

CS212 Final Review Section

Suresh Nambi

15 March 2024

Administrivia

- Time: Wednesday, March 20, 3:30pm-6:30pm
- Place: TBA on Edstem
- The exam is open-note you can bring any printed materials you want except for the textbook but you may not use any electronic devices during the exam.

- Suggestion: Printout of course slides is helpful
- For questions on Lecture material / Exams
 - Please post on Lectures+Exams thread on Edstem

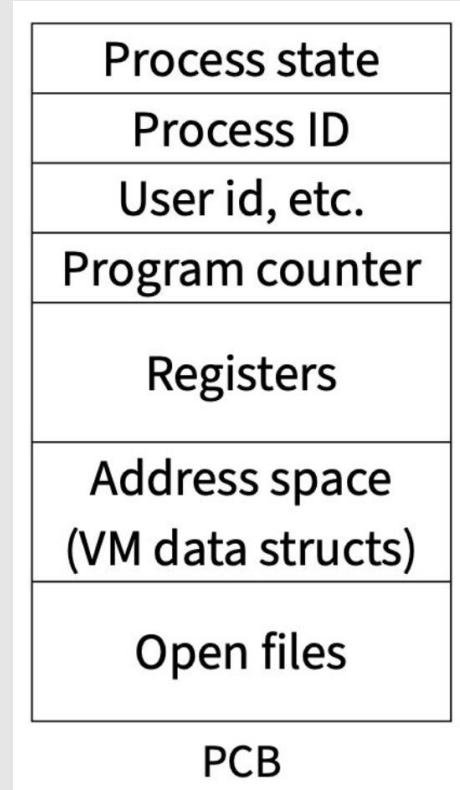
Agenda

- Recap of Concepts
 - Processes and Threads
 - Virtual Memory
 - Concurrency
 - Synchronization
 - Linking
 - Memory Allocation
 - Device I/O
 - File Systems
 - Networking
 - Security
 - Virtual Machines
- Solve Questions from Last Year

Quick Review of Midterm Topics

Process

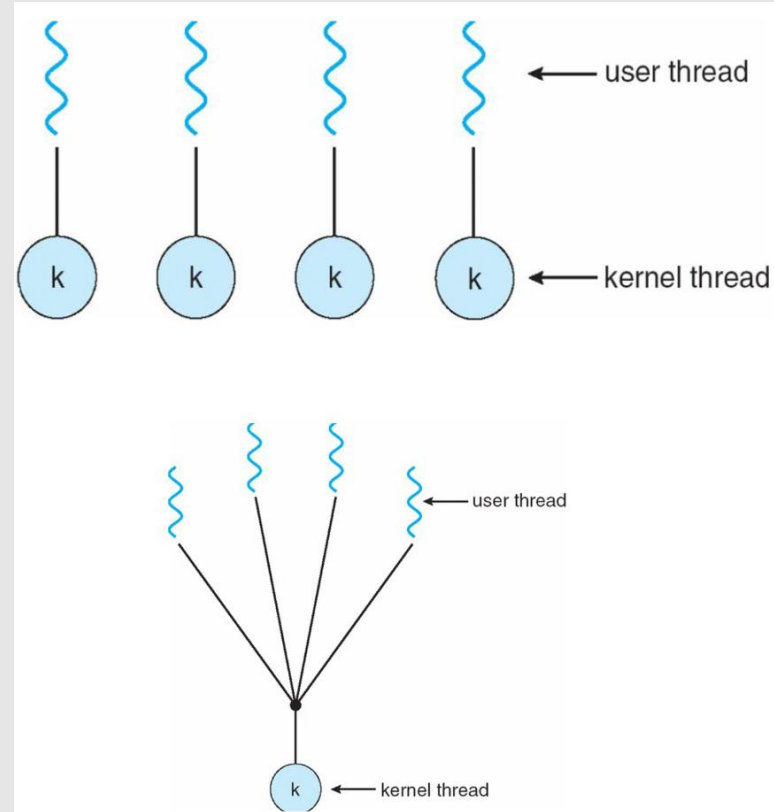
- Instance of a program running
- Why do we want to support multiple Processes ?
 - Increased CPU utilization
 - Reduced latency
- Process control block (PCB)
- Stores state, registers, open files, etc
- Equivalent: struct thread in pintos



