CSE 422S – Spring 2021 Exam 2

Released 4pm Thursday, March 25th, 2021

Name (please print):

Tara Renduchintala

This exam focuses on concepts and details we have covered in the assigned readings, lectures, and studios. Please answer each question as completely and correctly as possible. Partial credit may be given for incorrect answers that show understanding of the material.

During the exam you may use your notes, text books, and on-line sources of information, but you may not post content from this exam anywhere at any time. Communicating or sharing materials with other people during the exam is not permitted.

Please sign or print your name below to indicate your understanding of, and agreement to abide by, the exam conditions described above:

Tara Renduchintala

Exam scoring (to be completed by the grader)

Question	Possible	Score
1	12	
2	12	
3	12	
4	12+1	
5	12	
6	12	
7	8	
8	8	
9	12	
Total	100+1	

1. (12 points) which syscalls are used to open and close a file?

The syscall to open a file is open(pathname, flags, mode) and the syscall to close the file is close(file descriptor).

When using those syscalls, how is the identity of the file represented?

The identity of the file is represented using a file descriptor that is then used to refer to the file.

Which standard I/O library functions are used to open and close a file?

To open a file, you use fopen(const char * path, const char * mode) and fclose(FILE * stream)

When using those standard I/O library functions, how is the identity of the file represented?

The file is represented as a file stream and is used by functions by passing a pointer that points to the file stream as the parameter.

Which syscalls are used to output data to a file and to input data from a file?

To output data, the syscall used is write(int file descriptor, const void * buffer, size_t total bytes) while to input data from a file, the syscall used is read(int file descriptor, const void * buffer, size_t total bytes.)

Which standard I/O library functions are used to output formatted data to a file and to input formatted data from a file?

To output formatted data to a file you use fprintf(FILE * stream, const char * format). To input formatted data from a file you use fscanf(FILE * stream, const char * format).

2. (12 points) Linux organizes page tables into four levels: Page Map Level (PML), Page Global Directory (PGD), Page Middle Directory (PMD), and Page Table Entry (PTE). If 9 bits are used to index the entries in one of those tables, how many entries can that table hold?

Number of Entries = 2^Number of bits used to index = 2^9 = 512 Entries

Please explain briefly what the kernel must check first, when it is translating a virtual address and it finds a page table entry that does not have a physical page mapped to it.

The kernel must first check if the virtual address of the process to be accessing is valid. For example if it is trying to read or write, it must be memory readable or memory writeable, respectively.

Please explain briefly what the kernel does if that check succeeds.

If the check succeeds, then the page tables will get updated with the physical address so that the hardware from then on will know how to complete the translation. The virtual address and the physical address are now connected by the page tables. Entries at every level of the page table will be mapped.

Please explain briefly what the kernel does if that check fails.

If the check fails, then the kernel raises a Segmentation Fault by sending a SIGSEGV signal. Nothing in the page tables get updated and no memory is allocated.

3. (12 points) Briefly, what is avoided by always acquiring a given set of locks in the same order, within any thread that needs to hold those locks at the same time?

By acquiring a given set of locks in the same order, a deadlock is avoided. This way, no two threads are fighting for a lock while holding onto the lock that the other needs.

Briefly, how does having each of those threads always acquire those locks in the same order prevent that?

If all the threads acquire the locks in the same order, the next resource that is needed will always be free. For example. if thread A has lock 1 and then needs lock 2, it can release lock 1 such that thread B can obtain it and then get lock 2. Thread B can then obtain lock 2 once Thread A releases it. If there was no ordering, thread B could obtain lock 2 and then there would be a deadlock as thread B wants lock 1 while holding lock 2 and thread A wants lock 2 while holding lock 1. Neither ever releases its lock, resulting in a deadlock. By maintaining an order, this deadlock would not be possible.

Briefly, what is one advantage of using a spin lock over using a sleeping lock?

A sleeping lock incurs a bit of overhead, specifically due to the context switches that are needed to switch out and into the blocking thread. This requires a lot more coding than a spin lock does. When the duration of the need for a lock is less than the duration of two context switches, it is best to use a spin lock.

Briefly, what is one advantage of using a sleeping lock over using a spin lock?

When the duration of a lock is going to be long, it is best to use sleeping lock. This is because you are not wasting the processor time by constantly waiting for the lock to become available. Additionally, if the process needs to be able to sleep while holding a lock, one would need to use a sleeping lock because the process cannot sleep while holding a spin lock.

4. (12 points) Briefly, how does a fast user space mutex differ from a standard mutex or semaphore?

When the lock is uncontended, the futex (fast user space mutex) does not require any context switching to the kernel. If a futex attempts to get a lock and gets a 'locked' back it tries again. If a 'waiters' is received then it goes to sleep. The advantage is that the second attempt allows the lock to have a higher probability of being acquired. The common mutex tries to get the lock and if it fails it goes straight back into the sleep queue. There is no second attempt to try and grab the lock.

What operation is implemented by the following code?

```
int expected, desired;
expected = UNLOCKED;
desired = LOCKED;
while (!__atomic_compare_exchange(p,
       &expected, &desired, 0,
       __ATOMIC_ACQ_REL, __ATOMIC_ACQUIRE)){
    expected = UNLOCKED;}
```

This is the lock operation for the spin lock.

Briefly, why does that code need to (possibly repeatedly) reset the value of the `expected` variable to be `UNLOCKED`?

If the function ends up failing, then the expected value that is currently being pointed to by the "expected" variable may be overwritten. Therefore, the "expected" variable must be reset to UNLOCKED to ensure that the lock is being set correctly when the function is successful.

Briefly, what is guaranteed for an `atomic_t` variable that is not guaranteed for a `volatile int` variable?

