CS2106 Introduction to Operating Systems

AY24/25 Sem 2, github.com/keithxun

Threads

- Less resources compared to processes
- User thread: Thread management done by user-level library, no kernel support, cannot exploit multiple CPUs, one thread will block other threads
- Kernel thread: Thread management done by kernel, kernel support, multiple thread from same process can run simultaneously, more resource intensive and less flexible
- Hybrid: User thread can bind to a kernel thread

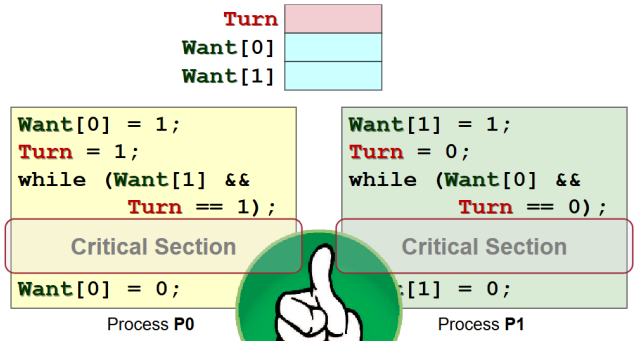
Inter Process Communication

- Shared memory: Fast and easy to use but synchronization is difficult, need to use semaphores or mutex
- Message passing: Messages stored in kernel memory space, can be direct (name the other party) or indirect(sends to mailbpx/port), can be blocking or non-blocking, can be synchronous or asynchronous, portable, easy to synchronize but inefficient
- Unix Pipes: Function as circular byte buffer, writers wait when buffer is full etc, pipefd[0] for reading, pipefd[1] for writing, int pipe(int pipefd[2]) to create

Synchronization

Mutual exclusion (prevent other process), progress (allow process if none in use), bounded waiting (after request to enter, upperbound of waiting time exists), independence (process not executing should never block others)

- Peterson's
  algorithm: 2 processes, 1 flag for each process, 1 turn variable, busy waiting



- Repeatedly test while-loop instead of going into blocked state
- Semaphore: wait decrement and block if 0, signal increment and wake up waiting process, deadlock possible if used incorrectly (1: Wait p, wait q, 2: wait q, wait p)
- Dining philosophers: (Tanenbaum) Only eat when both neightbour aren't eating, when finished, put down chopsticks and signal neighbours. (Limited eater) Footman semaphore initialized to N-1 so at least one philosopher is always prevented from eating

Memory Management

Memory Abstraction - Contigious Memory Allocation

- Fixed partitioning: Internal fragmentation (small process wastes space), size need to be large enough to contain largest process
- Dynamic partitioning: External fragmentation (many holes), need to maintain more information in OS, first fit, best fit, worst fit, buddy system (s bit of B and C is a complement)

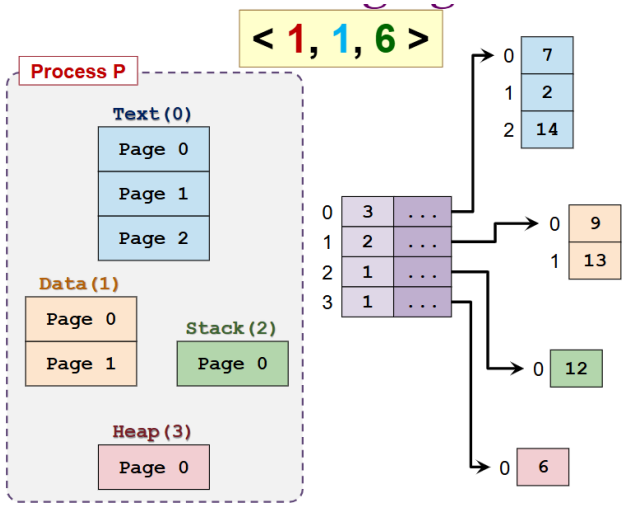
Paging

- Logical address space divided into pages, physical memory divided into frames
- Page table maps logical page to physical frame
- Keep frame size as power of 2
- Physical frame size = logicial page size
- PA = (frame number) * (frame size) + offset
- Internal fragmentation (last page not full)
- Page table information stored in PCB
- 2 Memory accesses for each logical address
- TLB: Cache for page table, TLB hit - retrieve frame number, TLB miss - page table lookup, update TLB
- TLB is part of hardware context switch
- Protection: Access-bits, valid bit
- Page sharing: Copy-on-write, shared pages

Segmentation

- Logical address space divided into segments, each segment has a name, base and limit
- LA = ¡segment number, offset¿
- PA = (base from segment table) + (offset)
- Can grow/shrink and be protected/shared independently
- Can cause external fragmentation