Quick Review of Midterm Topics

Threads

- Schedulable execution context
- Why do we need threads ?
 - Concurrency
 - Multi-core execution
- Kernel threads
 - **More scheduling control**
 - **Heavy weight**
 - **Everything must go through kernel**
- User threads
 - **Lightweight and flexible**
 - **Less control**



Questions : ChatGPT vs CS212 Students

This section gives several interactions the instructor had with ChatGPT. For each interaction, you are asked to critique ChatGPT's answer or provide your own.

Q: When should a kernel disable interrupts?

A: A kernel may disable interrupts when it needs to ensure that a critical section of code is executed atomically, i.e., without being interrupted by other processes or interrupts. Disabling interrupts can prevent race conditions and other synchronization issues. Interrupts should be re-enabled as soon as the critical section has completed to allow the system to continue processing other events.

Is ChatGPT right? Select the best answer & Justify

A The answer is correct on all systems.

B The answer is correct only on single-core systems.

C The answer is correct unless you need inter-processor interrupts (IPIs) for TLB shootdowns.

D The answer is not correct.

Answer : ChatGPT vs CS212 Students

- Remember from class / notes
- Multicore systems need more than just interrupt disabling to implement synchronization primitives

- A The answer is correct on all systems.
- B The answer is correct only on single-core systems.
- C The answer is correct unless you need inter-processor interrupts (IPIs) for TLB shootdowns.
- D The answer is not correct.

Briefly justify your answer:

Answer:

On a multicore system, you also need to acquire a spinlock to prevent other CPUs from entering the critical section.

Quick Review of Midterm Topics

Virtual Memory : Mapping Memory

- Base + bound

- Physical address = virtual address + base

- Segmentation

- Divide memory into segments

- Demand Paging

- Divide memory into small, equal-sized pages
- Each process has its own page table
 - Multilevel
 - Translation Lookaside Buffer (TLB) caches recently used translations
- Any process can have a page
- What happens during a page fault?
- Eviction?
- LRU: Use past to predict future

Quick Review of Midterm Topics

Virtual Memory : Memory Considerations

- Fragmentation

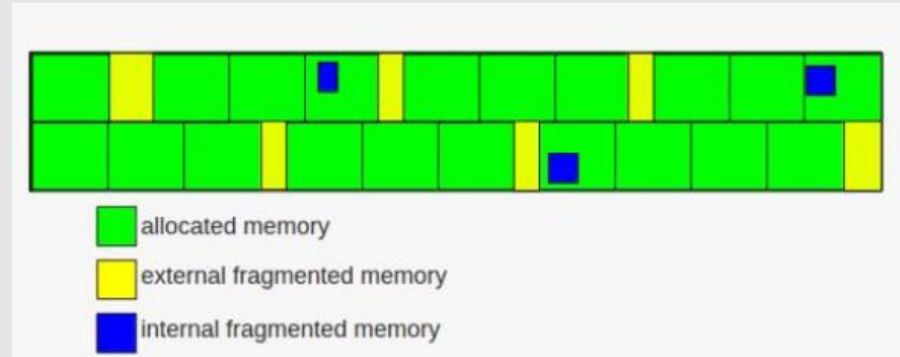
- Inability to use free memory
- External fragmentation
 - Many small holes between memory segments
- Internal fragmentation
 - Unused memory within allocated segments

- Speed

- Disk much slower than memory
- 80/20 rule
 - Hot 20 in memory = “working set”

