing a memory test that iterates over all of memory making sure that I can read back the same pattern I write into memory. If I don't set bit 7 of the page table entries to 1, I get permission faults. Do you know what might be happening?"
H. "Pat, I almost have user level processes running! It seems that the user processes take permission faults unless I have both bit 4 and bit 3 set. Do you know why?"

2 points each for A-E (10 points).
2 points each for F-H (6 points).
Try not to take off twice for the same thing -- that is if they miss F and then misinterpret bit 4 again, try not to dock them for it.

A. 16 (2^4)
B. 20 (8 bits of physical page number + 12 bits of page offset)
C. LARGER (20 bits versus 16 bits)
D. 0xFFF
E. (vaddr >> 12) OR ((vaddr) & 0xF000 >> 12)
F. bit 4 is the "present or valid bit"
G. bit 7 is the "write" bit
H. bit 3 is the "user bit"

7. Teensy OS Page Tables (24 points)

The Teensy engineers are well on their way now, but they do have a few bugs and they need your help debugging the VM system. They hand you the following page table, using the notation we used for Assignment 6 for permissions, and need your help specifying correct behavior for the operations that follow.

Index	Physical Page Number	Permissions
0	0x00	PTE_U
1	0x01	PTE_P
2	0x02	PTE_P PTE_W

3	0x03	PTE_P PTE_U PTE_W
4	0xFF	PTE_U PTE_W
5	0xFE	PTE_U
6	0x80	PTE_W
7	0x92	PTE_P PTE_W PTE_U
8	0xAB	PTE_P PTE_W PTE_U
9	0x09	PTE_P PTE_U
10	0xFE	PTE_P PTE_U
11	0x00	PTE_W
12	0x11	PTE_U
Rest of PTEs follow and are all invalid		

For each problem below, write either the physical address of the given virtual address or identify what fault would be produced. The fault types should be one of:

- Invalid page access (there is no mapping for the requested page)
- Privilege violation (user level process trying to access a supervisor page)
- Permission violation (attempt to write a read-only page)

A. The kernel dereferences a NULL pointer
B. A user process dereferences a NULL pointer
C. The kernel writes to the address 0x8432
D. A user process writes to the address 0xB123
E. The kernel reads from the address 0x9876
F. A user process reads from the address 0x7654
G. A user process writes to the address 0xABCD
H. A user process writes to the address 0x2321

3 points each P (24 points)

A. Invalid page access
B. Invalid page access
C. 0xAB432
D. Invalid page access (we'll give credit for a privilege violation too, since the U bit isn't set)
E. 0x09876
F. 0x92654
G. Permission violation
H. Privilege vioation

8. Processes (20 points)

Consider the two programs shown below.

```
// Program 1
#include <stdio.h>
#include <unistd.h>
int
main(void)
{
    printf("PID %d running prog1\n", getpid());
}
```

```
// Program 2

#include <stdio.h>
#include <unistd.h>

int
main(void)
{
    char *argv[2];
    argv[0] = "prog1";
    argv[1] = NULL;
    printf("PID %d running prog2\n", getpid());
    int r = execv("./prog1", argv);
    printf("PID %d exiting from prog2\n", getpid());
}
```

A. How many different PIDs will print out if you run Program 2?
B. How many lines of output will you see?

Now, let's assume that we change Program 2 to the following:

```
#include <stdio.h>
#include <unistd.h>

int
main(void)
{
    char *argv[2];
    argv[0] = "prog1";
    argv[1] = NULL;

    printf("PID %d running prog2\n", getpid());
    pid_t p = fork();
```

```
    if (p == 0) {
        int r = execv("./prog1", argv);
    } else {
        printf("PID %d exiting from prog2\n", getpid());
    }
}
```

C. How many different PIDs will print out if you run Program 2?
D. How many lines of output will you see?

Finally, consider this version of Program 2.

```
#include <stdio.h>
#include <unistd.h>

int
main(void)
{
    char *argv[2];
    argv[0] = "prog1";
    argv[1] = NULL;

    printf("PID %d running prog2\n", getpid());
    pid_t p = fork();
    pid_t q = fork();
    if (p == 0 || q == 0) {
        int r = execv("./prog1", argv);
    } else {
        printf("PID %d exiting from prog2\n", getpid());
    }
}
```

E. How many different PIDs will print out if you run Program 2?
F. How many lines of output will you see?

3 points each for A-D (12 points)
4 points each for E-F (8 points)

A. 1
B. 2
C. 2
D. 3
E. 4
F. 5

9. Synchronization (20 points)

For each question below, select the synchronization primitive best matched to it. Include a sentence or two justifying your choice. Select from:

- Mutex
- Mutex and CV
- Binary semaphore
- Counting semaphore

A. You are implementing a server. Every incoming request is handled by a separate thread. Your server uses an old clunky and slow database and you know that if you have more than ten threads accessing the database at the same time, bad things happen. How can you limit concurrency in your database to no more than 10 threads?
B. Deep in your database code you have a buffer cache containing pages from the database files. Each page contains one or more variable sized records. When you want to add an item, remove an item, or change an item, you inevitably have to move data around on the page. What synchronization primitive would you use to protect each page?
C. You notice that the pthread library does not contain the equivalent of the wait system call, so you decide that you'll implement it on top of the library. What synchronization primitive will you use?
D. You are building a reader/writer lock. Reader/writer locks allow either a single writer to have access to a resource or multiple readers. On top of what primitives will you build the reader/writer lock? Explain roughly how you'll do it.

