Final Exam

CSE 202: Computer Organization and Architecture (Spring 2017)

May 15th, 2017

Name: ________________________ Lehigh Email ID: ________________________

Instructions: Write down your name and Lehigh Email ID in the above spaces; you have 75 minutes; this is a closed-book, closed-notes exam; all calculators, PDAs, portable audio players and cell phones must be put away for the duration of the exam. ***This exam is single-sided, no work/answers on the backs of any pages will be accepted.***

Write down your Lehigh Email ID at the top of each following page, do not put your name on those pages. Do that now. Seriously, don't screw that up. Your pages may not get graded if you do not do this.

Exponent	$2^{Exponent}$	Register	Usage
0	1	rax	Return Value
1	2	rdi	Argument 0
2	4	rsi	Argument 1
3	8	rdx	Argument 2
4	16	rcx	Argument 3
5	32	r8	Argument 4
6	64	r9	Argument 5
7	128	rip	Instruction Pointer
8	256	rsp	Stack Pointer
16	65536	r10, r11	Caller-saved Temps
24	16777216	rbx, r12, r13, r14	Callee-saved Temps
31	2147483648		
32	4294967296		

For named registers, prefix r is for 8 bytes, e is for 4 bytes.
For numbered registers, suffix d is for 4 bytes, no suffix for 8 bytes (r8-r14)
Floating Point: Sign Bit (s), Significant (M), Exponent (E) such that $-1^s * 2^E * M$ is the expressed value. Exponent E = unsigned exponent bits - bias. The bias is $2^{k-1} - 1$.

Instruction	Meaning
mov	copies value into destination
lea	load effective address - does not dereference
call	goes to functions
cmp	subtracts and sets flags
jg	jump greater
ja	jump above (unsigned)
jmp	jump to an address
je	jump equal
jne	jump not equal
test	applies bitwise and to both arguments and sets flags

Lehigh Email ID: ________________________

1. (10 points: 4/3/2/1/1, max 10): Shell Lab:

```
1  void sigchld_handler(int sig)
2  {
3      pid_t pid;
4      int status;
5
6      if((pid = waitpid(-1, &status, 0)) > 0)
7      {
8          if(WIFSTOPPED(status))
9          {
10             printf("Job [%d] (%d) stopped by signal %d\n",
11                 pid2jid(pid), pid, WSTOPSIG(status));
12             getjobpid(jobs, pid)->state = ST;
13             sigtstp_handler(SIGTSTP);
14         }
15         else if(WIFSIGNALED(status))
16         {
17             printf("Job [%d] (%d) terminated by signal %d\n",
18                 pid2jid(pid), pid, WTERMSIG(status));
19             sigint_handler(SIGINT);
20         }
21         else
22         {
23             deletejob(jobs, pid);
24         }
25     }
26     if((errno != ECHILD) && (errno != 0))
27     {
28         unix_error("Waitpid error");
29     }
30 }
```

Consider the code above. There are at least five (5) distinct types of errors in this code. Indicate, by line number(s), what the five errors are and fix them (code fixes are best but an explanation is acceptable). Note that an error type can have multiple lines. Also, the same line can appear in more than one error type.

(a) Error 1 (4 points):

(b) Error 2 (3 points):

(c) Error 3 (2 points):

(d) Error 4 (1 point):

(e) Error 5 (1 point):

2. (10 points): Page Faults: When a page fault occurs, the kernel handles exception control flow in order to resolve the situation. What formally happens? Make sure to start from the task structures maintained by the kernel and consider how each possible outcome can be determined from the kernel. Do not forget to include how protection bits come into play here.

   Specifically, assume you are the kernel and were just told that process X tried to access address Y and it was not valid in the page table.

Lehigh Email ID: ________________________

3. (10 points: 4/2/4): Malloc: Consider a situation where realloc is called on *ptr* in order to obtain a larger block. Assume the block that is physically after *ptr* is free but not of sufficient size for the reallocation.

   If the block that is physically before *ptr* is sufficiently large (both on its own or in combination with the block physically after *ptr*), make an distinct argument for each of the following:

   (a) Reallocate into the block physically before *ptr* starting at the beginning of the block.

   (b) Reallocate into the block physically before *ptr* starting as far into the block as possible (leaving the beginning of that block free) and using the block physically afterwards, if applicable.

   (c) Just copy the block into a new, larger, location.

Lehigh Email ID: ________________________

4. (10 points) Malloc: Consider if we were to implement the free list(s) as Binary Search Tree(s) (BST). Describe all of the implementation and performance differences between a BST model and an explicit free list.

