CS-323: OS Final Exam (2021 Fall)

January 22, 2022: 16:15 to 19:15

GENERAL GUIDELINES AND INFORMATION

1. This is a *closed* book exam. No extra material is allowed.
2. Answer your questions with a pen. Answers written with a pencil will be ignored.
3. Answer the questions in English. Answers in other languages will be graded as best as the TA's comprehension.
4. To limit the spread of COVID, we cannot answer any questions during this exam. Similarly, (and due to the limited amount of proctors), we cannot orchestrate toilet breaks during the exam.
5. You have 180 minutes, and there are 180 points. Use the number of points as *guidance* on how much time to spend on each question.
6. Write your answers directly on the test. Use the space provided. If you need more space your answer is (probably) too long.
7. Leave your CAMIPRO card on the table so it can be checked.
8. With your signature below you certify that you solved these problems on your own, that you turn in your solution, and that there were no environmental or other factors that disturbed you during this exam or that diminished your performance.
9. State all assumptions that you make above those stated as part of a question.

Question	Points	Max
Virtualization 1		24
Virtualization 2		21
Concurrency 1		45
Persistence 1		30
Persistence 2		10
Security 1		35
Security 2		15
Total		180

Name: ______________________________

SCIPER: ______________________________

Signature: ______________________________

1 Memory Segmentation

1.1 Theory (10p)

(1) Modern computing allows for multiple processes running side by side. Operating systems do not require fully cooperative and well-behaving processes. Which OS feature ensures that one misbehaving process does not have access to the memory of another process, and how? (2p) The OS virtualizes the memory so that processes do not have direct access to the underlying physical memory. The OS checks on memory accesses if the process is allowed to access the desired memory.

(2) Name and describe two benefits of segmentation-based virtual memory, compared to executing directly on physical memory. (2p) Avoid waste of memory between segment Handle large address space. Provide some isolation between segments. Easier relocation of memory

(3) Who (OS, process) is responsible for sharing virtual mem. segments (e.g., shared libs.)? Explain.(1p) The OS, the processes do not have a view of the whole memory.

(4) Which additional hardware-supported mechanism is necessary to support sharing segments between processes (i.e., the same segment is used by at least two processes safely, e.g., the code segment)? Describe how this mechanism is used and why it is necessary. (2p) Protection bits. It allows the MMU to know who has access to which segment and for which operation.

(5) It is not always possible to share segments among processes. Which segment attribute ensures that a segment is safe to share ? (1p) If a segment is read-only everywhere.

(6) A context switch typically involves changing the contents of general-purpose registers. On x86, which extra step(s) are necessary when switching context if segmentation is used? (2p) The base and bounds of each segment have to change to the ones of the new process.

1.2 Practical (14p)

A process' virtual memory is divided into 4 segments. The underlying physical memory has size 0xFFF while the virtual memory is of size 0xFFFF.
The two most significant bits of the virtual address indicate to which segment a virtual address belongs. The remaining bits indicate the offset in the segment.
The base and bound register values of some segments are indicated below (the bound register stores the size of the segment):

	Segment 0 (stack)	Segment 1 (code)	Segment 2 (heap)	Segment 3 (unused)
Base register	0x100	0x730	??? 0x400	-
Bound register	0x100	0x90	??? 0x2A0	-

(1) What is the base of segment 2 if the translation from virtual address 0x8110 returned 0x510? (1p) 0x400

(2) Given that the above translation was successful and knowing that the translation of virtual address 0x82A0 failed due to an unsuccessful bound check, compute the range of values the bound register could hold? Give the minimum and maximum value. (2p) 0x111 - 0x2A0

(3) For each of the following operations, what is the physical memory address resulting from the translation from virtual to physical? Note any errors that would occur during the operation. (6p)

Accessing the virtual address 0x4078 0x7A8

Accessing the virtual address 0xc2A0 SegFault

Accessing the virtual address 0x0812 SegFault

Accessing the virtual address 0x00D3 0x1D3

Accessing the virtual address 0x80EE 0x4EE

Accessing the virtual address 0x47C1 Segfault

(4) List all the possible virtual addresses of these physical memory addresses? (5p)

Physical address: 0x323 None

Physical address: 0x1AE 0x00AE

Physical address: 0x4EA 0x80EA

Physical address: 0x777 0x4047

Physical address: 0xAE0 None

2 Scheduling

2.1 Theory (7p)

An impatient TA is doing two tasks at the same time:

- typing an exam question in a word processor, and
- compiling a huge program in the background.

