Name: _____

Write your name and computing ID above. Write your computing ID at the top of each page in case pages get separated. Sign the honor pledge below.

Generally, we will not answer questions about the exam during the exam time. If you think a question is unclear and requires additional information to answer, please explain how in your answer. For multiple choice questions, write a * next to the relevant option(s) along with your explanation.

On my honor as a student I have neither given nor received aid on this exam.

_____

1. Suppose a system uses:
   - 65536 byte (2^{16}-byte) pages
   - 24-bit virtual addresses
   - 40-bit physical addresses

(a) (4 points) If this system used one-level page tables with 8-byte entries, then how much space, in bytes, would these page tables take up? You may leave your answer as an unsimplified arithmetic expression.

$2^{24-16} \times 8 = 2048$ bytes

(b) (8 points) A process running on the system has the following memory layout:

- `0x0`–`0x100000`: unused (inaccessible)
- `0x100000`–`0x200000`: code (executable; loaded on demand from executable file)
- `0x200000`–`0x300000`: globals (read/write; allocated on demand)
- `0x300000`–`0x400000`: heap (read/write; allocated on demand)
- `0x400000`–`0x700000`: unused (inaccessible)
- `0x700000`–`0x800000`: stack (read/write;allocated on demand)
- `0x800000`–`0xFFFFFF`: unused (inaccessible)

Assume the system does loading and allocating on demand by allocating only one page per page fault and initially having no pages mapped in the page table. (A page fault is the exception that occurs when a program accesses memory that is not mapped in its page table.)

When the program starts its stack pointer is set is set to 0x7FFF00.

The first things the program does are (in order):

- run an instruction at addresses 0x11FFF0 through 0x11FFF7 inclusive that pops an 8-byte value from the stack
- run an instruction at addresses 0x11FFF8 through 0x11FFFF that pops a second 8-byte value from the stack
- run an instruction at addresses 0x120000 through 0x120007 that jump to address 0x132000
- run an instruction at addresses 0x132000 through 0x132003 that pushes an 8-byte value to the stack

List the page faults that will occur during these operations. For each page fault, indicate the virtual address accessed to trigger it.

Solution: 0x11FFF0 (code) , 0x7FFF00 (stack, first pop), 0x120000 (code), 0x132000 (code)

2. Consider the following C program:

```
1  int x = 0;
2  int main() {
3      int y = 0;
4      printf("begin\n"); fflush(stdout);
5      pid_t p[2];
6      for (int i = 0; i < 2; i += 1) {
7          p[i] = fork();
8          if (p[i] == 0) {
9              x += 1;
10             y += 1;
11             printf("loop x:%d y:%d\n", x, y); fflush(stdout);
12             exit(0);
13         } else {
14             x += 10;
15             y += 10;
16         }
17     }
18     for (int i = 0; i < 2; i += 1)
19         waitpid(p[i], NULL, 0);
20     printf("end x:%d y:%d\n", x, y); fflush(stdout);
21 }
```

For the following questions, assume fork and waitpid do not fail.

(a) (8 points) Write two plausible outputs for the above program.

Output 1:	Output 2:

```
begin
loop x:1 y:1
loop x:11 y:11
end x:20 y:20
```

```
begin
loop x:11 y:11
loop x:1 y:1
end x:20 y:20
```

(b) (4 points) Suppose that when `i` is 0, the first access to `x` that runs after the call to `fork()` is the `x += 10`. Which, if any, of the following will be true about this assignment? **Select all that apply.**

- ■ most likely, an exception will occur while it is running *copy-on-write*
- ☐ running this assignment will result in space being allocated on the heap
- ☐ the physical address written by this assignment is the same address that will be written by the `x += 1` assignment
- ■ this assignment will run in the same process that ran `printf("begin\n")`

3. (6 points) Consider the following cache contents of a 2-way set associative cache with 2 sets. (The index and tag are written in binary. Data bytes are written as hexadecimal byte values, with the leftmost value obtained from the lowest address.)

index	valid	tag	data bytes	valid	tag	data bytes	LRU
0	1	1100	11 22 33 44	1	1111	55 66 77 88	1
1	1	1100	22 44 55 66	1	1111	88 99 AA FF	1

Suppose a program reads a byte with the value `0xCC` (hexadecimal) from address `0x9` using this cache. Indicate above how the cache is modified as a result, writing out the new values for fields in the space below them. If you need the values for bytes of memory which are not given, write a `*` in place of those unsupplied values.

