**COMP 362 Final Review**

**Lecture 9 Main Memory**

When can the binding of instructions and data to memory addresses occur? For each binding type, explain the details of the binding process and the implications.

Slide 4

Binding can happen at 3 different stages:

Compile Time

If memory location is known in advance, then the compiler generates absolute code

Code requires recompilation if the starting location changes

Load Time

If memory location isn’t known at compile time, code must be relocatable

Binding is done when loading the program

Program must stay in memory

Execution Time

Binding delayed until runtime

The process must be movable during its execution from one memory location to another

What are the challenges of load time and execution time binding? What are the mechanisms that support relocatable code? Describe the process of loading code into any part of memory.

What are the challenges of load time and execution time binding?

Slide 4

Load Time

Program must stay in memory

Code must be relocatable

Execution Time

The process must be movable during its execution from one memory location to another

What are the mechanisms that support relocatable code? Describe the process of loading code into any part of memory.

Slide 6

Base

Beginning of the space for the process

Limit

Scope of the space

Scope = base + limit

To load the code, the code is relocated into the designated space at one of the three times

You have 10 programs that use a certain library. Explain the process of linking the programs with

* a static version of the library.
* a dynamic version of the library.

Slide 5

Static

Becomes a part of the program after linking

The code within the library is cloned to every program that uses it

Dynamic

Small piece of code called the stub is used to locate the correct memory-resident library routine

The stub then switches itself with the address of the routine and then executes the routine

The OS then checks if the routine is in the process’ memory address

Assume that the hit ratio in the MMU that performs paging with an associative memory (translation look-aside buffer (TLB)) is 90%. The access to main memory is 50 nanoseconds, and the lookup time for the TLB is 10 nanoseconds.  
  
Compute what is the effective access time of this system.  
Show the result and explain how did you obtain it.

Slide 20

EAT = Effective Access Time

TLB = Translation Look-aside Buffer

EAT = memory access time + (lookup time \* hit ratio) + (memory access time + lookup time) (1- hit ratio)

EAT = (50 nanoseconds) + (10 nanoseconds\*.9) + (50 nanoseconds + 10 nanoseconds) (1-.9)

EAT = (50 nanoseconds) + (9 nanoseconds) + (60 nanoseconds) (.1)

EAT = (59 nanoseconds) + (6 nanoseconds)

EAT = 65 nanoseconds

You have a system with 1 MB RAM. The system uses 4 kB pages. Your system loads 5 programs with the following lengths:

167852 bytes

209376 bytes

32866 bytes

254871 bytes

128527 bytes

Calculate the scope of internal fragmentation after all programs are loaded into the memory.  
Show the result and explain how did you compute it.

Slide 18

Page size is 4 \* 1024 = 4096 bytes/page  
  
167852/4096 = 41 frames with 0.1 left  
209376/4096 = 52 frames with 0.9 left  
32866/4096 = 9 frames with 0.98 left  
254871/4096 = 63 frames with 0.8 left  
128527/4096 = 32 frames with 0.6 left

Total scope of fragmentation is (0.1+0.9+0.98+0.8+0.6) (4096) = 3.38\*4096 = 13844.48

Consider the following requests (Rn) for memory allocation (A) and deallocation (D) from a memory pool of size 20:

R1 A 6

R2 A 3

R3 A 5

R4 A 2

R1 D

R5 A 4

R4 D

R6 A 1

R7 A 2

Compute explicitly (i.e., show all steps) the external fragmentation assuming the **best-fit** allocation policy.

Use the following notation:

[xxxxxxxxxxxxxxxxxxxx]

[111111xxxxxxxxxxxxxx]

[111111222xxxxxxxxxxx]

which shows the initial state and then the two following allocations.

[xxxxxxxxxxxxxxxxxxxx]

[111111xxxxxxxxxxxxxx]

[111111222xxxxxxxxxxx]

[11111122233333xxxxxx]

[1111112223333344xxxx]

[xxxxxx2223333344xxxx]

[xxxxxx22233333445555]

[xxxxxx22233333xx5555]

[xxxxxx222333336x5555]

[77xxxx222333336x5555]

Consider the following requests (Rn) for memory allocation (A) and deallocation (D) from a memory pool of size 20:

R1 A 6

R2 A 3

R3 A 5

R4 A 2

R1 D

R5 A 4

R4 D

R6 A 1

R7 A 2

Compute explicitly (i.e., show all steps) the external fragmentation assuming the **worst-fit** allocation policy.

