CSCI 356 Fall 2017 : Practice Final Exam Solutions

DO NOT OPEN EXAM PACKET UNTIL INSTRUCTED TO DO SO

YOU MAY FILL IN INFORMATION ON THE FRONT NOW

PLEASE TURN OFF ALL ELECTRONIC DEVICES

ID#:	
Name:	

- This exam is closed book. You are allowed one (2) 8.5" x 11" **handwritten** note sheets
- You will have one hundred and ten (110) minutes to complete this exam.
- Answer the questions only in the spaces provided on the question sheets.
- If you give multiple solutions to a problem without indicating which one you want graded, the grader may select one to grade.
- Your answers do not need to be complete, grammatically correct sentences.
- This practice exam *is not* a substitute for reading the textbook, doing practice problems, reviewing the course and assignments, or discussing material with your classmates.
- This might not be exhaustive coverage either.
- Instead, this exam is a chance to practice some material you might not have seen in an exam-like context yet.

Problem	1	2	3	4	5	6	7
Possible							
Earned							

1. What are the possible output sequences from the following program:

```
int main() {
      if (fork() == 0) {
            printf("a");
            exit(0);
      }
      else {
            printf("b");
            waitpid(-1, NULL, 0);
      }
      printf("c");
      exit(0);
}
```

Circle the possible output sequences: abc acb bac bca cab cba

2. I am going to ask you what the output of the following program is.

```
pid_t pid;
int counter = 5;

void handler1(int sig) {
      counter = counter - 2;
      printf("%d", counter);
      fflush(stdout);
      exit(0);
}

int main() {
      signal(SIGUSR1, handler1);
      printf("%d", counter);
      fflush(stdout);
      if ((pid = fork()) == 0) {
            while(1) {};
      }
      kill(pid, SIGUSR1);
      waitpid(-1, NULL, 0);
      counter = counter + 1;
      printf("%d", counter);
      exit(0);
}
```

What is the output? 536

3. Suppose a system has the following parameters:
   - Virtual addresses are 20 bits wide
   - Physical addresses are 18 bits wide
   - Page size is 1KB (= 1024 bytes)
   - The TLB is 2-way set associative and has 16 total entries.

   a. Show a diagram of the breakdown of a virtual address. Indicate which bit(s) are used for the virtual page number, the virtual page offset, the TLB index, and the TLB tag.

19 10 9 0

<table>
<tr><td colspan="2">VPN</td><td colspan="10">VPO</td></tr>
<tr><td>TLB Tag</td><td>TLB Index</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
</table>

19 13 12 10 9 0

   b. Show a diagram of the breakdown of a physical address. Indicate which bit(s) are used for the physical page number and physical page offset.

17 10 9 0

PPN	PPO

   c. Suppose we are going to translate virtual address `0x078E6` to physical memory. The current state of the system is on the next page.
      i. What is the virtual page number?
      0000 0111 10 (01E)
      ii. What is the TLB index?
      110 (6)
      iii. What is the TLB tag?
      000 0011 (03)
      iv. Will this lookup produce a TLB hit (yes or no)?
      no (no tag match)
      v. Will this lookup produce a page fault (yes or no)?
      No (valid bit is 1)

      vi. What is the physical address that corresponds to `0x078E6`?
      0101 0111 00 1110 0110 (0x15CE6)

What follows is the state of the TLB and Page Table for use in problem 4.

TLB:

Index	Tag	Physical page #	Valid
0	03	C3	1
	01	71	0
1	00	28	1
	01	35	1
2	02	68	1
	3A	F1	0
3	03	12	1
	02	30	1
4	7F	05	0
	01	A1	0
5	00	53	1
	03	4E	1
6	1B	34	0
	00	1F	1
7	03	38	1
	32	09	0

Page Table:

VPN	PPN	Valid	VPN	PPN	Valid
000	71	1	010	60	0
001	28	1	011	57	0
002	93	1	012	68	1
003	AB	0	013	30	1
004	D6	0	014	0D	0
005	53	1	015	2B	0
006	1F	1	016	9F	0
007	80	1	017	62	0
008	02	0	018	C3	1
009	35	1	019	04	0
00A	41	0	01A	F1	1
00B	86	1	01B	12	1
00C	A1	1	01C	30	0
00D	D5	1	01D	4E	1
00E	8E	0	01E	57	1
00F	D4	0	01F	38	1

4. Three of the following four statements are benefits of virtual memory. For each one *that is a benefit*, briefly explain how virtual memory allows this benefit. For the one (and it is only one) that is not, mark it as "not a benefit."