- Local or global page allocation

- Thrashing



Quick Review of Midterm Topics

Virtual Memory : Memory System

- Coherence

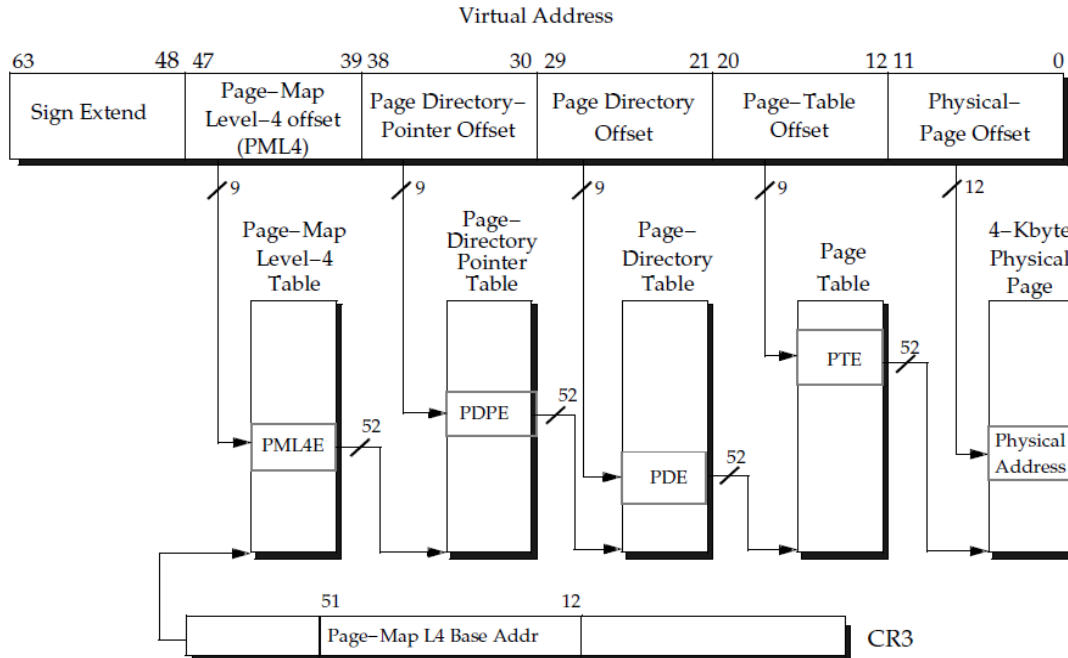
- Concerns access to single memory location
- Multiple processes writing to same variable

- Consistency

- Concerns ordering across multiple memory locations
 - If $x=1, y=2$, A reads x, y and B writes $x=3, y=4$, could A ever see $x=1, y=4$?
- Sequential consistency matches our intuition

Questions : ChatGPT vs CS212 Students

For the next two questions, it may be helpful to refer to the x86-64 paging structures we saw in the virtual memory hardware lecture. For convenience, we reproduce the picture here:



Questions : ChatGPT vs CS212 Students

Q: An application on x86-64 needs only two pages of virtual memory, at addresses 0x1000 and 0xffffffff0000. How many pages of physical memory are required to populate both the address space and all page mapping structures required by the architecture?

A: For x86-64 architecture, the page size is typically 4KB. So, to hold two pages of virtual memory, we need 8KB of virtual memory space.

To map virtual memory addresses to physical memory addresses, we need to use a page table. For x86-64, each page table entry is 8 bytes long, and a page table can contain up to 512 entries. Therefore, each page table can map up to 4MB of physical memory.

To map the virtual memory pages at addresses 0x1000 and 0xffffffff0000, we need two page table entries. Since each page table can contain up to 512 entries, we need two page tables.

Therefore, to populate both the address space and all page mapping structures required by the architecture, we need two page tables, which can map up to 8MB of physical memory. Since the page size is 4KB, we need $8\text{MB} / 4\text{KB} = 2048$ pages of physical memory.

What is your answer to the number of physical pages required to construct the two-page virtual address space?

Answer : ChatGPT vs CS212 Students

What is your answer to the number of physical pages required to construct the two-page virtual address space?

Number of pages: **9**_____

Justify your answer:

Answer:

The two pages of virtual memory are at opposite ends of the virtual address space, which means they cannot share any intermediary page mapping structures other than the address-space-wide PML4. So the physical page frames you need are: 1 PML4, 2 page-directory pointer tables, 2 page directories, 2 page tables, and finally two pages to hold the actual virtual memory.