(1) Which of the two tasks is composed of short CPU bursts? Justify (1p) Typing is a sequence of short keypress.

(2) Our TA really dislikes when the keypress does not appear right away but they are fine with the compilation taking longer. Which scheduler metric (turnaround time, response time, fairness, high-throughput) will highlight our TA preference the best? Justify. (2p)

Response time because we want that a new keypress is processed right away and do not sit waiting behind the compilation.

(3) Which scheduler parameter should the TA tune in order to get best results for the metric in the previous question? Justify. (1p) Slice length. This allows the compiler to be preempted sooner, and the CPU to be given to the word processor sooner.

(4) How would you advise to deal out tickets if the OS is using a lottery scheduler and the goal is to match the TA preferences? Why? (2p)

Give all ticket but one to the word processor

The following question is not related to the previous ones.

(5) In a MLFQ scheduler, does the addition of a queue above the lowest priority one impact the response time of really short jobs? Justify. (1p)

Short jobs don't have time to fall to the queues further down.

2.2 Practical (14p)

Assume a Multi-Level Feedback Queue (MLFQ) scheduler with the following rules. Please be careful, some rules have changed compared to what you have seen in the book/course! Additions are *in italics* and deletions are ~~struck-through~~.

- **Rule 1**: When a job enters the system, it is placed at the highest priority (the topmost queue).
- **Rule 2**: Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced (i.e., it moves down one queue).
- **Rule 3**: After some time period S, move all the jobs in the system ~~one queue up (if possible)~~ *to the topmost queue* putting jobs from erstwhile higher privilege first.
- **Rule 4**: If Priority(A) > Priority(B), A runs (B doesn't).
- **Rule 5**: If Priority(A) = Priority(B), ~~B run in round-robin fashion using the time slice (quantum length) of the given queue~~ *A (first entered in this queue) runs.*

The rules should be applied in the same order as listed above.
The MLFQ is composed of 4 queues, highest priority first:

- **Priority 1 (Highest)**: maximal time allotment at this level: 1 tick
- **Priority 2**: maximal time allotment at this level: 2 ticks
- **Priority 3**: maximal time allotment at this level: 4 ticks

- **Priority 4 (Lowest):** no maximal time allotment at this level

The time period S is set to 10.
The scheduler is set to preempt the running task at an interval of 1.

Imagine the following arrival of task (top most arrived first):

Task ID	Arrival time	Task duration
T_0	0	7
T_1	0	3
T_2	2	2
T_3	5	0.99, wait for I/O during 2 ticks, then 2
T_4	9	8

(1) For each time interval, complete the queue with the task ids (in correct order) present right after the application of the scheduler rules but before execution. Also note which tasks will be running for the coming tick. (12p)

	Tick 0	Tick 1	Tick 2	Tick 3	Tick 4	Tick 5
Queue 1	0, 1	1	2	-	-	3
Queue 2	-	0	0, 1	0, 1, 2	0, 1, 2	1, 2
Queue 3	-	-	-	-	-	0
Queue 4	-	-	-	-	-	-
Next running task	0	1	2	0	0	3

	Tick 6	Tick 7	Tick 8	Tick 9	Tick 10	Tick 11
Queue 1	-	-	3	4	2,3,4,0	3,4,0
Queue 2	1,2	1,2	2	2,3	-	-
Queue 3	0	0	0	0	-	-
Queue 4	-	-	-	-	-	-
Next running task	1	1	3	4	2	3

