Giancarlos Marte

Question 1:
1a) (The Banker’s Algorithm for a Single Resource), Chp 6.5.3, pg 458-461: The banker's algorithm was new and interesting to me because of the fact that it can guarantee a safe state. Absolutes for useful things are often rare, which is why I find it so interesting. The way it works is also intriguing as it is simple, but so effective.

1b) (Virtual Machines on Multicore CPUS), Chp 7.8, pg 501: Deduplication was a new and interesting concept to me. I had always thought that virtual machines must stay completely isolated from one another even if they are identical in certain ways. To have multiple virtual machines share specific pages and memory seems like a very strange thing as I thought it would lead to a lot of problems in terms of things like complexity and security.

1c) (Load Balancing) Chp 8.2.7, pg 576-579: Multi computer scheduling was an interesting section. I expected a lot of the algorithms to be similar to multiprocessors scheduling, but a lot of them were quite different. The sender-initiated distributed heuristic algorithm was the most interesting as I felt I could understand it much better than ones like the graph-theoretic deterministic.

1d) (Passwords - UNIX Password Security) Chp 9.4.1, pg 640-641: The way unix handles password security was new and interesting to me. I did not know that a function was used each time a user puts in their password. I also did not know that those encrypted passwords are stored in one password file as I thought something like that would be too simple and insecure.

1e) (Binding Time) Chp 12.3.5, pg 1061 : Binding time was an interesting concept. I had always assumed that operating systems did the majority of things using late time binding as many things rely on the user making a decision of some kind. It is cool to learn that there are a lot of early binding things that I assumed were not.

Question 2:

1) (6.4) To be most effective, new checkpoints should not overwrite old ones but should be written to new files, so as the process executes, a whole sequence accumulates.
Flipped: To be most effective, new checkpoints should overwrite old ones but should not be written to new files, so as the process executes, a whole sequence does not accumulate.

2) (7.7) Device pass through allows the physical device to be directly assigned to a specific virtual machine.
Flipped: Device pass through does not allow the physical device to be directly assigned to a specific virtual machine.

3) (8.1.4) It matters whether the threads are kernel threads or user threads.
Flipped: It does not matter whether the threads are kernel threads or user threads.

4) (9.5) Specifically, many modern operating systems try to ensure that data segments are writable, but are not executable, and the text segment is executable, but not writable.
Flipped: Specifically, many modern operating systems do not try to ensure that data segments are writable, but are not executable, and the text segment is executable, but not writable.

5) (10.7) Users can be organized into groups, which are also numbered with 16 bit integers called GIDs (Group IDs).
Flipped: Users cannot be organized into groups, which are also not numbered with 16 bit integers called GIDs (Group IDs).

6) (11.4) Each thread has two separate call stacks, one for execution in user mode and one for kernel mode.
Flipped: Each thread does not have two separate call stacks, one for execution in user mode and one for kernel mode.

7) (12.3.6) When a thread switches from user mode to kernel mode, or a kernel-mode thread is run, it needs a stack in kernel space.
Flipped: When a thread switches from user mode to kernel mode, or a kernel-mode thread is run, it does not need a stack in kernel space.

8) (8.3.2) Reliable connection-oriented service has two relatively minor variants: message sequences and byte streams.
Flipped: Reliable connection-oriented service does not have two relatively minor variants: message sequences and byte streams.

Question 3:
3a)
According to https://www.tutorialspoint.com/c_standard_library/c_function_fread.htm
Prototype:
“size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream)
where:
-ptr − This is the pointer to a block of memory with a minimum size of size*nmemb bytes.
-size_t − This is the size in bytes of each element to be read.
-nmemb − This is the number of elements, each one with a size of size bytes.
-stream − This is the pointer to a FILE object that specifies an input stream.”

In my own words:
type numOfObjects fread(void *ptr, type objectSize, type numOfObjects, FILE *stream)
-ptr is used to store the contents and its minimum size is the size of the object * number of objects
-type can be char, int, etc
-objectSize is the size of 1 object
-numOfObjects is the number of objects
-stream is the FILE pointer to the stream of content.
The function fread returns the number of successfully read objects.