Quick Review of Midterm Topics

Linker & Dynamic Linking

- Goal : Object Files -> Executable transformation

Pass 1

- Coalesce like segments
- Construct global symbol table
- Compute virtual address of each segment

- Pass 2

- Fix addresses of code and data using global symbol table

- Dynamic Linking

- Linked at runtime
- Helps with shared libraries
- May lead to runtime failure
- No type checking

Questions : ChatGPT vs CS212 Students

Q: What operations are slower in position-independent code (PIC) than in code that runs at a fixed virtual address?

A: Position-independent code (PIC) is a type of code that can be loaded and executed at any memory address, without relying on fixed virtual addresses. This is often used in shared libraries, where the library can be loaded at any address in memory, depending on the layout of the program's memory.

However, PIC code typically incurs a performance penalty compared to code that runs at a fixed virtual address. Here are some examples of operations that can be slower in PIC code:

1. Accessing variables: Since PIC code cannot rely on fixed virtual addresses, it needs to use indirect addressing modes to access variables. This requires an extra memory access to read the address of the variable from the Global Offset Table (GOT) or Procedure Linkage Table (PLT), which can increase the overhead of variable access.
2. Branch instructions: Branch instructions in PIC code also need to use indirect addressing, which requires an extra memory access to read the target address of the branch from the GOT or PLT.
3. Function calls: PIC code typically uses the PLT to call functions, which requires an extra memory access to read the address of the function from the PLT. This can increase the overhead of function calls.
4. Data cache performance: PIC code can have worse data cache performance than code that runs at a fixed virtual address. This is because the memory locations of the code and data are not fixed, which can lead to cache thrashing as different parts of the code and data are loaded into the cache.

Questions : ChatGPT vs CS212 Students

Which of ChatGPT's examples are correct (meaning they make sense and actually represent something slower in PIC than non-PIC code)?

Circle all that apply. (There may be multiple answers.)

- A (1) Access to variables
- B (2) Branch instructions
- C (3) Function calls
- D (4) Data cache performance

Justify your answer:

Answer : ChatGPT vs CS212 Students

OPTIONS A & C are correct

Answer:

In static code, access to a variable can be a single load or store instruction. In PIC, you must first access the variable's address from the GOT, then access the variable, requiring more instructions to be executed.

Branch instructions are typically PC-relative (meaning that they add or subtract some value from the program counter) rather than absolute (meaning they encode the target address). Hence, branches are already position-independent and don't incur any extra overhead in PIC.

Function calls that might be into other shared objects do need to indirect through the PLT, which will add overhead. This doesn't apply to static functions within the same shared object, since those can't be overridden by dynamic linking and use PC-relative call instructions without indirecting through the PLT.

ChatGPT's answer about the data cache doesn't make sense. Sure, the GOT might consume a few data cache lines and the text segment will be slightly dilated from extra instructions. Still, there's no reason that code should be loaded into the data cache. Lower layers of the cache hierarchy do have a unified instruction/data cache, so the extra memory for code and data could put additional pressure on the cache, but it's because the application needs slightly more memory, not because it can run at any address.

Review of Memory Allocation

Minimize fragmentation when allocated memory has

- different lifetimes
- different sizes

Understand implication of Allocation strategies

- (e.g. best fit, first fit)
- Tradeoffs/pathologies based on workload characteristics
- Memory Access Workload ramps, peaks, and plateaus

Recovering from Faults and Garbage Collection

- Introduce level of indirection (sub-page permissions, vm, concurrent snapshotting, mmap)
- For Garbage collection (stop & copy without stop gc)
 - mutator runs & collector collects, uses fault + resumption
- Familiarize how to work with reference counts (e.g. ownership in Rust)

Review of Device drivers & communication

How to communicate with a Device

- Memory-mapped device registers
 - Regular read/write interface except access device's registers directly
- Memory-mapped device memory
 - Regular read/write interface except access device's internal memory
- Special instructions (e.g. inb, outb)
 - Communicates using port numbers
- DMA (direct memory access)
 - CPU offloads read/write of main memory to device/DMA Engine

Device Driver operation modes

- Polling
 - Loop until some condition X is true
- Interrupt Driven Devices
 - Ask card to Interrupt on CPU events

Review of File Systems

Understanding Disk Properties

- Sector is the unit of atomicity
- Sequential I/O is much faster than random
- Long seeks much slower than short ones
- Disk Scheduling Strategies
 - First Come First Serve
 - Shortest Positioning Time First
 - Elevator Scheduling or SCAN