	Tick 12	Tick 13	Tick 14	Tick 15	Tick 16	Tick 17
Queue 1	4,0	0	-	-	-	-
Queue 2	-	4	4,0	4,0	0	0
Queue 3	-	-	-	-	4	4
Queue 4	-	-	-	-	-	-
Next running task	4	0	4	4	0	0

	Tick 18	Tick 19	Tick 20	Tick 21	Tick 22	Tick 23
Queue 1	-	-	4,0	0	-	
Queue 2	-	-	-	4	4	
Queue 3	4,0	4,0	-	-	-	
Queue 4	-	-	-	-		
Next running task	4	4	4	0	4	

(2) Describe one improvement to the proposed scheduler that would lead to a better turnaround time? (2p)

Rule 5 to RR

3 Concurrency

3.1 Theory (13p)

(1) Is a read/write lock always better than a single global lock? If not, give a counterexample. (4p) No, read/write locks have more "housekeeping" costs compared to a single global lock, and will perform worse if there are not going to be multiple threads reading in parallel. For example, if we expect a workload where the number of writes greatly exceeds the number of reads, a read/write lock mechanism will perform worse due to the time spent in tracking the number of readers, which most of the time is going to be just zero.

(2) Give an example of an instance where spinlocks are preferable to mutexes. (3p) A spinlock is preferable to a mutex in an environment where the critical section is very short and we expect to acquire the lock in a very short amount of time. Having multiple cores and a low amount of context switches are also factors that would make a spinlock more performant.

(3) Why are spinlocks considered "unfair"? (3p) A thread that is waiting on a spinlock is not guaranteed to eventually take the lock as long as there are other threads racing on the same lock. Contrary to queue locks, spinlock mechanisms do not care of the waiting time a thread has spent on a given lock - it only depends on which thread is able to claim the lock first.

(4) A naive implementation of read/write locks such as the one seen during lectures can also be unfair—in fact, a writing thread could have to wait a very long time until there are no reading threads using the same lock. Can you explain a possible modification that can avoid this problem? (3p) We could add a second counter that we fix to an initial value of our choice that defines the maximum number of consecutive reads before a write can happen. Each time a read is performed, the counter is decreased. When the counter reaches zero, all reads are paused and the write lock is released. As soon as a write happens, or if there are no writes pending, the counter is reset to its original value, and a new batch of reads can be performed.

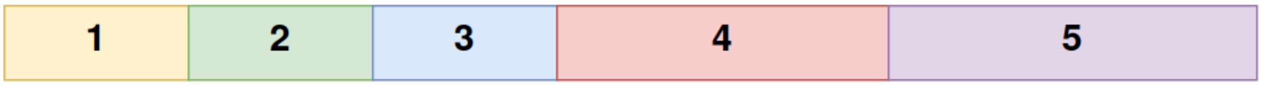
3.2 Practice (32p)

In this exercise, we provide the implementation of the linked list of lab 2. Your job is to modify methods `list_insert` and `list_find`, making them thread-safe using a readers-writer lock. The readers-writer lock is a part of each node for fine-grained locking, i.e., multiple threads can concurrently read or write to disjoint parts of the list: locking is only required when multiple threads read or write to the same node in the list. You should modify the `node_t` structure to add the required elements for the lock. You are free to use the following locking primitives in your code: `sem_init`, `sem_post`, and `sem_wait`.
This exercise requires you to also implement the functions:
`rwlock_acquire_readlock`, `rwlock_acquire_writelock`, `rwlock_release_readlock`, `rwlock_release_writelock`.
This question consists of two parts: implementing mutual exclusion and implementing the primitives. The points are distributed 50/50 between the two parts. The two parts will be graded individually i.e., if one part is not correct you may still get points in the other part.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>

node_t *head = NULL;

void rwlock_acquire_readlock(node_t *rw);
void rwlock_release_readlock(node_t *rw);
void rwlock_acquire_writelock(node_t *rw);
void rwlock_release_writelock(node_t *rw);

typedef struct node_t {
  int data;
  int key;
  node_t *next, *prev;


sem_t lock;
sem_t wlock;
int readers;