0x9 = (tag 0001)(index 0)(offset 01)

index	valid	tag	data bytes	valid	tag	data bytes	LRU
0	1	1100	11 22 33 44	1	~~1111~~ 0001	~~55 66 77 88~~ * CC * *	~~1~~ 0
1	1	1100	22 44 55 66	1	1111	88 99 AA FF	1

4. Consider an SSH client that displays messages from a remote server.

   First, the server sends the text "Please wait...", and the SSH client displays that. Then, 10 seconds later, the server sends the text "done" and a newline, and the SSH client displays that.

(a) (4 points) On the system where the SSH client is running, while the SSH client is waiting 10 seconds, most likely _____. **Select all that apply.**

- ☐ the SSH client's stack pointer is stored in a register on the processor
- ■ the processor's page table base pointer is set to a valid value
- ■ values on the SSH client's stack are stored in physical memory
- ☐ running an instruction that reads from the virtual address stored in the SSH client's stack pointer will successfully read a value from the SSH client's stack

(b) (7 points) On the system where the SSH client is running, which of the following is likely to occur as part of the process above? **Select all that apply.**

- ☐ a new thread is created
- ■ the operating system starts running from an exception triggered by network hardware, interrupting a non-SSH program
- ☐ the operating system starts running from an exception triggered by network hardware, interrupting the SSH program
- ☐ the operating system starts running from an exception triggered by display hardware, interrupting the SSH program

- [x] the SSH client makes a system call to request input from the network
- [x] the SSH client makes a system call to display output to the screen
- [ ] the SSH client experiences a page fault *dropped*

5. Consider the following Makefile:

```
all: quux libfuncs.so

quux: quux.o foo.o
        clang -o $@ $^

libfuncs.so: foo.o bar.o
        clang -shared foo.o bar.o -o libfuncs.so

bar.o: bar.c constants.h library.h bar.h
        clang -fPIC -c $< -o $@

foo.o: foo.c library.h
        clang -fPIC -c $< -o $@

quux.o: quux.c constants.h
        clang -c $< -o $@

constants.h: make_constants.py constants.dat
        python3 make_constants.py
```

(a) (6 points) Assume initially all files are up to date. Then, modifying `constants.dat` and then running `make` will cause the commands for which rules to be run? Write the targets of the rules *in an order in which they might be run.*

> **Solution:** (first) constants.h, (after constants.h) bar.o, (after bar.o) libfuncs.so, (after constants.h) quux.o, (after quux.o) quux, (no deduction for including 'all')

(b) (4 points) Which of the following statements would be consistent with the makefile above (even if the makefile does not *require* them to be true)? **Select all that apply.**

- ☐ `foo.c` calls functions in `bar.c` *not possible if quux links*
- ■ `make_constants.py` reads `constants.dat` when it is run
- ☐ `quux` loads `libfuncs.so` when it is run
- ■ `quux.c` includes `constants.h`

6. Suppose a system uses a 32768-byte (2^{15} byte) two-way set associative data cache with 32-byte blocks, a write-back policy, a write-allocate policy, and an LRU policy.

   On this system, a programmer is developing a program maintains a catalog of 1 million books in memory.

   Each book has an associate ID number, category ID number, and title.

   The programmer needs to choose between representing the catalog like:

```
struct Book {
    long id;
    long category_id;
    char title[112];
};

struct Book books[1000000];
```

   Or like:

```
long book_ids[1000000];
long book_category_ids[1000000];
char book_titles[1000000][112];
```

   In the first scheme, `book[4].category_id` would be the category of the book with index 4 in the catalog (and similar for book IDs and titles). In the second `book_category_ids[4]` would access that ID.

   For the questions below, assume that

   - longs are 8 bytes in size and chars are 1 byte in size.
   - structs and arrays laid in memory with no padding (and in the order in which they are declared)
   - each global variable starts at the beginning of a cache block
   - only accesses to the book catalog use the data cache

   You may leave each of your answers below as an unsimplified arithmetic expression.

(a) Suppose the program sequentially accessed all category_ids to find books matching a particular category ID. If the system's data cache is initially empty, how many data cache misses would likely occur…

i. (3 points) …in the first scheme (array of structs)?

1000000

ii. (3 points) …in the second scheme (multiple arrays)?

1000000 / 4

(b) Suppose the program updated the IDs, category IDs, and titles of the books with indices 0 through 9, inclusive. Assume that the new book titles take up all 112 available characters.

If the system's data cache is initially empty, how many data cache misses would likely occur…

i. (3 points) …in the first scheme (array of structs)?

10 * 4 = 40

ii. (4 points) …in the second scheme (multiple arrays)?

ceil(10 / 4) + ceil(10 / 4) + ceil(112 * 10/ 32) = 3 + 3 + 35 = 41