Understanding Flash Memory

- Completely solid state
- Limited Durability and Limited number of overwrite support
 - Need for Flash Translation Layer

Review of File Systems

Important Concepts

- In Unix “Everything is a file”
 - Directories are files with a special format
- Hard Link vs Symbolic Link (Synonyms for name)
- Speeding up the file system (Fragments, Cylinder Clustering & Free map)

Handling Crashes & Minimizing corruption

- Ordered updates
 - to ensure fsck works (e.g. write new inode to disk before directory entry)
- Soft updates
 - update order may create cycles
 - break cycles by temporarily rolling back all changes that created the cycle
- Journaling
 - allow operations to act as though they are atomic
 - use a write-ahead log, then replay the log on crash

Questions : ChatGPT vs CS212 Students

Q: A work-conserving disk scheduler always keeps the disk busy when there are disk requests. Are there scenarios in which a non-work-conserving disk scheduler (that may keep the disk idle even with pending requests) can achieve higher throughput in requests/second than a work-conserving scheduler?

A: Yes, there are scenarios in which a non-work-conserving disk scheduler can achieve higher throughput than a work-conserving scheduler.

A work-conserving disk scheduler always serves a request as soon as it is available, without any delay. This may lead to increased seek times and rotational latency if the requests are not arranged in a logical order. For example, if a disk is serving requests for two different files located on opposite ends of the disk, then the disk head may need to travel back and forth between the two locations, resulting in increased seek times and rotational latency. This can reduce the overall throughput of the disk, as it spends more time seeking and less time serving requests.

On the other hand, a non-work-conserving disk scheduler can arrange the requests in a logical order to minimize the seek times and rotational latency. For example, it may delay a request temporarily to allow other requests that are closer to the current location of the disk head to be served first. This can reduce the overall seek times and rotational latency, and hence increase the overall throughput of the disk.

However, it should be noted that a non-work-conserving disk scheduler may not be suitable for all scenarios. In some cases, such as real-time systems where low latency is critical, a work-conserving scheduler may be preferred, even if it results in lower overall throughput.

Answer : ChatGPT vs CS212 Students

Is ChatGPT correct that a non-work-conserving disk scheduler can in some scenarios achieve higher throughput than a work-conserving disk scheduler?

Circle the best answer.

A Yes, ChatGPT's high-level answer is correct.

B No, ChatGPT is incorrect.

Justify your answer:

Answer:

Surprisingly, ChatGPT got this one right. In fact, a 2001 research paper by Sitaram Iyer and Peter Druschel called Anticipatory scheduling: a disk scheduling framework to overcome deceptive idleness in synchronous I/O quantitatively evaluates the benefit of leaving the disk idle when the kernel anticipates another request near the current location of the disk head.

Questions : ChatGPT vs CS212 Students

Q: You have a journaling file system containing a directory A/D and another directory B. You attempt to move D into B by invoking the system call `rename("A/D", "B/")`, but the system loses power while the kernel was still writing changes to disk. What are the possible states of the file system after a reboot and `fsck`?

A: If the system loses power while the kernel was still writing changes to disk during a `rename()` operation on a journaling file system, there is a risk of data loss or file system corruption. However, because a journaling file system maintains a journal of file system changes, the risk is mitigated compared to file systems without a journal.

After a power loss during a `rename()` operation on a journaling file system, the file system may be in one of the following states after a reboot and `fsck`:

1. The file system may be consistent and fully recovered if the journal contained all of the changes made during the `rename()` operation. In this case, the journal replay process during boot will complete without errors, and the file system will be fully consistent and operational.
2. The file system may be partially inconsistent or corrupted if the journal did not contain all of the changes made during the `rename()` operation before the power loss. In this case, the journal replay process during boot may fail, and the file system may be marked as "dirty" or "unmounted". An `fsck` may be required to recover the file system, which may result in data loss or corruption.
3. The file system may be completely corrupted or damaged beyond recovery if the power loss occurred during a critical operation, such as journal initialization or file system creation. In this case, the file system may be unable to mount or recover, and data loss is likely.

