Final Exam Solutions

Question 1

Each question had only one correct answer, and was worth 1 mark. Any answer that marked multiple options as correct were marked as incorrect.

1. Which of the following is true about TLB misses on an x86-64 system?
   C - *TLB miss sometimes results in a page fault.*

2. Which of the following is true about races?
   A - *A race occurs when correctness of the program depends on one thread reaching point a before another thread reaches point b.*
   (B is is incorrect because mutual exclusion doesn't exclude possibility of race conditions e.g. one thread might need to fill a buffer before another thread reads it, which is an issue even if neither thread is reading the buffer at the same time)

3. Each thread has its own:
   C - *Stack*

4. Which of the following is the correct ordering (left-to-right) of a file's compilation cycle (a filename with no extension is an executable):
   D - *foo.c → foo.s → foo.o → foo*

5. How many sets does a 4-way set-associcitive cache with 256 byte capacity and 32 byte block size has?
   B - 2

6. Which of the following sentences about reader-writer locks is not true?
   C - *Many readers and exactly one writer can hold the same rwlock at the same time.*

7. When can short counts occur?
   A - *When an EOF is encountered during a read.*

8. Which of the following is true about the page cache?
   D - *Page cache amortizes the cost of accessing disks.*

9. We use dynamic memory because:
   D - *Heap allows allocating data that has a lifetime not related to the execution of the current routine.*

10. Consider the following segment of network code. Which of the following is true about read()?
    C - *the read() call is not guaranteed to return quickly.*

11. Which of the following is true about signals?
    C - *A signal handler is a separate logical flow that runs concurrently with the main program.*
    (D is wrong because it suggests signals of the same TYPE as the current one are allowed, but that is not true)

12. Which of the following is not true about physical pages?
    B - *Physical addresses have a one-to-one mapping to virtual addresses.*
    (not all virtual addresses are mapped to physical addresses at a given time)

13. Multi-level page tables are used because:
    A - *They are space-efficient.*

14. What is the maximum amount of physical memory a server with 44 bit physical addresses can have?
    D - *16TB*

15. What is the approximate capacity of a hard disk with 512 byte sectors, 100 sectors per track, 10,000 tracks per surface, and one platter?
    B - *1 GB*
    (A is incorrect because there are 2 surfaces per platter)

16. I/O multiplexing cannot fully utilize multicore processors.
    True

17. Processes are more space-efficient for the kernel to manage than threads.
    False

18. HTTP is a text-based kernel-level protocol
    False

19. TCP and UDP are both connection-oriented protocols.
    False (UDP is not connection oriented; you don’t need to establish a connection before sending things)

