Operating Systems 2016/17
Tutorial-Assignment 14

Prof. Dr. Frank Bellosa
Dipl.-Inform. Marc Rittinghaus

Question 14.1: File System Cache

a. As stated in the lecture, disks cannot access individual bytes but only blocks of data. If you write byte-granular data to a file, is it reasonable to assume that the data is physically on the disk after you have executed the write system call?

Solution:
If the block was flushed to disk every time a byte is written, the write performance would be abysmal. Thus it is likely that the actual disk contents are the same before and after the write system call as the writes are cached in main memory.

b. What is the basic idea of a file system cache in main memory?

Solution:
Disk blocks that have been referenced in the near past are likely to be referenced again in the near future. So, instead of accessing the disk multiple times querying the same block, each block is only fetched once and kept in memory until it is likely not to be used anytime soon or the file system cache becomes full. Explicit calls (`flush`, `sync`) or a daemon (`flushd`) may write modified blocks back to disk.

c. Discuss the pros and cons of a fixed vs. variable limit between the file system cache and the main memory area used for paging.

Solution:

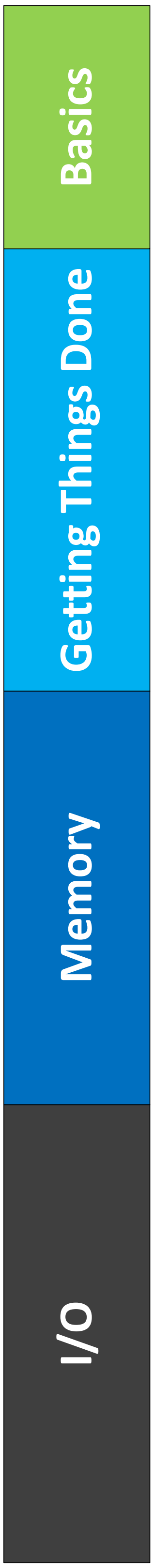
+ *Fixed easy to implement*
+ *Fixed guarantees possible (important for real-time systems)*
– *Fixed hardly adaptable to different workloads (much file I/O vs. high memory use/swapping)*

d. Some systems offer read-ahead. Discuss pros and cons.

Solution:

+ *Good performance on sequential/predictable file access patterns*
+ *Improved disk throughput if files reside in successive disk blocks*
– *Wasted bandwidth if read-ahead data is not used*

Recap



- C Basics
- Operating System Basics
  - User-/Kernel Boundary
  - System Calls
  - Interrupts and Exceptions
- Processes and Threads
  - Anatomy of a Program and Program Loading
  - Creation, Forking and Termination
  - Thread Models
  - Dispatching
  - Scheduling
- Interprocess Communication (IPC)
- Synchronization
- Memory Management
  - Basics (Addresses and Address Spaces, AS Layout, . . .)
  - Hardware (MMU, TLB, Caches, . . .)
  - Segmentation vs. Paging
  - Page Tables
  - Page Fault Handling
  - Frame Allocation and Replacement
  - Allocators
  - Shared Memory (CoW, . . .)
- Storage
  - Raid
  - HDD vs. SSD
- File Systems
  - Basics (Files and Directories, Access and Access Control, Locking, . . .)
  - Disk Space Allocation
  - File System Implementation
  - Virtual File System (VFS)
  - File System Cache

The following questions do NOT cover all material from the lecture, but merely serve as a small reminder of some of the things discussed. Be sure that you not only can enumerate things as done in most answers below, but also explain them. The abbreviations help you locate where we have talked about the respective topics (T = Tutorial, A = Assignment, L = Lecture).

Question 14.2: Operating System Basics

T1, A1, L2

a. What are the major tasks of an operating system?

Solution:

- *Abstraction/Standardization*
- *Resource Management*
- *Security/Protection*
- *Execution Environment for Applications*

b. What are privileged instructions? Give some examples.

Solution:
Instructions that control and manage the system as a whole and should only be performed by the operating system.

- *Enable/disable interrupts*
- *Set current address space (set CR3)*
- *Halt the processor . . .*

c. When is the kernel of the operating system invoked?