} node_t;


// Insert a new node after the one provided as argument
// NOTE: for simplicity, edge cases (such as inserting as head or
// tail of the list) are not checked.
void list_insert(node_t* node, int key, int data) {
        node_t *new = ( node_t*) malloc(sizeof( node_t));
        new->key = key;
        new->data = data;
        new->next = node->next;
        new->prev = node;
        node->next = new;
        new->next->prev = new;
}

// Return a node with the given key
node_t* list_find(int key) {
        node_t* current = head, *new_current;
        if(head == NULL) {
                return NULL;
        }
        while(current->key != key) {
                if(current->next == NULL) {
                        return NULL;
                } else {
                        current = current->next;
                }
        }
        return current;
}

void rwlock_acquire_readlock(node_t *rw) {


sem_wait(&rw->lock);
rw->readers++;
if (rw->readers == 1)
        sem_wait(&rw->wlock);
sem_post(&rw->lock);


}

void rwlock_release_readlock(node_t *rw) {


sem_wait(&rw->lock);
rw->readers--;
if (rw->readers == 0)
        sem_post(&rw->wlock);
sem_post(&rw->lock);


}

void rwlock_acquire_writelock(node_t *rw) {


sem_wait(&rw->wlock); // last r, also release wlock


}
```

```
void rwlock_release_writelock(node_t *rw) {

sem_post(&rw->wlock); // last r, also release wlock

}

void safe_list_insert(node_t* node, int key, int data) {

node_t *new = ( node_t*) malloc(sizeof( node_t));

new->key = key;
new->data = data;
sem_init(&new->lock, 0, 1);
sem_init(&new->wlock, 0, 1);
new->readers = 0;

rwlock_acquire_writelock(node);
rwlock_acquire_writelock(node->next);
new->next = node->next;
new->prev = node;
node->next = new;
new->next->prev = new;
rwlock_release_writelock(node);
rwlock_release_writelock(new->next); // it is new->next !!

}

node_t* safe_list_find(int key) {

node_t* current = head, *new_current;
if(head == NULL) {
   return NULL;
}

rwlock_acquire_readlock(current);

while(current->key != key) {
   if(current->next == NULL) {
           rwlock_release_readlock(current);
           return NULL;
   } else {
           rwlock_acquire_readlock(current->next);
           new_current = current->next;
           rwlock_release_readlock(current);
           current = new_current;
   }
}

rwlock_release_readlock(current);
return current;