In general, a journaling file system is designed to minimize the risk of data loss or corruption in the event of a power loss during a file system operation. However, it is still important to perform regular backups and to ensure that the file system is properly configured

Answer : ChatGPT vs CS212 Students

Which of ChatGPT's possible outcomes are correct (meaning they make sense and could result from the scenario in question)?

Circle all that apply. (There may be multiple answers.)

- (1) The rename is completed on recovery.
- (2) The journal entry is corrupted, causing file system recovery to fail and requiring manual intervention or a full fsck.
- (3) The file system may be corrupted beyond recovery.

Justify your answer:

Answer:

(1) If the journal entry has been written correctly, then even if the crash happens while the actual directory contents are inconsistent, the journal entry will be replayed, completing the rename and leaving the file system with the directory in B/D.

(2) A journaling file system uses a write-ahead log, meaning a complete log entry must be written to the journal before any affected file system structures can be changed in-place on disk. Hence, if the journal entry at the end of the log is corrupted, the file system can simply ignore it because the file system is still consistent after replaying previous records. In this case, the file system will just leave the directory at A/D.

(3) makes even less sense. The premise of the question is that you already have the file system with this directory structure, so you can't still be in the middle of creating the file system or initializing the journal.

Review of Networking and NFS

Allow two applications on different machines to Communicate

- OS provides the abstraction for communication
 - Handles packaging, sending, unpacking, and delivering of information

Networking Implementation

- mbuf used to store packet data
 - Packets made up of multiple mbufs
 - mbufs are basically linked-lists of small buffers
- protosw structure as abstract network protocol interface
 - Goal: abstract away differences between protocols
 - In C++, might use virtual functions on a generic socket struct
 - Here just put function pointers in protosw structure

Networking File System

- File system where data is potentially stored on other machines (vnode – virtualize file system)

Review of Security

Something to think about :

Suppose we want to write a program than need to allows users to change their password but not use Setuid how can this be done ? (Background Daemon)

Setuid (continued)

- **Examples**
 - passwd – changes user’s password
 - su – acquire new user ID (given correct password)
 - sudo – run one command as root
 - ping (historically) – uses raw IP sockets to send/receive ICMP
- **Have to be very careful when writing setuid code**
 - Attackers can run setuid programs any time (no need to wait for root to run a vulnerable job)
 - Attacker controls many aspects of program’s environment

Questions : ChatGPT vs CS212 Students

Q: Give an example of how to use the `access()` system call in a `setuid` Unix program to ensure that you are writing a file that the invoking user had permission to write.

A: The `access()` system call in Unix can be used to check if a user has permission to perform a specific operation on a file or directory. In a `setuid` program, the invoking user is typically the user who executed the program, not the user who owns the file being accessed. To ensure that the program is writing to a file that the invoking user has permission to write, you can use the following steps:

1. Get the user ID of the invoking user by calling `getuid()`.
2. Call `access()` with the path to the file and the `W_OK` flag to check if the user has write permission to the file. If the function returns a non-zero value, then the user has permission to write to the file. If it returns a zero value, then the user does not have permission to write to the file.
3. If the user has permission to write to the file, then proceed with the file write operation. If the user does not have permission, then exit the program or return an error code.

Here's an example C code snippet that demonstrates how to use the `access()` system call to check if the invoking user has permission to write to a file:

Questions : ChatGPT vs CS212 Students

```
int main(int argc, char *argv[]) {  
    int uid = getuid(); // Get the user ID of the invoking user  
  
    char *filename = "example.txt";  
  
    if (access(filename, W_OK) != 0) {  
        fprintf(stderr, "User %d does not have write permission to file %s\n",  
            uid, filename);  
  
        exit(EXIT_FAILURE);  
    }  
}
```

```
// The user has permission to write to the file, so open it  
and write to it  
int fd = open(filename, O_WRONLY | O_CREAT, 0666);  
if (fd == -1) {  
    perror("open");  
    exit(EXIT_FAILURE);  
}  
// Write some data to the file  
char *data = "This is an example file.";  
if (write(fd, data, sizeof(data)) == -1) {  
    perror("write");  
    exit(EXIT_FAILURE);  
}  
close(fd);  
return 0;  
}
```

Answer : ChatGPT vs CS212 Students

Is ChatGPT's code secure?

