Final Exam

601.229 Computer Systems Fundamentals

Spring 2020

Johns Hopkins University

Instructors: Xin Jin and David Hovemeyer

14 May 2020

Complete all questions.

Use additional paper if needed.

Take-home exam

Q1. Caches *20 points*

(a) [4 points] An architecture has 32 bit addresses. Assume the following cache parameters:

- 262,144 (2^{18}) total bytes of data
- Each block contains 256 (2^8) bytes
- 4-way set associativity

Draw a diagram showing the structure of an address, indicating the exact ranges (bit positions) of the offset bits, the index bits, and the tag bits.

(b) [4 points] How many address bits are used to represent the index when a cache is fully associative? Explain briefly.

(c) [12 points] Complete the following table. For each address in the *Request* column, indicate the tags of cached blocks after handling the request. Addresses are 8 bits, blocks are 8 bytes, there are 4 sets, and the cache is 2-way set associative. All slots are initially empty. Use FIFO (First-In, First-Out) when replacing blocks.

	Set 0		Set 1		Set 2		Set 3	
Request	Slot 0	Slot 1	Slot 0	Slot 1	Slot 0	Slot 1	Slot 0	Slot 1
	empty	empty	empty	empty	empty	empty	empty	empty
01101111								
10010111								
10100111								
00010111								
10101100								
11001010								
00101100								
10100101								
10000101								
01011011								

Q2. Processes and linking *20 points*

(a) [10 points] Consider the following program:

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

void *addr1, *addr2;

void f(void) {
  int b;
  addr2 = &b;
}

int main(void) {
  int a;
  addr1 = &a;
  f();
  assert(addr1 != addr2);
  if (addr1 < addr2) {
    printf("addr1 < addr2\n");
  } else {
    printf("addr2 > addr1\n");
  }
  return 0;
}
```

Assuming that this program is compiled for x86-64, what output will the program produce? Explain.

(b) [10 points] When generating a non-position-independent executable, the linker assigns an absolute address to each definition (function or global variable) in the executable. In general, many processes can be executed using the same executable. For example, let's say that

- `/usr/bin/vi` is the executable for the vi editor
- The executable is not position-independent
- Many users are running the executable at the same time

Explain how is it possible that many instances of the executable can be running simultaneously, even though they are all using the same addresses.

Q3. Processes and virtual memory *20 points*

(a) [5 points] Name two sections of a Linux executable such that it would be safe for the operating system kernel to map the memory pages containing data loaded from those sections into the address spaces of multiple processes. Explain why it is safe to share the memory for those sections. If any hardware mechanisms are necessary to ensure that the sharing can be done safely, indicate what they are.

(b) [5 points] Name a section of a Linux executable where it would *not* be safe for the operating system kernel to map the memory pages containing data loaded from that section into the address spaces of multiple processes. Explain why sharing pages for this section would not be safe.

(c) [5 points] The x86-64 architecture has (effectively) 48 bit virtual addresses. How much memory would be required to represent the mappings of virtual to physical pages for the entire 48-bit virtual address space if the page tables were a single "flat" array? Assume that each page table entry requires 8 bytes.

(d) [5 points] On systems with virtual memory, it is possible for the contents of a data cache to be indexed by either virtual or physical addresses. State one advantage and one disadvantage for both indexing a cache by virtual address and indexing a cache by physical address.

Q4. Threads *20 points*

(a) [5 points] Briefly explain why each thread in a multithreaded program needs its own call stack.

(b) [5 points] Is it possible that it would be useful for a program to create multiple threads even if it is running on a single-core CPU? Explain briefly.

(c) [10 points] Consider the following `Queue` data type and `queue_try_enqueue` function:

```
struct Queue {
  void *items[MAX];
  int head, tail, count;
  pthread_mutex_t lock;
};

// Try to add item to queue, returns true if successful
// and false if the queue is currently full.
bool queue_try_enqueue(struct Queue *q, void *item) {
  pthread_mutex_lock(&q->lock);

  if (q->count == MAX) { return false; }

  q->items[q->tail] = item;
  q->tail = (q->tail + 1) % MAX;

  pthread_mutex_unlock(&q->lock);

  return true;
}
```

Briefly explain the problem with the `queue_try_enqueue` function, and how to fix it.

Q5. Networks *20 points*

(a) [5 points] Briefly explain why server programs generally need to use some form of concurrency, such as processes or threads.

[continued on next page]

(b) [15 points] Complete the following function. It should read one line of text from the file descriptor given as the parameter, and then send back (using the same file descriptor) a line of text of the form

`Hello,` *text*

where *text* is the contents of the line of text read from the file descriptor. Make sure that the function will work correctly if the file descriptor refers to a network socket.

The function should return 1 if successful, and 0 if an error occurs.

```
int hello_transaction(int fd) {
```