}
```

4 Lab 3 File System (30p)



(1) In lab 3 you have implemented a simple file system. The above figure represents the layout of the emulated disk used by this file system. Provide a legend with the names of the different sections and specifying what is stored in them. (5p)

1. Superblock: stores the metadata of the file system (magic number, block size, number of blocks, ...)
2. Inode bitmap: a bitmap indicating the free inodes.
3. Data bitmap: a bitmap indicating the free data block.
4. Inode blocks: the section in which the inodes are stored.
5. Data blocks: where the actual data is stored.

(2) Although being present on disk, two of those sections were not used by your simple file system. Name the two sections and explain their purpose. (4p)

The two ignored sections are the inode and data bitmaps, their purpose is to improve the performance of the file system when creating new files.

(3) Suppose you want to extend the file system API of lab 3 to support file creation. Will the fact that we ignored those two sections have any drawbacks? Justify your answer. (2p)

Yes! Without using them, when creating a new file, we have to traverse all inodes in order to find a free inode block and identify the free data blocks.

(4) The user, the operating system and a process view files in three different ways. List (name) the three different views and justify their existence. (6p)

1. User: file name, it's human readable.
2. Operating System: inode, are unique, unambiguous and allows to store all needed metadata.
3. Process: file descriptor, keeps track of per-process state.

(5) Describe in one sentence the open file table used in lab 3 and what it is used for. (2p)

The open file table had an entry for each file descriptor, associating the latter with the corresponding inode and the seek offset.

Note: in the following questions we assume the open file table to be global, i.e. shared between all processes. Open and read operations happen sequentially, following the questions order.

(6) Explain what happens under the hood when process A opens `file.txt` (we assume the file is valid, i.e. it can be opened), the system returns file descriptor 3. Specify the role of each of the three views and how they interact. (4p)

1. Find the inode corresponding to the provided file name (file.txt in this case). The file name is one of the metadata stored in the inode.
2. Find an unused file descriptor.
3. Add an entry to the open file table associating the file descriptor to the inode and the seek offset (0 at the beginning).
4. Return the file descriptor to the user.

(7) Process A reads 72 bytes from `file.txt` (assume the file has size 128 bytes). List which views are used to perform the read operation specifying what they are used for. (3p)

The file name is not used anymore. The file descriptor is a unique ID that can be used to access the open file table. This allows us to retrieve both the seek offset and the inode. The latter is used to verify that the read will not exceed the file size as well as for retrieving the address at which the file starts.

(8) Will there be any difference in the open file table before and after the read? Justify. (1p)

Yes, the seek offset will be updated from 0 to 72.

(9) Now process B - another process - opens the same file file.txt, the system returns file descriptor 4. Does this result in any change in the open file table? Justify. (1p)

Yes, a new entry is added.

(10) If process B reads 100 bytes from file.txt, will the open file table change? Justify. (1p)

Yes, the seek offset of entry of process B file descriptor will be changed from 0 to 100.

(11) With the current design, process B can access file descriptors of process A and vice versa, which results in a security vulnerability. How would you modify the design in order to prevent this? Justify your answer. (1p)

Make the open file table per-process instead of global.

5 Flash Drive Scheduling (10p)

(1) We plug a flash drive into a laptop. Assume the flash drive uses Programmed IO. Which one of the following situations will result in a large CPU cost? Justify. (2p)

1. Reading the root node (few bytes).
2. Reading a huge file.

Situation 2. In PIO the CPU has to explicitly copy the data to/from the device one word at time. The CPU overhead is thus proportional to file size.

(2) Propose and explain an alternative data transfer method that can handle both situations without causing CPU overhead. (2p)

Direct Memory Access (DMA). With this approach the CPU has to communicate to the DMA where the data lives in memory, how much data to copy, and which device to send it to. Then the CPU is done with the transfer and can move on as the DMA will take care of copying the data.

(3) Now assume you want to write a file to the flash drive. The drive can choose between two policies to acknowledge the write as complete: write back caching and write through. Describe them. (4p)

Write back caching: the drive acknowledges the write as completed when it hasput the data in its memory.

Write through: the drive acknowledges the write after it has actually been written to disk.

(4) If your goal is to have the least latency possible, which policy will you pick? Is there a case in which this policy can cause some problems? If yes, explain it and propose a solution. (2p)

Write back caching. It can result in problems when the order between subsequents writes has to be enforced. Write barriers solve this problem.

6 Sanitizers

AddressSanitizer (ASan) is a program sanitizer that enforces memory safety policies, detecting out-of-bound accesses, use-after-free, and use-after-return bugs, among others.

6.1 Theory (15p)

(1) Can ASan be used as a security mitigation? Justify with 2 reasons. (4p)

No.

- It provides probabilistic guarantees and can be bypassed, given knowledge of its use.
- It introduces high performance and memory overheads.

(2) What changes in memory layout does ASan enforce to detect buffer over/under-flows? (2p)

By surrounding the allocated objects with poisoned memory (red zones), contiguous out-of-bounds access will be detected once those red zones are reached.

(3) How is ASan useful in fuzzing? Support your answer with an example showing how the use of ASan helps find a bug. (2p)

ASan can detect more bugs at runtime, and detect them earlier. When fuzzing, if the input triggers a bug and results in a benign buffer overwrite, the program may continue execution without observable signs of malfunction. With ASan, the overwrite is flagged immediately and the bug is reported through an explicit crash.

(4) How does ASan hinder fuzzing? (1p)

Performance cost reduces the number of executed tests per second.

(5) Consider the following code:

```c
struct my_struct {
    char name[4];
    int value;
};
struct my_struct global_1, global_2;
void main(void) {
    const int total = 3;
    pid_t pids[total];
    int task = 0;
    do {
        pids[task] = fork();
        if (pids[task]) {
            wait();
            ++task;
            continue;
        }
        switch (task) {
        case 0:
            struct my_struct obj;
            strcpy(obj.name, "MAIN");
            break;
        case 1:
            char my_name[4];
            strcpy(my_name, "MAIN");
            break;
        case 2:
            char *ptr = &global_2;
            ptr[-0x18] = '\x41';
        default:
            break;
        }
        break;
    } while (task <= total);
}
```

Assume that global objects are separated by a red zone of 16 bytes. For every value of `task`, which lines of code trigger an ASan crash when the program runs (feel free to mark the code above)? (6p)

- When task == 0, an intra-object overflow occurs, which goes undetected by ASan.
- When task == 1, a stack buffer overflow occurs, and ASan crashes.
- When task == 2, `ptr[-0x18]` points to `&(global_1.name)`, which is a valid region.
- When task == 3, a stack buffer overflow occurs in the parent process when accessing `pids[3]`, and ASan crashes.

6.2 Exercise (20p)

To conserve space, ASan packs the access bits of each 8 consecutive bytes in memory into 1 metadata byte. Moreover, ASan assumes that objects in memory are always allocated at 8-byte-aligned boundaries, such that only the first $N <= 8$ bytes in each group of 8 consecutive bytes are accessible (i.e., no object starts in the middle of such a group).

(1) On a 64-bit Linux/x86_64 system, when an ASan-sanitized process is launched, a chunk of its memory is allocated for ASan's metadata (shadow memory). How big is this region? How is it possible to allocate this much memory? *Recall: On 64-bit Linux, userspace memory is practically limited to 47-bit addresses.* (3p)

To cover a 47-bit address space, the shadow memory must have a size of: $(2^{47} >> 3)$ bytes = 16 TiB. This is only virtually allocated.

(2) By default, ASan's shadow memory starts at a `SHADOW_OFFSET=0x00007fff8000`, and the memory layout looks as follows:

Begin	End	Description	Permissions
0x000000000000	0x00007fff7fff	LowMem	???
0x00007fff8000	0x00008fff6fff	LowShadow	rw-
0x00008fff7000	0x02008fff6fff	ShadowGap	---
0x02008fff7000	0x10007fff7fff	HighShadow	rw-
0x10007fff8000	0x7fffffffffff	HighMem	???

To access the shadow bytes of an arbitrary memory address, it is first divided by 8 then added to `SHADOW_OFFSET`. What purpose does the ShadowGap region serve? Explain. (2p)

To preserve the integrity of the shadow memory region. Addresses that belong to this region map to the ShadowGap, which has no read/write permissions, and thus a segmentation fault occurs.

(3) Explain how ASan poisons memory around objects on the stack. (3p)

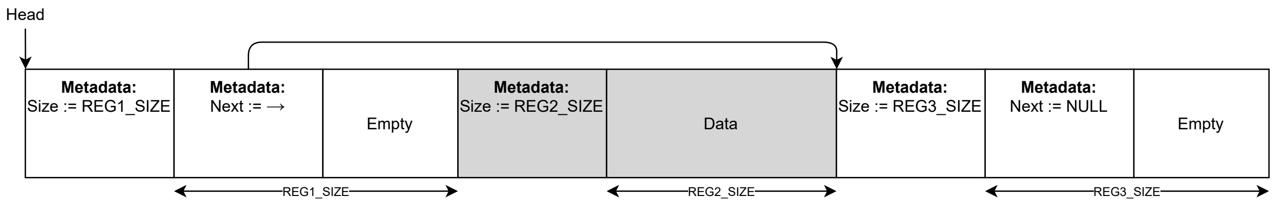
ASan instruments each function's prologue and epilogue to add red zones around stack-allocated objects and destroy them on return. On entry, it initializes the shadow bytes of the red zones around each object to indicate invalid access. On exit, those bytes are marked as accessible again.

(4) To instrument the heap, ASan provides its own memory allocator, with a 4 TiB area starting at `kAllocatorSpace=0x600000000000`. When a data region is allocated, ASan resets the shadow bytes of that region to indicate valid access, and surrounds the region by 2 red zones. Assume that:

- ASan uses the list allocator you've seen in the labs.