20. Garbage collection can have an impact on memory locality
    True

Question 2

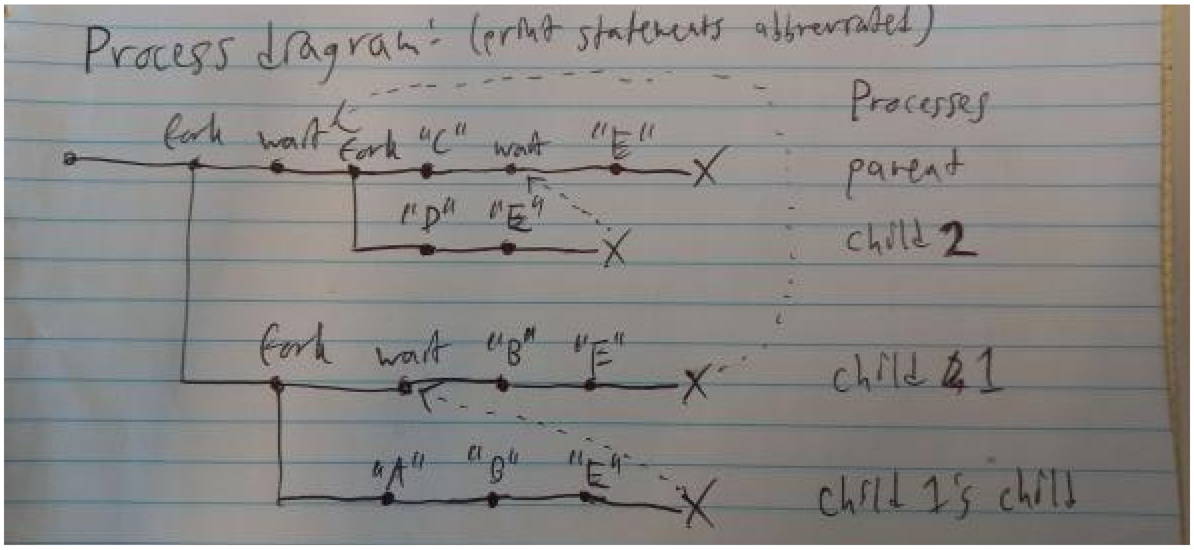
M = 9, N = 12

- First note initially rdi = i, rsi = j (actually edi/esi, but we just extend these from the 32 bit subregisters to the full 64 bit words using movslq anyway)
- We set rax = i, then do:
  - rax = rax << 3 = i << 3 = 8i
  - rax = rax + rdi = 8i + i = 9i
  - rax = rax + rsi = 9i + j
- We also do:
  - rsi = rsi + rsi*2 = j + j*2 = 3j
  - rsi = rdi + rsi*4 = i + 4*(3j) = 12j + i
- Then for access to array 1 (array1[j][i] in the C code), we read the memory address array1 + 4*rsi = array1 + 4*(12j + i). Note that the multiplication by 4 is to account for elements of array1 being ints and 4 bytes long. The fact that the row index i is multiplied by 12 indicates that rows of array1 are 12 elements long, which corresponds to N. Hence, N = 12.
- Finally, for access to array 2 (array2[i][j] in the C code), we write to the memory address array2 + 4*rax = array2 + 4*(9i + j). We similarly find array2's rows are 9 elements long, so M = 9.

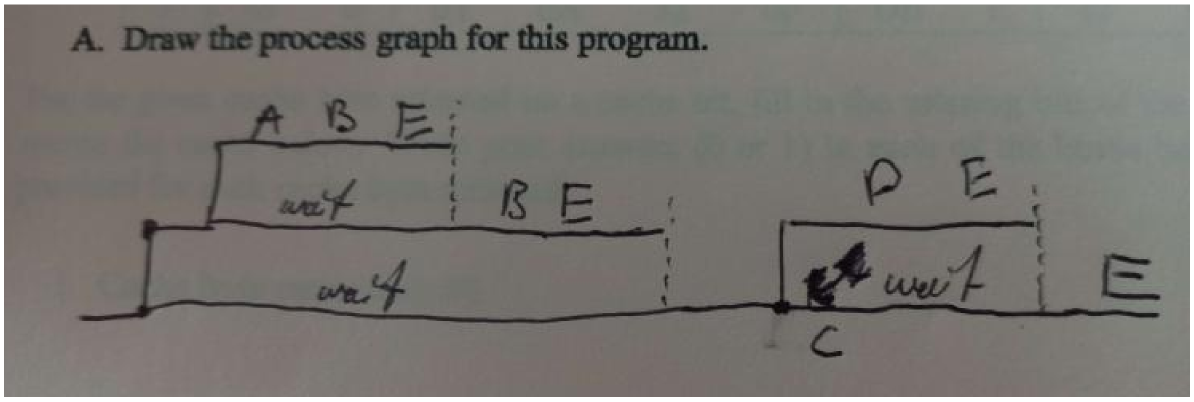
Some marks were given for accuracy of the working out if the final answer was incorrect. Answers that wrote M and N the wrong way around received 9/10.

Question 3

Process diagram: (note: printf("X") was just abbreviated to "X", and dashed lines indicate which process each wait() syscall is waiting for).



This kind of diagram is also correct (they've placed the second child further to the right and made the diagram indicate what happens over time in a nice way).


A. Draw the process graph for this program.

Possible outputs:
ABEBECDEE
ABEBEDCEE
ABEBEDECE

- 4 marks awarded for correct process diagram. Using dotted lines to indicate wait calls was not required to receive full marks.
- Listing all correct outputs was worth 6 marks. Each incorrect output listed was given -1 marks (total mark had a minimum of 0)

Question 4

First, consider how addresses are divided for cache lookup:

- 4 bytes per line, so 2 bits for block offset
- 8 sets in cache, so 3 bits for set index

- Remaining 8 bits of 13-bit physical addresses consist of tag

For cache byte returned of 06:

- Hints indicate that the first bit of tag is 0, and that the last bit of block offset is 1, so block offset is either 1 or 3. There are several instances of 06 in the cache, but the hints mean that it must be the 06 in set 1, second block, at block offset 1.
- Hence, tag is 0x38, set index is 0b001 and block offset is 0b01
- **Answer: 0b 0011 1000 001 01**

For cache byte returned of 32:

- Hints indicate that the first bit of tag is 1, and that the last bit of block offset is 0 (block offset is either 0 or 2). This time the only possibility is the 32 in set 6, second block, with block offset 0
- Hence, tag is 0xF0, set index is 0b110 and block offset is 0b00
- **Answer: 0b 1111 0000 110 00**

Marks:

- Each address was worth 5 marks. Providing working out was not required to receive full marks.