- It allows the virtual address space to be larger than the physical address space
  Virtual memory allows the mapping from a virtual address to a physical address. If there aren't enough physical addresses available, then the kernel can overwrite the existing mapping and store the old data at that physical location to hard disk.

- No process can accidentally access the memory of another process
  Virtual addresses are mapped to physical addresses by the kernel, so a process can't access physical memory that it doesn't own

- The TLB is more effective since without it dereferencing a virtual address now requires two or more memory accesses
  Not a benefit - property of the TLB rather than virtual memory itself

- Different processes can have overlapping virtual address spaces without conflict
  Every process has the illusion of using the same address space, but they don't actually overlap in physical memory due to memory mapping behind the scenes

5. Suppose an int A is stored at virtual address 0xff987cf0, while another int B is stored at virtual address 0xff987d98. I assert that if the size of a page is 0x1000 bytes, then A's physical address is numerically less than B's physical address.

A. Is the assertion always, sometimes true, or never true?
   always

B. Why?
   Page size is 0x1000 bytes = 16^3 bytes = 2^{12} bytes, so the upper 32 - 12 = 20 bits form the VPN. A and B are in the same virtual page (VPN = 0xff987), which means they must be in the same physical page.

6. Consider a 32-bit system with a page size of 4KB. A certain kernel designer wishes to analyze the merits of using 2-level page tables.

a. How many entries are there in the page directory?
4KB/page / 4B/entry = 1024 entries/page

b. How much virtual memory is reachable from a single page directory entry? (i.e.: 4KB are reachable from a single page table entry).
4KB/page * 1024 pages/directory = 4MB/directory

7. Consider the following dump of assembler code for function foo:

```
0x0000000000400632 <+0>:      sub    $0x1,%esi
  0x0000000000400635 <+3>:          mov    $0x0,%r9d
  0x000000000040063b <+9>:          jmp    0x400666 <foo+52>
  0x000000000040063d <+11>:     lea    (%r9,%rsi,1),%eax
  0x0000000000400641 <+15>:     mov    %eax,%ecx
  0x0000000000400643 <+17>:     shr    $0x1f,%ecx
  0x0000000000400646 <+20>:     add    %eax,%ecx
  0x0000000000400648 <+22>:     sar    %ecx
  0x000000000040064a <+24>:     mov    %ecx,%eax
  0x000000000040064c <+26>:     movslq %ecx,%r8
  0x000000000040064f <+29>:     mov    (%rdi,%r8,4),%r8d
  0x0000000000400653 <+33>:     cmp    %edx,%r8d
  0x0000000000400656 <+36>:     je     0x400670 <foo+62>
  0x0000000000400658 <+38>:     cmp    %edx,%r8d
  0x000000000040065b <+41>:     jle    0x400662 <foo+48>
  0x000000000040065d <+43>:     lea    -0x1(%rcx),%esi
  0x0000000000400660 <+46>:     jmp    0x400666 <foo+52>
  0x0000000000400662 <+48>:     lea    0x1(%rcx),%r9d
  0x0000000000400666 <+52>:     cmp    %esi,%r9d
  0x0000000000400669 <+55>:     jle    0x40063d <foo+11>
  0x000000000040066b <+57>:     mov    $0xffffffff,%eax
  0x0000000000400670 <+62>:     repz retq
```

Write the C code for function foo. The signature is `int foo (int * a, int b, int c)`

```
// compact translation
int foo (int *a, int b, int c) {
      int high = b - 1, low = 0;
      while (low <= high) {
            int mid = (low + high) / 2;
            if (a[mid] = c) return mid;
            else if (a[mid] < c) low = mid + 1;
            else high = mid - 1;
      }
      return -1;
}
```

(See next page for a more line-by-line translation)

```c
// direct translation
int foo (int *a, int b, int c) {
	b--; // int high
	int r9d = 0; // int low
	int eax, ecx, r8d;
	long r8;
	while (r9d - b <= 0) {
		eax = r9d + b; // low + high
		ecx = eax;
		ecx >>= 31;
		ecx += eax;
		ecx >>= 1; // (low + high) / 2
		eax = ecx; // eax = (r9d + b) / 2 (no overflow)
		r8 = ecx;
		r8d = a[r8]; // r8d = a[eax]
		if (r8d == c) {
			return eax;
		}
		else if (r8d - c <= 0) {
			r9d = ecx + 1; // r9d = eax + 1
		}
		else {
			b = ecx - 1; // b = eax - 1
		}
	}
	return -1;
}
```