Solution:

- *Interrupts*
- *Exceptions*
- *System Calls*

d. How does a system call work?

Solution:

- *On startup, OS configures trap instruction*
- *Program puts arguments on stack/registers as specified by the OS-ABI*
- *Program puts system call number, which identifies the requested service, on stack/register*
- *Program executes trap instruction*
- *CPU switches to privileged mode and jumps to system service dispatcher (kernel routine defined by OS on startup)*
- *Dispatcher copies arguments on kernel stack, performs various checks*
- *Find service routine for system call number and calls it*
- *System service routine checks arguments and performs service*
- *Return to dispatcher and call to system exit instruction*
- *Program continues in user-space*

e. Why is it more secure to use kernel stacks, when in kernel mode?

Solution:

- *Programs can read/manipulate user stack. If user stack would be used in kernel, user space code could lead kernel to execute arbitrary code (modify return address, modify arguments etc.)*
- *Information leakage*

Question 14.3: Processes and Threads

T2, T3, T4, T5, A2, A3, A4, A5, L3, L4, L5, L6

a. What is the difference between a program and a process?

Solution:
The program can be compared to the recipe and the process of the actual cooking using the recipe. The program just contains the instructions and (some of) the data used to perform actions. The process executes a program.

b. What are typical segments in a program?

Solution:

- *Code*
- *Read-only Data*
- *Data*
- *BSS (Block Started by Symbol)*
- *(Stack and heap are not part of the program, but they are part of the final address space layout of the running process)*

c. What is the difference between processes and threads?

Solution:
Processes are containers for the execution of a program and hold many of the necessary resources (address space, handle tables, . . .). Threads actually execute the code. Depending on the context a process is often also considered to contain at least one implicit thread (thus the phrase 'Process X does Y').

d. What types of threads do you know?

Solution:

- *Kernel-level Threads*
- *Kernel-mode Threads*
- *User-level Threads*

e. What does the fork-system call?

Solution:
Creates a copy (child) of a process (parent) that shares most of the resources with its parent (e.g., address space, open file table, . . .). However forking a process does only copy the thread that called fork(). All other threads are not copied.

f. What threading models do you know?

Solution:
- *One-to-One*
- *Many-to-One*
- *Many-to-Many (Hybrid)*

g. What is a context switch? How does it work?

Solution:
Switch between two processes (assuming one implicit thread). Save process context and restore context of next process. Context contains CPU registers, address space configuration (e.g., address of page directory). Registers can be exchanged by writing them on stack, changing stack pointer, and then reading registers (from next process) back in from the newly set stack (in reverse order).

h. In what states can a thread be?

Solution:
- *Running*
- *Ready*
- *Blocked*
- *(. . . more if needed by the OS)*

i. What is difference between a short-term scheduler and a long-term scheduler?

Solution:
Short-term selects which process should be executed next and allocates CPU. Long-term selects which processes should be brought into the ready queue (controls degree of multiprogramming).

j. What scheduling goals do you know?

Solution:
- *Throughput*
- *Turnaround Time*
- *CPU Utilization*
- *Waiting Time*
- *Response Time*

k. What scheduling polices did we discuss in the lecture?

Solution:
- *First-Come, First-Served (FCFS)*
- *Shortest-Job-First (SJF)*
- *Preemptive Shortest-Job-First (PSJF)*
- *Round-Robin Scheduling*
- *Priority Scheduling*
- *Multi-Level Feedback Queue (MLFB)*
- *Lottery Scheduling*

Question 14.4: Interprocess Communication (IPC)

T6, A6, L7

a. What techniques for interprocess communication (IPC) do you know?

Solution:

- *Shared Memory*
- *Message Passing (Pipes, Message Queues)*

b. What sender/receiver synchronization can you have?

Solution:

- *Blocking (Synchronous)*
- *Non-Blocking (Asynchronous)*

c. When do you need to buffer messages? Where can you do this?

Solution:
You need to buffer messages when using asynchronous communication. You may store messages in:

- *Sender Process User Memory*
- *Receiver Process User Memory*
- *Kernel Memory*

Question 14.5: Synchronization

T7, T8, A7, A8, L7, L8

a. What is a race condition?