Variables:
int size1 = 4;
int objsize1 = 256;

int size2 = 8
int objsize2 = 128

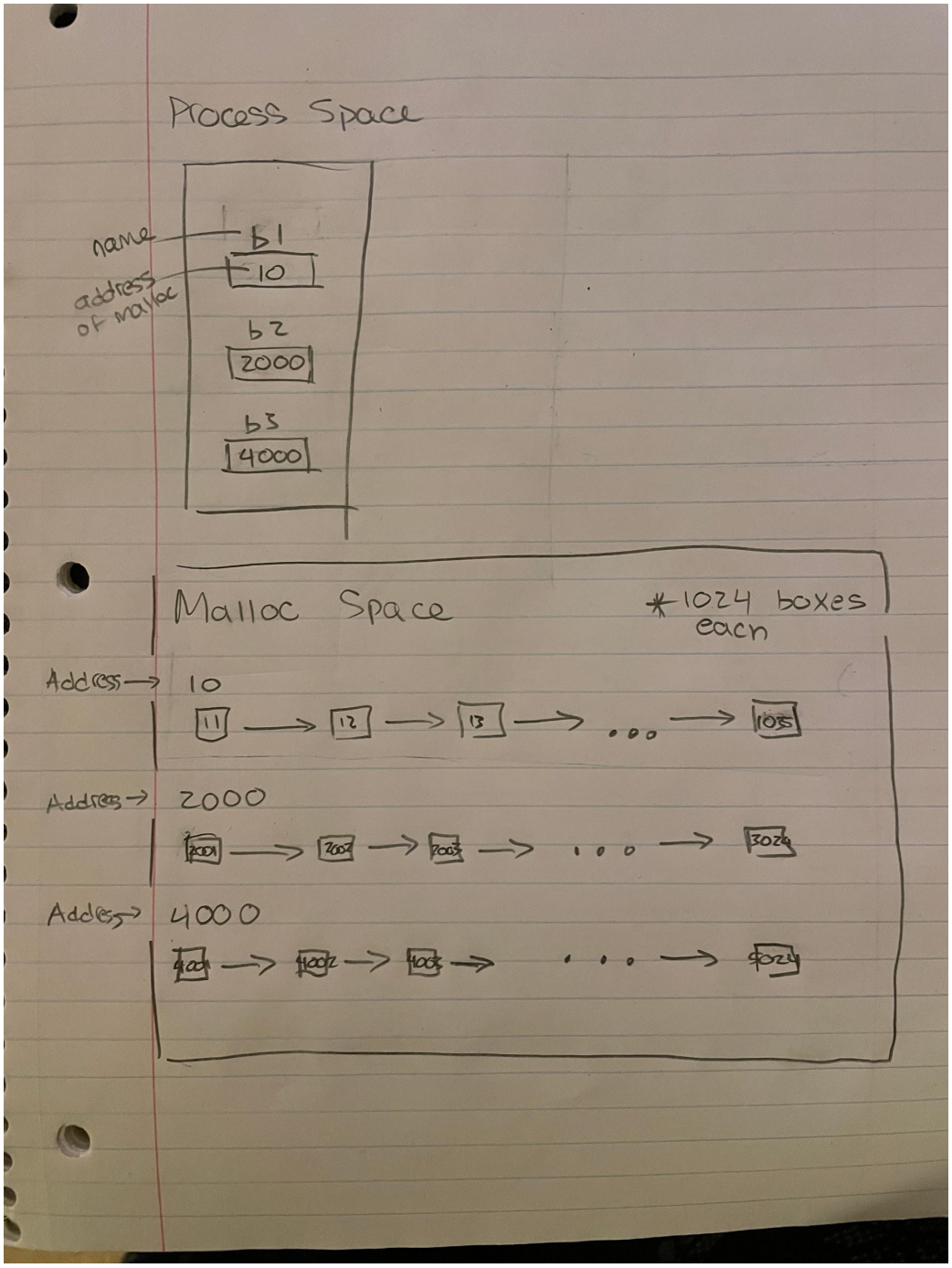
int size3 = 1
int objsize3 = 1024

Fread using variables:
int one = fread(b, size1, objsize1, fp);
int two = fread(b, size2, objsize2, fp);
int three = fread(b, size3, objsize3, fp);

Return code values:
one = 256
two = 128
three = 1024

rb means read file in binary mode. It is basically reading the file in a non translated mode.

3b)
Malloc request for buffer b:
char* b1 = (char *) malloc(sizeof(char *) * (objsize1 * size1));
char* b2 = (char *) malloc(sizeof(char *) * (objsize2 * size2));
char* b3 = (char *) malloc(sizeof(char *) * (objsize3 * size3));
// address of malloc is the pointers returned by malloc



// each box has an address increased by 1 until 1024 reached

Question 4:

// did not spend enough time on this problem

4a) The difference between both codes is one is using mutexes, which allows for a correct solution to the problem, while the other one does not and is also not a solution. The code that does not have mutexes can lock, but the one that does cannot.

4b) Using sleep (down) and wake(up) locks/semaphores can cause deadlocks.

4c) The conditions for resource deadlock are:

1) mutual exclusion condition
2) hold and wait condition
3) no-preemption condition
4) circular wait condition

4d) Removing hold and wait can cause deadlock to not occur because that is what semaphores depend on.