Circle the best answer.

- A Yes, ChatGPT's code is correct and prevents invoking users from writing files to which they do not have access.
- B No, ChatGPT's code is insecure.

Justify your answer:

Answer:

This is a classic time-of-check-to-time-of-user (TOCTTOU) bug, in which the invoking user can replace `example.txt` with a symbolic link between the call to access and the call to open. There is no safe way to use the access system call for permission checks.

By the way, ChatGPT describes the access system call wrong in English in step 2—access returns 0 when the user has access and -1 when the user doesn't—but the code checks the return value with the right polarity.

Also, the gratuitous use of `O_CREAT` when opening a file that's already supposed to exist (given the `access()` check) means that the `setuid` program will create a new file (which will be owned by the effective user ID, not the user ID of the invoking user) in the event that the file has been deleted after the call to `access()`. That doesn't make the already vulnerable program any easier to exploit, but it's not good behavior either.

Questions : ChatGPT vs CS212 Students

Q: Suppose you are using nested paging to run a 64-bit virtual machine on an x86-64 CPU. What is the worst-case number of additional memory accesses required for a load instruction that misses in the TLB?

GPT Answer: In a system that uses nested paging, the CPU must perform additional memory accesses to traverse the page table hierarchy, which can cause additional latency compared to a system that uses a flat page table.

For a load instruction that misses in the TLB in a 64-bit virtual machine on an x86-64 CPU using nested paging, the worst-case number of additional memory accesses required would be 4:

1. First, the CPU would need to perform a memory access to load the root page table base address from the CR3 register.
2. Next, the CPU would need to perform a memory access to load the page directory pointer table entry corresponding to the virtual address's top-level page directory pointer table index.
3. The CPU would then need to perform a memory access to load the page directory entry corresponding to the virtual address's second-level page directory index.
4. Finally, the CPU would need to perform a memory access to load the page table entry corresponding to the virtual address's third-level page table index, which contains the physical address that the load instruction is trying to access.

Note that in some cases, the number of additional memory accesses required could be fewer than 4, depending on the specifics of the page table hierarchy and the location of the page table entries in memory. However, 4 is the worst-case scenario for a load instruction that misses in the TLB.

Questions : ChatGPT vs CS212 Students

Assume that the virtualization hardware's guest-physical-to-host-physical page mapping uses 4 levels just like regular virtual-to-physical page mapping (consulting a PML4E, PDPE, PDE, then PTE).

What do you think the worst-case number of extra memory accesses is for a TLB miss (not counting the one memory access actually performed by your load instruction, but counting all other memory accesses)?

Number of extra memory accesses: _____

Justify your answer:

Answer : ChatGPT vs CS212 Students

Answer:

Without virtualization, a TLB miss in a physical machine can require 4 extra memory accesses, one for each of the 4 levels of the paging hierarchy.

With nested paging, the address of each of these intermediary paging data structures will be a guest physical address that must also be translated to a host physical address. Hence:

- *Translating the guest physical of the PML4 in the guest's %cr3 to its host physical address could require 4 memory accesses.*
- *Reading the PML4E takes 1 memory access.*
- *Given a PML4E (which contains the guest physical address of a PDPT), it takes 4 layers of translation to find the host address of the PDPT, but at a minimum the top level translation will be cached from translating the PML4E, so the worst case number of accesses is 3.*
- *Accessing the PDPE costs 1 memory access.*
- *Given a PDPE, it will take 3 accesses to learn the host address of the page directory, plus 1 to read the PDE, for a total of 4 accesses.*
- *Given a PDE, it will similarly take 4 accesses to obtain the PTE.*
- *Finally, the PTE contains a guest physical address that must be translated, which could require 3 more accesses.*

Hence, the worst-case total number of extra accesses is 20. Of course, in practice most virtual machines have limited guest physical memory, so likely require only a single PDPT and a small number of PDEs in the guest-to-host mapping structure (also known as the second-level address translation "SLAT" mapping).

Backup Slides with Additional Questions



Thank You