Segmentation with Paging



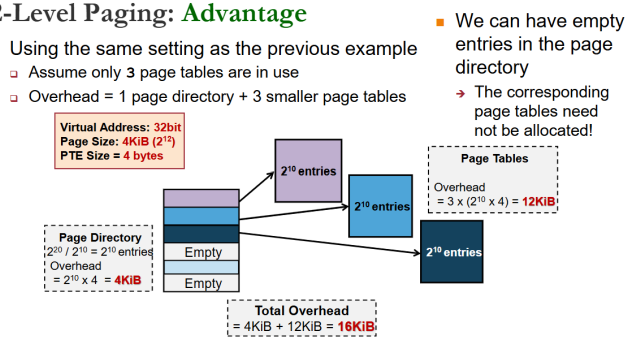
- Segment table contains page table
- Page table contains frame number
- LA = ¡segment number, page number, offset¿
- PA = (frame number from page table) + (offset)

Virtual Memory

- Secondary storage ¿¿ RAM
- Memory resident bit in page table
- Page fault: page not in memory, OS handles page fault, restart
- Temporal locality: recently used pages likely to be used again
- Spatial locality: pages near recently used pages likely to be used
- Demand paging: Starts with no memory resident page, only loads a page when page fault
- Fast startup time but appear sluggish at the start

Page Table structures

- Direct paging: All entries in a single table, page table size = 2^n * (size of page table entry)
- 2-level paging: 2-level page table, first level page table points to second level page table, second level page table points to frame
- Only allocate page table entries for pages that are in memory
- Advantage: Less memory usage, no need to allocate page table for all pages



- Inverted page table: Maps physical frames to ⟨pid, page number⟩, only one page table for all processes, one table for all processes but need to search for page table entry, pid + page number is unique but not page number only

Page Replacement Algorithms

- Dirty page: modified -¿ need to write back to disk
- $T_{access} = (1-p) \cdot T_{mem} + p \cdot (T_{page\_fault})$
- FIFO: First in first out, replace oldest page, time is not updated when referenced again, more frames lead to more page faults (Belady's anomaly), does not exploit temporal locality
- LRU: Least recently used, replace page not used for longest time, does not suffer from Belady's anomaly, need to maintain a counter/stack (able to remove entries anywhere) to track "last access time", expensive to implement, strictly increasing can lead to overflow for counter
- Second chance: FIFO with reference bit, if reference bit is 1, give second chance, if 0, replace page
- Optimal: Replace page not used for longest time in future

Frame Allocation

- Global replacement: All frames are shared, allow self-adjustment between processes but badly behaved process can take all frames, thrashing limited to one process but it can hog the I/O and degrade performance of other processes
- Local replacement: Each process has fixed number of frames, no self-adjustment, can lead to underutilization of memory, thrashing process "steals" oage from other processes leading to cascading thrashing
- Working set model: W(t, Δ) = set of pages used in the last Δ time units, Δ = working set window, t = time of reference

File Management

Citeria: Self-contained, persistent, efficient

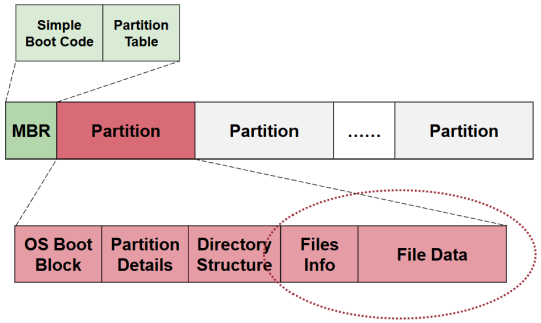
File System

- Store data + metadata
- ASCII files: Can be displayed or printed, e.g. text file, source code
- Binary files: Have a predefined internal structure that can be processed by a specific program, e.g. executable, Java class, pdf, png, mp3
- File operations as system calls: Create, open, close, read, write, delete, seek
- Per-process open file table (in pcb) -¿ system-wide open file table -¿ disk

Directory Structure

- Single-level: All files in one directory, easy to implement, no hierarchy, no grouping
- Tree-structure: Hierarchical, easy to implement, easy to navigate, easy to group
- DAG: One copy of actual content, hard link (files only, deletion problems), symbolic link (directory or file, special link file containing path name of file, no deletion problems but larger overhead)
- Symbolic link in unix is allowed to link to directory creating a general graph

Disk organization



- Must keep track of the logical blocks, allow efficient access, utilize disk space effectively
- Contiguous allocation: Simple, fast, no fragmentation, but need to know size of file in advance, external fragmentation (name, start, length)
- Linked allocation: No external fragmentation, but random access is very slow, part of disk block is used for pointer, less reliable (name, start, end)
- Linked list 2.0 (FAT): FAT is in memory at all times, simple yet efficient, FAT keeps track of the links between blocks, fast random access as traversal is done in memory but FAT can be huge and consumes valuable memory
- Indexed allocation: Each file has an index block (block[N] == N block address), lesser memory overhead, fast direct access but limited maximum file size and index block overhead (name, index block)