For questions E and F, we'll examine the reader/writer lock in a bit more detail.

E. Imagine a scenario where you have one or more readers holding the lock. A writer is blocked waiting for access, but before the last reader releases the lock, another reader comes along and gets a read lock (since you can grant a read lock in the presence of other readers). What term would we use to describe this situation (from the perspective of the writer)?
F. Another challenge for reader/writer locks is the upgrade operation. Let's say that a thread has something locked for reading, but upon reading it, decides that it wishes to upgrade its read lock to a writelock. Consider the scenario where two threads, each of whom have a read lock want to upgrade. What term would we use to describe this situation?

A-D: 4 points , graded like P5, but 2 points for the primtive and 2 points for demonstrating they understand what they are synchronizing.
E-F: 2 points

A. Counting sempahore (This maps quite nicely to simply letting threads access the DB)
B. Mutex (you want to ensure that only one thread can modify a page at any time)
C. I think we'll accept either of two answers: 1: Mutex and CV (you need to have a mutex to protect information about how many and/or which threads have exited and then you want the "parent" to wait on a CV until at least one thread exits).

   2: Binary semaphore per child used as a scheduling semaphore and then a second binary semaphore used when the parent wants to wait on any child.

D. Mutex and CV (basically you need a count of readers and writers and then if you are a writer, you wait until writers & readers is 0; if you're a reader, you wait until writers is 0).
E. Starvation
F. Deadlock

10. Be a CS61 TF! (18 points)

You are a CS61 teaching fellow. A student working on A4 is having difficulty getting pipes working. S/he comes to you for assistance. The function below is intended to traverse a linked list of commands, fork/exec the indicated processes, and hook up the pipes between commands correctly. The student has commented it reasonably, but is quite confused about how to finish writing the code. Can you help? Figure out what code to add at points A, B, and C.

```
#include "sh61.h"

typedef struct command command;
struct command {
    command *next; // Next in sequence of commands
    int argc;      // number of arguments
    int ispipe;    // pipe symbol follows this command
    char** argv;   // arguments, terminated by NULL
    pid_t pid;     // pid running this command
};

void
do_pipes(command *c)
{
    pid_t newpid;
    int havepipe = 0;   // We had a pipe on the previous command
    int lastpipe[2]= {-1, -1};
    int curpipe[2];

    do {
        if (c->ispipe)
            assert(pipe(curpipe) == 0);

        newpid = fork();
        switch (newpid) {
            case 0:
                if (havepipe) {
                    // There was a pipe on the last command; It's stored
                    // in lastpipe; I need to hook it up to this process???
                    // Part A
                }
                if (c->ispipe) {
                    // The current command is a pipe -- how do I hook it up???
                    // Part B
                }
```

```
                execvp(c->argv[0], c->argv);

                fprintf(stderr, "Exec failed\n");
                c->pid = -1;
                break;
            case -1:
                c->pid = newpid;
                break;
            default:
                // I bet there is some cleanup I have to do here!?
                Part C

                // Set up for the next command
                havepipe = c->ispipe;
                if (c->ispipe) {
                    lastpipe[0] = curpipe[0];
                    lastpipe[1] = curpipe[1];
                }
                c->pid = newpid;
                c = c->next;
                break;
        }
    } while (newpid != -1 && havepipe);
}
```

Each part has 3 lines, but it's going to take them some time to figure out the code. Let's count basically 2 points per line. That's a total of 18 points.

A)

```
close(lastpipe[1]);
dup2(lastpipe[0], STDIN_FILENO);
close(lastpipe[0]);
```

B)

```
close(curpipe[0]);
dup2(curpipe[1], STDOUT_FILENO);
close(curpipe[1]);
```

C)

```
if (havepipe) {
        close(lastpipe[0]);
        close(lastpipe[1]);
}
```

11. Making Network Servers Robust (16 points)

A. You've built a network server, list the resources that you might run out of if someone launched a DOS attack on you.
B. Someone is launching a DOS attack on you: how can you guarantee that you do not block, because a client sets up a connection, but never sends you any messages? (Be as specific as possible.)
C. Sam suggests that you just create a separate thread to handle each incoming connection. Why isn't this necessarily going to work?
D. A server sets up a socket to listen on a connection. When a client wants to establish a connection, how does the server manage the multiple clients? In your answer indicate what system call or calls are used and what they do.
E. Which of the following system calls might block:
    - accept
    - bind
    - connect
    - listen
    - setsockopt
    - select
    - socket

A: 3 points -- 1 for each resource
B: 3 points for a correct technique
C: 2 points for realizing that you will run out of stack space
D: 4 points: 2 points for accept; 2 points for a correct explanation
E: 4 points: 2 points for the first one correct + 1 for each additional (subtract one for each incorrect, but don't go negative!)

A. file descriptors, memory (stack), processes
B. You can set the file descriptors in non-blocking mode using fcntl or you can keep all the file descriptors in an FD_SET and use select, with a timeout, so that if nothing happens, you still unblock.
C. Each thread requires a stack and it's easy to run out of space for those stacks.
D. You call accept, which creates a new fd that is particular to the connection with a particular client. That is, the server uses a different fd for each client.
E. accept, connect, select