An atomic_t variable will only be modified by one process at a time. A volatile int variable can be accessed by multiple threads at one time, therefore the value that is being used by each thread could be different – the new value essentially depends on which thread overwrites the other's changes. Thus, the result is a race condition. An atomic variable ensures that no two processes interleave (ie. only one of them is in the critical region at one time). This ensures that the value is being correctly updated.

Extra credit (1 point): Please name another atomic function besides `__atomic_compare_exchange` that we have used in the studio exercises or the lab assignments.

We used atomic_set(), atomic_add(), __atomic_sub_fetch(), __atomic_add_fetch() and __atomic_store()

5. (12 points) Briefly, what does the `ftruncate` call do to a shared memory region?

The purpose of a ftruncate call to a shared memory region is to resize of the memory region to be of the size we want.

Briefly, why is using shared memory for communication between two processes more efficient than using other inter-process communication mechanisms we've studied?

Using shared memory is more efficient because processes can share a physical page as an inter-process communication channel such that the processes can see updates made to the memory by the other process. Once the mapping is set up, the writing into physical memory happens at the speed of the process's own memory rather than continuing to have to make syscalls to send information using read() and write(). T

Briefly, how does the layout of virtual addresses relative to to physical addresses differ between user-space virtual memory and kernel virtual memory?

The kernel virtual memory is always pre-paged, meaning that when the system boots all the physical memory is mapped in contiguous order to the kernel virtual address space. The user-space virtual memory mappings are created at runtime. An earlier page in the user-space virtual memory may map to a later page in the physical memory. User-space virtual memory is not guaranteed to be contiguous and is also not guaranteed to be in order. Kernel space virtual memory is both in order and contiguous.

Briefly, in the copy-on-write optimization following a fork() call, what does setting the parent and child processes' page table entries to unwriteable ensure will happen if either process then calls write() at one of its virtual addresses?

It ensures that the physical page gets copied and then the process writes to the copy leaving the other physical page unchanged.

If either process wants to write() to its virtual address then the page table entry will trigger a page fault and will check for validity. Since it is valid, it will make a copy of the physical page and point the page table entry to the copy of the physical page. Then, changes can be made to this copies physical page without dirtying the other processes' physical page. Then, the page table entries for both processes can be marked as writeable.

6. (12 points) Next to each of the statements below, please write T next to it if it is true or write F next to it if it is false.

F - Demand paging maps physical pages as soon as virtual addresses are allocated.

T - Prepaging maps physical pages as soon as virtual addresses are allocated.

F - Shared memory always maps to the same virtual addresses in the processes that share it.

T - Formatted output to a file may be buffered both in user space and in the kernel.

T - Direct I/O improves most applications' performance.

F - Code without deadlocks is always free of data races.

T - Code without data races is always free of deadlocks.

T - A process can use a syscall to expand its heap.

T - A single syscall can read data into multiple buffers.

F - All content in the virtual file system is backed up by a hard disk or other persistent storage device.

T - The MMU performs virtual to physical memory address translations.

T - Most 32-bit architectures have physical memory pages that are each 4KB in size.

7. (8 points) Briefly, what does the `percpu` interface do?

The percpu interface gives core specific storage so that there is no contention from activities on other cores. It disables kernel preemption and gives memory access through alloc and put on per-cpu data.

Briefly, what is stored in a cache created by the slab allocator interface?

The cache is managing the slab as a single allocation and deallocation call. Each cache stores slabs of kernel objects. Each slab stores a unique object. If you exceed a slab's worth of kernel objects, the cache can have another slab that is allocated for that object.

If a kernel module calls vmalloc() in its init() function, what function should it call in its exit() function?

If a kernel module calls vmalloc() in the init() function, it should call vfree() in the exit function.

What function should be used when allocating memory, to improve system security by ensuring that (possibly sensitive) data that had been written to a physical page is overwritten?

The function that should be used is get_zeroed_page(). This is so that all of the data in the physical page is overwritten before it is reallocated, therefore, no data is randomly just stumbled upon when the page is freed.

8. (8 points) Briefly, what is contained in an operations object in the Linux virtual file system?

The operations object is a component of a primary object (ie. a file) that describe the methods that the kernel can invoke against the primary object. The object could be a filesystem, a specific file, a specific directory entry, or an open file. The operations object maintains a structure of pointers to functions that are used to operate on the parent object.

Briefly, how do those operations objects support a form of object-oriented programming within the Linux kernel?

Although they are not objects themselves, they represent instances of an object with its associated data and methods. In this manner, they enable object oriented programming by representing a data structure that can be manipulated using methods and containing data that can be retrieved.

Briefly, when is a new file object added to the virtual file system data structures corresponding to a user-space process?

The file object is created and added to the virtual file system data structures when there is an open() system call. There can be multiple file objects in existence for the same file. The file object that is created by open() in a process will only represent that process's view of the open file.

Briefly, what does the bootloader program do, once it is loaded from the /boot directory?

Once the bootloader program is loaded, it will load the kernel. The kernel's entry point is start_kernel().

9. (12 points) In the blank next to each phrase on the left, please write the letter for the description that best matches it and that it best matches.

J - atomic	A. sleep/wake if a lock is not yet acquired
L - deadlock	~~B. retries without sleeping until lock is acquired~~
K - data race	~~C. represents a specific mounted filesystem~~
F – file stream	~~D. represents a file or directory~~
G - file descriptor	~~E. exponent for number of pages to allocate~~
H - file offset	~~F. created/used by `stdio` functions~~
E - order	~~G. created/used by syscalls~~
A - futex	~~H. modified by `read`, `write`, and `lseek`~~
B - spin lock	~~I. represents a single component of a path~~
C - superblock	J. completes without interruption
I - dentry	K. interleaving may corrupt memory
D - inode	~~L. can be caused by interdependences of locks~~