Solution:
A race condition is a situation where multiple threads execute the same operations (e.g., incrementing a variable) and the outcome of the operations (e.g., the final value of the variable) depends on the scheduling of the threads. Race conditions usually lead to hard to find bugs, because their can be difficult to reproduce.

b. When is disabling interrupts a valid solution to avoid race conditions?

Solution:
When in kernel-mode on a single-core system.

c. What are the requirements for a valid synchronization?

Solution:

- *Exclusiveness*
- *Progress*
- *Bounded Waiting*

d. What synchronization primitives do you know?

Solution:

- *Spinlock*
- *Counting Semaphore*
- *Binary Semaphore / Mutex*
- *Condition Variable*
- *Reader-Writer Lock*
- *Futex*

e. What hardware feature is required to implement synchronization primitives?

Solution:
The CPU must provide means to perform certain operations atomically. For a spinlock for example we need an atomic compare-and-swap.

f. What is a deadlock? How can it be prevented?

Solution:
A deadlock is a situation where threads working on the same resources do not make any progress because all of the following conditions hold:

- *Mutual Exclusion*
- *Hold and Wait*
- *No Preemption*
- *Circular Wait*

Prevention breaks at least one condition.

Question 14.6: Memory Management

T9, T10, T11, T12, A9, A10, A11, L3, L9, L10, L11, L12, L13, L14

a. What is a memory management unit (MMU)?

Solution:
Hardware support for virtual to physical address translation and memory protection

b. What is a translation lookaside buffer (TLB)?

Solution:
Address translation cache to reduce memory accesses due to address translation. On a TLB miss, the MMU tries to resolve the situation by walking the memory mapping structure (hardware-managed TLB) or directly invokes the OS (software-managed TLB).

c. What major virtual to physical mapping techniques did we discuss?

Solution:

- *Base and Limit Register*
- *Segmentation*
- *Paging*

d. What identifies an address space with segmentation?

Solution:
The segment table, which is specified with:

- *Segment-table base register (STBR)*
- *Segment-table length register (STLR)*

e. What information is stored in a segment table?

Solution:
Each row describes a (valid) memory region.

- *Base address*
- *Length*
- *Protection*

f. What is a page? What is a frame?

Solution:
Unit of translation for paging-based memory management (typically 4 KiB in size). Virtual address space is build from pages. Pages in the physical address space (RAM) are called frames. They must have the same size.

g. What identifies an address space with paging?

Solution:
The page table, which is specified via its physical address in RAM (CR3 register on x86).

h. What types of page tables did we discuss?

Solution:
- *Linear Page Table*
- *Multi-Level Page Table*
- *Linear Inverted Page Table*
- *Hashed Inverted Page Table*

i. What information is typically stored in a page table entry (PTE)?

Solution:
- *Valid/Present Bit*
- *Page Frame Number*
- *Read/Write Bit*
- *User/Kernel Bit*
- *Caching Bit*
- *Accessed Bit*
- *Dirty Bit*

j. What trade-offs exist between small and large pages?

Solution:
Large pages:

- *More memory wasted due to internal fragmentation*
- *Fewer bits needed for page/frame number (more bits in the offset)*
- *Fewer page tables / fewer PTEs*
- *More data needs to be loaded from disk to make page valid*

k. What is a page fault?

Solution:
The MMU cannot translate a virtual to a physical address (page not present) or detected an illegal access (protection fault such as write on read-only page) and invokes the operating system to handle the situation.

l. What steps are involved in handling a page fault?

Solution:

- *OS checks validity of access (requires information of address space layout and allocated memory regions of the process)*
- *Get empty frame (potentially perform page replacement)*
- *Initialize frame (load contents as needed)*
- *Adapt page table (incl. setting valid bit*
- *Restart instruction that caused the fault*

m. What is copy-on-write?

Solution:
Mechanism to transparently share pages and save memory as long as no modification is made to the respective pages. Pages in all address spaces marked read-only. On write access, a page fault is triggered (protection fault). OS allocates new frame, copies page, updates mapping of respective process and restarts write operation (optionally allows RW-access to original page for other process if it is the last to reference the page).

n. What page fetch policies do you know?