Question 5

Since we iterate through columns then rows, the arrays are accessed in memory order. Since the cache is write-allocate, the dest array's entries are also cached. And since line size is 32 bytes = 8 ints, we store 8 consecutive entries of the arrays at a time.

For Part 1 and 3, the src array is of size 64 and 384 respectively, causing the dst array to be mapped at a different location into the cache so the two don't conflict. So we get a miss rate of 12.5%.

For Part 2, the src array is of size 256, causing the dst array's entries to map to the same location as the src array's. So when accessing the dst array it evicts the data cached just before for the src array and vice versa, leading to only cache misses. So, cache miss is 100%.

So in summary:

- Part 1 (3 marks): 12.5%
- Part 2 (3 marks): 100%
- Part 3 (4 marks): 12.5%
- If hit rate was given instead of miss rate, 1 mark was taken off the relevant part(s).
- No partial marks given for working out.

Question 6

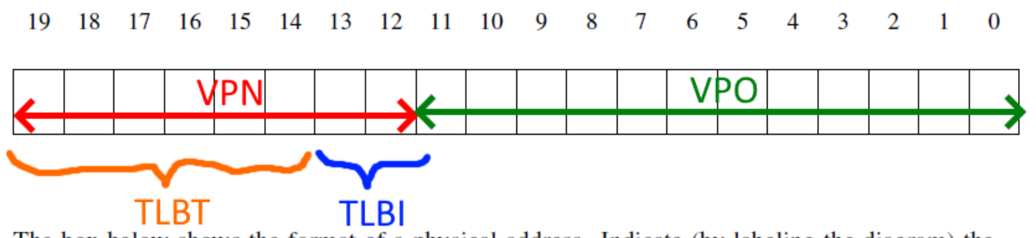
Part 1:

- Page size is 4096 => page offset is 12 bits
- So virtual page number is 8 bits and physical page number is 4 bits
- For accessing the TLB, VPN is split into TLB tag and TLB index. There are 4 sets in the TLB so the TLB index is 2 bits, and the TLB tag is 6 bits. Note that the tags shown in the TLB do all fit within 6 bits.
- So answer is:

A. Part 1

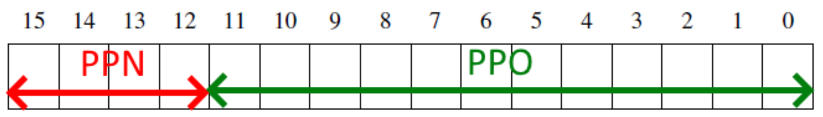
(a) The box below shows the format of a virtual address. Indicate (by labeling the diagram) the fields (if they exist) that would be used to determine the following: (If a field doesn't exist, don't draw it on the diagram.)

VPO	The virtual page offset
VPN	The virtual page number
TLBI	The TLB index
TLBT	The TLB tag



(b) The box below shows the format of a physical address. Indicate (by labeling the diagram) the fields that would be used to determine the following:

PPO	The physical page offset
PPN	The physical page number



Part 2:

- For physical address 0x07A50:
  - VPN is **0x07** and VPO/PPO is **0xA50**
  - TLBI is **0x3**
  - TLBT is **0x1**
  - TLB miss (none of the tags in set index 3 match)
- Page fault occurs (entry in page table for VPN 0x07 does not have valid bit set)
- So blank answers for rest

- **5 marks for Part 1**
  - Identifying page offset and hence VPO, PPO and PPN: 2.5 marks
  - Identifying TLBT and TLBI: 2.5 marks
- **5 marks for Part 2**
  - 1 mark for each of:
    - Virtual address (0.5 if they get the virtual address in binary correct but the wrong VPN based on what they said the VPN bits were)
    - TLB Index
    - TLB Tag
    - TLB Hit?
    - Page fault and physical address
      - If said a page fault occurs but still gave a physical address/PPN, marks were not given.