Free Space Management

- Located in partition details section
- Bitmap: Each bit represents a block, 1 = free, 0 = allocated, easy to find free blocks, but slow and need to keep in memory
- Linked list: Each block has a pointer to the next free block, easy to locate free block and only first pointer is needed in memory but high overhead

Implementing Directories

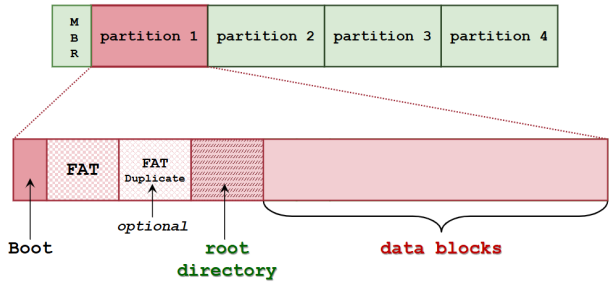
- Linear list: Each entry represents a file, slow linear search, solved by caching (name, start, length)
- Hash table: Each directory contains a hash table, fast lookup, but limited size and depends on good hash function
- File information: File name and other metadata, disk blocks information

File Operations

- Create: Locate parent directory, check for duplicates, find free disk blocks, adds an entry
- Open: Search system-wide table for existing file entry, creates an entry in process table to point to this entry and return this pointer. If entry does not exist, locate the file and load the file information into a new entry in the system-wide table.

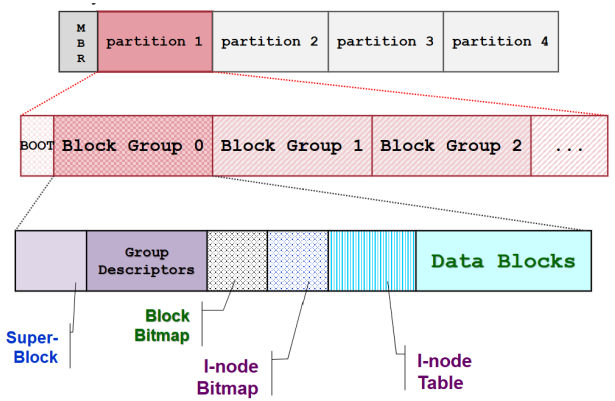
File System Case Studies

Microsoft FAT File System

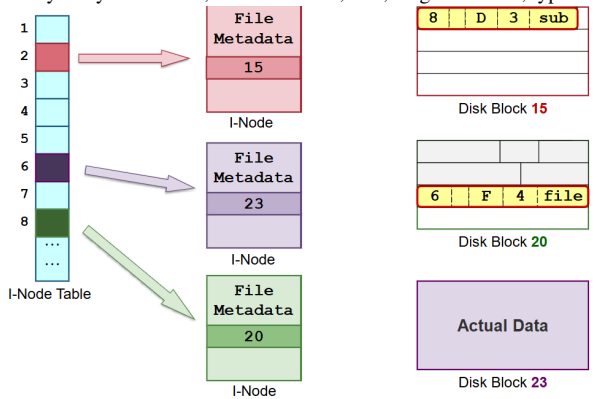


- FAT: 1 entry per data block (Free, Block number, EOF, BAD)
- Directory is a special type of file with root directory in a special location
- Directory entry in a Disk Data Block (File or subdirectory): Name + Extension (8 + 3), Creation Time and Date, First Disk Block Index
- First block (Directory entry) → Next (FAT) → EOF (FAT)
- Larger cluster size = larger usable partition but larger internal fragmentation
- Long file name can use multiple directory entries

Extended-2 File System Linnux



- Disk space is split into blocks and then grouped into block groups
- Each file/directory is described by I-node which contains file metadata, data block addresses
- Superblocks: Describes the whole file system, total I-node number, I-node per group, Total disk blocks ...
- Group Descriptor: Describes the block group, number of free blocks, number of free I-nodes, location of the bitmaps, duplicated in each block group
- Block bitmap: Track usage status of blocks of this block group
- I-node bitmap: Track usage status of I-nodes of this block group
- I-node Table: Array of I-nodes for only this block group
- I-node: 15 Disk block pointers, 12 direct pointers, 1 single indirect pointer, 1 double indirect pointer, 1 triple indirect pointer
- Data block of directory stores linked list of directory entries within this directory
- Directory entry: File name, I-node number, size, length of name, type



- Hard link points directly to the same I-node, I-node stores reference counter
- Symbolic link: Special file containing the path name of the file, no reference counter