Solution:

- *Demand-Paging*
- *Pre-Paging*

o. What page replacement policies do you know?

Solution:

- *First-In-First-Out (FIFO)*
- *Optimal Page Replacement*
- *Least Recently Used (LRU)*
- *Clock Page Replacement*
- *Random*

p. What is the working set of a process?

Solution:
The set of pages referenced in the most recent time-delta.

q. What memory allocation policies do you know?

Solution:

- *First-Fit*
- *Best-Fit*
- *Worst-Fit*

r. What more advanced memory allocators did we discuss?

Solution:

- *Buddy Allocator*
- *Slab Allocator (Slab Cache)*

s. What is internal fragmentation? What is external fragmentation?

Solution:
Fragmentation is the inability to use free memory. Internal fragmentation occurs when we have a fixed allocation granularity (e.g., 4 KiB, 2^x, . . .) and require less memory. The remaining space up to the allocation granularity is wasted. External fragmentation occurs if the allocator can provide memory blocks of almost any size. Due to the different lifetimes of these tailored memory blocks, the memory as a whole may be scattered: It may contain a lot of memory holes, each of them being too small for an upcoming memory request, although the total of free memory would be sufficient.

t. Can we always use compaction to reduce external fragmentation?

Solution:
No. Only if we have an extra layer of indirection between the addresses that we gave away and the actual position of allocated memory blocks. Otherwise, previously returned addresses are no longer valid after compacting memory blocks. Example: We cannot use compaction for simple C programs and the standard heap (malloc), but we can use compaction to move around frames in physical memory (if they are only referenced through their virtual address).

Question 14.7: Storage

T12, A12, L15

a. How are hard disks addressed today?

Solution:
Via a sector number (LBA), that is internally translated to a physical sector by the hard disk (CHS).

b. What special performance characteristic do SSDs possess?

Solution:
Rewriting data is much slower than writing it. You cannot simply write a 1 or 0 into a NAND memory. A single bit can only be set to 0. Setting a bit to 1 (erasing it) can only be

performed on large blocks (64-128 pages) due to limitations in the way NAND memory can be accessed. Rewriting thus requires: a) reading an entire block b) erasing it c) writing the modified pages as well as all unmodified pages in that block back.

c. What two approaches help in this matter?

Solution:

- *Spare Blocks*
- *Trim Command*

Question 14.8: File Systems

T12, T13, T14, A12, A13, L16, L17

a. What is a file descriptor?

Solution:
Simple integer, representing an instance of an open file. Multiple file descriptors may exist for the same file (after multiple calls to read()).

b. What data structures are used to handle open files?

Solution:

- *Local Open File Table (translates file descriptors to open file state objects)*
- *Global Open File Table (contains open file state objects)*
- *Inodes (represents file on disk)*

c. What access control mechanisms did we discuss?

Solution:

- *UNIX access rights*
- *Access control lists (ACLs)*

d. What methods of disk block allocation did we discuss?

Solution:

- *Contiguous Allocation*
- *Chained Allocation*
- *Linked-List Allocation and File Allocation Table*
- *Indexed Allocation*

e. What data is stored is stored in an inode?

Solution:

- *File size*
- *Abstract File type (Symbolic link, directory, device, plain file, . . .)*
- *Access rights*
- *Timestamps*
- *Ordered list of data blocks numbers*

f. How can directories be implemented?

   Solution:
   Simple file, containing entries for each file in the directory. An entry contains at least the name of file and a link to the inode. Entries may be organized in a simple list or in more complex data structures to improve search speed (e.g., binary trees). File names have variable length. May be stored inline (as part of the entry) or in a heap fashion, where each entry has a fixed size and contains an pointer to the name on the name heap.

g. What are pros and cons of a virtual file system (VFS)?

   Solution:
   VFS provides unified interface for different file systems. The same tools can be used without taking care of the type of file system. A drawback is that special FS features (e.g., data deduplication, snapshots, encryption, etc.) are only accessible through narrow interface (ioctl - I/O Control).