Question 7

- Address being freed is 0x500b010
- Address given is start of the payload; header of block is 4 bytes back at 0x500b00c
- The header is 0x00000011
  - Last 3 bits are 001, indicating previous block is free
  - Block size is 0x10 = 16
- Next block is at 0x500b01c, with last 3 bits 011, indicating it is occupied
- So when freeing this block we merge it with the previous one
- The new free blocks hence starts at 0x500affc, has size 32 = 0x20, has previous block allocated (since the block that was there says prev block is allocated), and is itself free. So the header and footer of this block is set to 0x00000022. The next block also now needs to say the previous block is free. Other than that, no changes to the heap.
- So answer is:

Address	
0x500b028	0x00000013
0x500b024	0x500b1008
0x500b020	0x500b512c
0x500b01c	0x00000013
0x500b018	0x00000011
0x500b014	0x500b511c
0x500b010	0x500b601c
0x500b00c	0x00000011
0x500b008	0x00000012
0x500b004	0x500b601c
0x500b000	0x500b4024
0x500affc	0x00000012

Address	
0x500b028	0x00000011
0x500b024	0x500b1008
0x500b020	0x500b512c
0x500b01c	0x00000011
0x500b018	0x00000022
0x500b014	0x500b511c
0x500b010	0x500b601c
0x500b00c	0x00000011
0x500b008	0x00000012
0x500b004	0x500b601c
0x500b000	0x500b4024
0x500affc	0x00000022

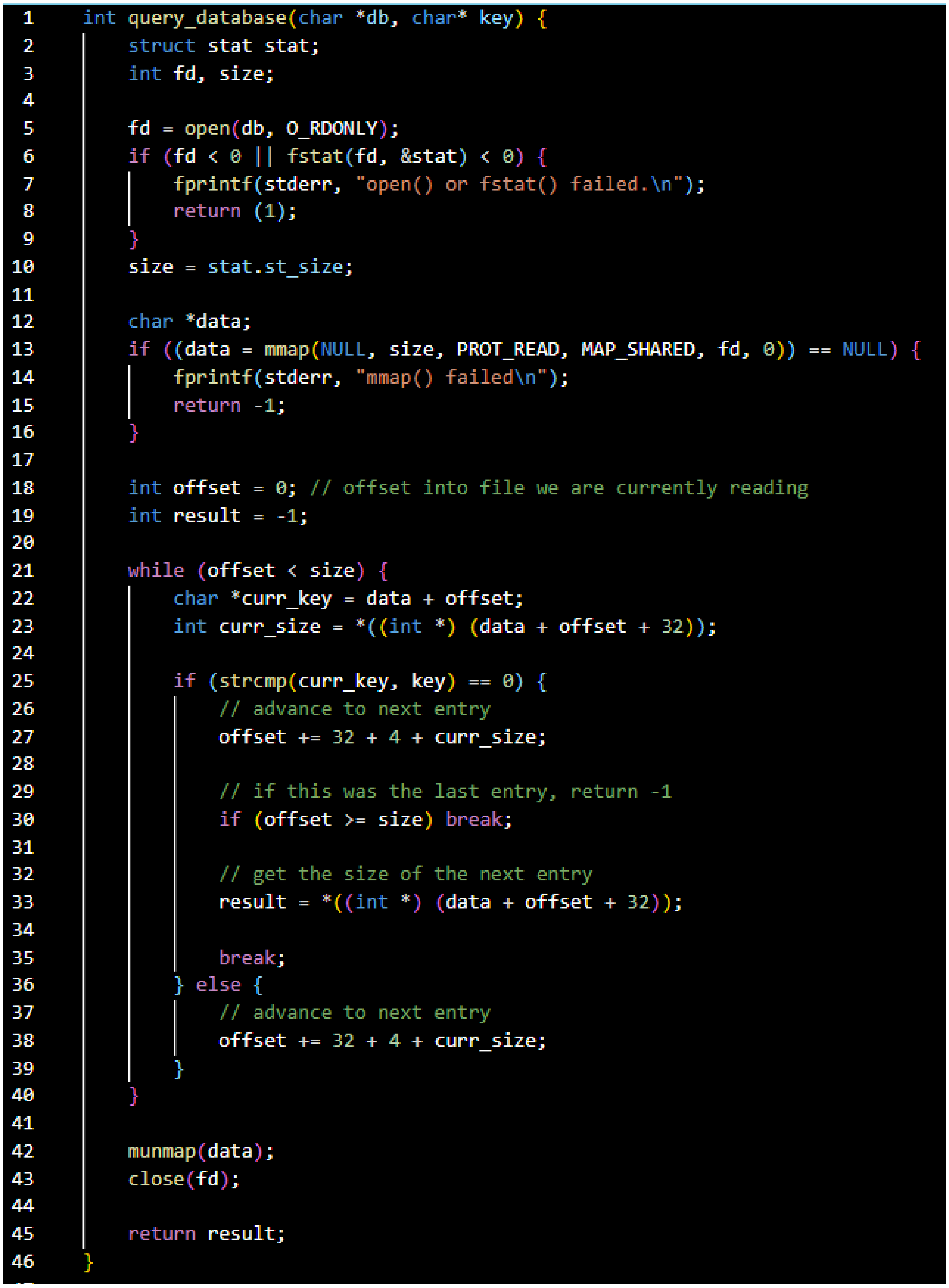
Marks (out of 10):