- The heap is initialized and empty, and malloc had not been called before.
- Accesses to the heap metadata by malloc/free are not instrumented by ASan.
- The size of a reserved region's header (metadata) is 48 bytes.
- The size of a red zone is 16 bytes.

The memory layout of an arbitrary heap using the list allocator is provided here for reference:



a. How big is the additional physical memory allocated when malloc(80 KiB) is called? *Recall: Copy-on-write* (4p)

   To allocate a region on the heap, the free region's header is modified to indicate the size of the new reserved region, and the free region is split, such that the second part's region header is modified to indicate the remaining free size. As such, two page accesses are incurred, but the first page is already mapped (since the heap is initialized). So CoW results in the second page (4 KiB) being physically allocated in addition. Also, to initialize ASan's shadow memory, ~10 KiB must be set. This incurs three new page allocations (12 KiB). Finally, the total physical memory allocated in addition is 16 KiB.

b. What's the address range in shadow memory that gets modified? (4p)

   Start: `0x00007fff8000 + (0x600000000000 + 0x30) >> 3 = 0x0c007fff8006`

   End: Start `+ 0x10 + 0x014000 + 0x10 = 0x0c008000c026`

c. How are the above quantities affected if calloc was used instead of malloc? (4p)

   The allocated region on the heap must be set to 0, and that incurs physical allocations proportional to the size of the region. Specifically, 21 pages will be occupied, of which 20 pages will be newly allocated (80 KiB). The shadow memory allocations remain the same, since ASan initializes the shadow bytes either way (12 KiB). So in total, the physical memory allocated in addition is 92 KiB.

7 Mitigations (15p)

(1) Address Space Layout Randomization (ASLR) shuffles the addresses of memory sections to reduce predictability of data and code locations and increase the difficulty of mounting exploits.

Long-running services (e.g., network servers) may be especially disadvantaged by ASLR. Explain why (answer not related to the next part). (3p)

If the attacker can exploit an ARP, they could exfiltrate ASLR offsets from the same process with multiple requests and use them to bypass ASLR.

(2) Network servers also tend to have a fork-exec loop for processing client requests. Why is it not feasible for the forked child to have a new random memory layout? (5p)

Memory pointers in the child are inherited from the parent. Re-randomizing the layout at fork-time would require re-adjusting all pointers in memory, which is an expensive operation.

(3) If ASLR is bypassed, code reuse attacks are again possible. To mitigate that, control-flow integrity (CFI) can be enforced. CFI instruments indirect jumps (branch registers and function returns) and performs a membership check of the jump target in the set of valid targets. If CFI is applied statically, it is not possible to obtain a comprehensive set of targets for each jump location. Instead, a static set is constructed based on some validity criteria. List 3 different examples of such criteria. (3p)

(1) Address-taken functions; (2) Function arity; (3) Function prototype

(4) The imprecision introduced by static CFI leaves some gaps for an attacker to construct CFI-compliant gadget chains. However, even with a completely precise CFI mechanism, data-only attacks can still be mounted. Why are such attacks possible? (4p)

Because CFI does not provide data-flow integrity. CFI-compliant control flow can still lead to arbitrary code execution by manipulating data flow and “bending” control flow in the desired manner.