Lehigh Email ID: ________________________

5. (10 points): Process Creation: The system call *fork*, if implemented as described, is a rather expensive behavior in terms of number of cycles. In particular, *fork* is generally called with the intention of using *exec* afterwards. Explain how *demand paging* and *private copy-on-write (COW)* come together to allow us an efficient model for *fork/exec* semantics.

Lehigh Email ID: ________________________

6. (10 points: 7/3): Consider the code below. Assume that neither the compiler nor the CPU will reorder any of the code.

```c
static jmp_buf env;
static int value = 4;

void foo(int *val)
{
    value += (*val);
    (*val)++;
    printf("Value is now: %d\n", value);
    longjmp(env, *val);
}

int main(int argc, char** argv)
{
    int local = 24;
    int jVal = (int)((setjmp(env) + local) / 3);
    if(jVal < 10)
    {
        if(jVal & 0x1)
        {
            foo(&local);
        }
        else
        {
            foo(&value);
        }
    }
    printf("Done, final values are: %d and %d\n", jVal, local);
    return 0;
}
```

(a) What is the output of this program?

(b) Assume *local* is instead set to 1 when initialized. What is the output of this program?

Lehigh Email ID: ______________________

7. (10 points): I/O: Consider the code below:

```
1  int main(int argc, char **argv)
2  {
3      int fd1, fd2, fd3, fd4;
4
5      fd1 = open(argv[1], O_CREAT|O_TRUNC|O_RDWR, S_IRUSR|S_IWUSR);
6      write(fd1, "apple", 5);
7      fd2 = open(argv[1], O_APPEND|O_WRONLY, 0);
8      write(fd2, "banana", 6);
9
10     fd3 = dup(fd1);
11     fd4 = dup(fd2);
12     write(fd4, "carrot", 6);
13
14     char buffer[10];
15     read(fd3, &buffer[0], 2);
16
17     write(fd3, "donut", 5);
18     write(fd4, "eclair", 6);
19
20     read(fd1, &buffer[0], 4);
21     write(fd3, "fig", 3);
22     return 0;
23 }
```

Assuming the code is executed with valid command line arguments answer the following. Assume that O APPEND will always append to the end (as if it calls lseek(file, 0, SEEK END) before each and every write) of the file. O CREAT—O TRUNC creates the file as a size 0 file.

(a) What are the contents of the resulting file?

(b) What is read by each read call in the code above? Specify each read by its line number.

8. (10 points) Caches: Consider a 4-way set-associative cache that is initially empty with block sizes of 8 bytes, addresses that are 64 bits long and a total data capacity of 32KB. The following code performs very slowly and our boss is mad at us and doesn't trust us with a compiler any longer. Our only option to improve the performance is to redesign the cache itself. We have three options, we can a) double the block size, b) double the associativity (number of ways/columns) or c) double the number of lines (rows/sets). In each case, we are doubling the total data capacity of our cache to 64KB.

   Which option will result in the best performance and why?

```
// assume array[0][0] is at address 0.
long long array[1 << 30][4];

for(size_t i=0; i<(1 << 30); i++)
{
    for(size_t j=0; j<4; j++)
    {
        if((i+1) < (1 << 30))
        {
            array[i][j] = array[i+1][j] + 1;
        }
    }
}
```

Lehigh Email ID: ______________________

9. (20 points): Virtual Memory

   On some machine we are using, Virtual Addresses are 12 bits, Physical Addresses are 10 bits and the Page Size is 64 bytes. Given the following Page Table (incomplete but contains all that is relevant) and 2-way set-associative TLB, fill in the table specified below. Note that VPN is Virtual Page Number, PPN is Physical Page Number and V is valid. Assume all numbers are in hex.

Page table				TLB					
VPN	PPN	V		Way A			Way B		
00	05	1	Row	Tag	Valid	PPN	Tag	Valid	PPN
01	01	0	0:	03	0	0A	02	1	0B
02	07	0	1:	03	0	0C	02	0	0D
03	0F	1	2:	04	1	03	00	1	07
04	0E	1	3:	03	1	0C	02	1	0D
05	04	1	4:	01	0	06	03	1	06
06	0F	0	5:	03	1	0A	02	0	0B
07	07	1	6:	05	1	09	03	1	07
08	06	0	7:	01	1	06	03	0	06
09	03	0							
0A	0C	0							
0B	04	1		0x4FB	0x61C	0x333	0x52C	0x002	
0C	0C	1	VPN						
0D	09	0	PPN						
0E	06	0	TLB Tag						
0F	0C	1	TLB Index						
10	0B	0	TLB Hit						
11	01	1	Page Fault?						
12	02	0							
13	0D	1							
14	03	1							
15	0A	1							
16	0B	1							
17	09	1							
18	0F	1							

Did you remember to put your Lehigh Email ID at the top of each page? Did you?