- **5 marks**: Noticed the block needed to be coalesced, correctly calculated the new size and put in the header/footer of the new coalesced block

- **2 marks**: New coalesced block has “previous block allocated” bit set
- **2 marks**: Remembered to update the “previous block allocated” bits of the next block
- **1 mark**: Data at address 0x500b010 is not changed

Question 8

One Possible Sample Solution:



```c
int query_database(char *db, char* key) {
    struct stat stat;
    int fd, size;

    fd = open(db, O_RDONLY);
    if (fd < 0 || fstat(fd, &stat) < 0) {
        fprintf(stderr, "open() or fstat() failed.\n");
        return (1);
    }
    size = stat.st_size;

    char *data;
    if ((data = mmap(NULL, size, PROT_READ, MAP_SHARED, fd, 0)) == NULL) {
        fprintf(stderr, "mmap() failed\n");
        return -1;
    }

    int offset = 0; // offset into file we are currently reading
    int result = -1;

    while (offset < size) {
        char *curr_key = data + offset;
        int curr_size = *((int *) (data + offset + 32));

        if (strcmp(curr_key, key) == 0) {
            // advance to next entry
            offset += 32 + 4 + curr_size;

            // if this was the last entry, return -1
            if (offset >= size) break;

            // get the size of the next entry
            result = *((int *) (data + offset + 32));

            break;
        } else {
            // advance to next entry
            offset += 32 + 4 + curr_size;
        }
    }

    munmap(data);
    close(fd);

    return result;
}
```

This question had a typo in it. It said to return the "size of the next key" when it should be the *size of the value associated with the next key*. Solutions that returned the size of the key were still accepted.

Code MUST use mmap and memory accesses from/to the mmapped region to do the task. Any solution that did otherwise (i.e. used "read") was given 0 marks.

A fully correct solution (10/10) would also have included:

- Error checking the mmap call
- Reasonably correct prot/flags arguments passed to mmap
- A call to munmap and close before returning from the function
- Handling a key that isn't present. The specific value returned in this case didn't matter, as long as it was reasonable (e.g. returning -1).

Solutions that exclude one or more of the above but still had the overall correct idea were given the majority of the marks for the question.

Question 9

Each of Part A-D were marked out of 2.5.

Part A (explain why this code will not work correctly):

- Each thread is passed a pointer to i, the loop variable used when creating the threads, which lives on the main function's stack. But there is a race condition because the main thread is updating i during the for loop while the threads are trying to read it. So e.g. main thread may have incremented i to 2 before thread 1 reads it.

Part B (explain the fix):

- This correctly fixes the issue because the main thread will be waiting for each thread it creates to finish completely before modifying the value of i, ensuring that any thread will have dereferenced vargp before it is changed.

Part C (explain the problem with fix):

- Only one thread is ever running at a time because main thread waits for the thread to finish before creating the next one, so there is no concurrency and the program is essentially serialised

Part D (Own Fix):

- Multiple different solutions to this question could be possible - it's just any modification that prevents multiple threads from using the same "i" value in memory.
- The intended solution was to set each index in the otherwise unused "array" variable, and pass the address of each variable in the array:

```
int main(void) {
 pthread_t tids[N];
 int i;
 pthread_mutex_init(&lock, NULL);
 for (i = 0; i < N; i++) {
   array[i] = i;
   pthread_create(&tids[i], NULL, foo, &array[i]);
 }
 for (i = 0; i < N; i++)
   pthread_join(tids[i], NULL);
 pthread_mutex_destroy(&lock);
 return 0;
}
```

- Another possibility was to malloc an integer each iteration. Since the question said to only modify the create loop this would result in a memory leak (since you can't free it safely). However, it does fix the race condition so this approach was given full marks.
- A solution that suggests passing the value of i directly (i.e. just removing the &) won't work because the threads dereference the vargp value causing the code to segfault. Partial marks are still given in this case.
- If a fix was suggested that looked mostly correct but still included "pthread_join" in the same loop, partial marks were given.