[xxxxxxxxxxxxxxxxxxxx]

[111111xxxxxxxxxxxxxx]

[111111222xxxxxxxxxxx]

[1111112223333344xxxx]

[xxxxxx2223333344xxxx]

[5555xx2223333344xxxx]

[5555xx22233333xxxxxx]

[5555xx222333336xxxxx]

[5555xx22233333677xxx]

Consider the following requests (Rn) for memory allocation (A) and deallocation (D) from a memory pool of size 20:

R1 A 6

R2 A 3

R3 A 5

R4 A 2

R1 D

R5 A 4

R4 D

R6 A 1

R7 A 2

Compute explicitly (i.e., show all steps) the external fragmentation assuming the **first-fit** allocation policy.

[xxxxxxxxxxxxxxxxxxxx]

[111111xxxxxxxxxxxxxx]

[111111222xxxxxxxxxxx]

[11111122233333xxxxxx]

[1111112223333344xxxx]

[xxxxxx2223333344xxxx]

[5555xx2223333344xxxx]

[5555xx22233333xxxxxx]

[55556x22233333xxxxxx]

[55556x2223333377xxxx]

What is the difference between a physical address and a logical address? How is the logical address mapped to its physical counterpart? What is the lifetime of such binding?

What is the difference between a physical address and a logical address?

Slide 7

Logical

Generated by the CPU

Physical

Address seen by the memory unit

Needed to actually fetch/push data to and from memory

How is the logical address mapped to its physical counterpart? What is the lifetime of such binding?

Slide 8

Memory-Management Unit (MMU) maps the logical address to the physical

The value in the relocation register is added to every address generated by a user process

The user program deals with the logical addresses and never sees the physical addresses

Explain how paging alleviates the problems inherent to contiguous allocation of memory to programs. What is a page table? What is the difference between a page and a frame? What is a free frame list?

Explain how paging alleviates the problems inherent to contiguous allocation of memory to programs.

Slide 13

Physical memory is divided into fixed-sized blocks called frames

Logical memory is divided into blocks of the same size called pages

Paging allows for logical memory to keep a program

When needed, a process from a page is allocated into a frame to be ran

Doing this reduces fragmentation from copying all the code of a process into physical memory

What is a page table?

Slide 17

A page table is a data structure in virtual memory that keeps track of the mapping between logical and physical addresses

It does this by storing the page numbers and the page offsets

Page Number

Index into the page table that has the base address of each frame in physical memory

Each frame holds a page of the program

Page Offset

Combined with the base address to define the physical memory address

The physical memory address is then sent to the memory unit

What is the difference between a page and a frame?

Physical memory is divided into fixed-sized blocks called frames

Logical memory is divided into blocks of the same size called pages

What is a free frame list?

Slide 13

A pool of free frames

Free frames are allocated from the list or removed from processes that have too many pages

If the list is empty and no processes have any free frames, then a process might be swapped out

What would be the maximum size of a page table for a system with 48-bit addressing assuming a 1 kb frame size and assuming that 4 bytes are needed for every entry. Describe each of the strategies to keep the size of the page table in check

Slide 22

2^48 = number of addresses

2^10 = frame size

(2^48) / (2^10) = 2^38 pages

2^38 pages \* 2^2 bytes / page = 2^40 bytes

Hierarchal Pages

Slide 23

Break up the logical address space into multiple page tables

In other words, divide the page table into its own pages

Only some of the pages in the page table need to be in memory, which means it takes up less space

Hashed Page Tables

Slide 26

Common in 32-bit and up systems

Virtual page number is hashed into a page table

This page table contains a chain of elements hashing to the same location

Virtual page numbers are compared with the elements in order to find a match

If a match is found, the physical frame that belongs to the page number is extracted

Inverted Page Tables

Slide 27

One table for all processes

One entry for each real page of memory

Each entry consists of the virtual address of the page stored in that real memory location(p) as well as the info about who owns the page (pid)

The base is the offset of the entry in the entry table

Decreases space but increases the time it takes to search the table when a page is referenced

Consider a computer system that uses 5-bit addressing scheme with 3 bits for page numbers and two bits for offsets. Let the logical memory of some process be a contiguous sequence of letters from a to z (26 letters, only lower case). What is the binary version of the logical address of letter m assuming that the following is the content of the page table: [4, 1, 0, 5, 2, 7, 6, 3]? Find the decimal version of physical address of the letter m. Show the result and explain how did you derive it.

Number of pages = 2^3 = 8

Number of addresses/page = 2^2 = 4

m is the 13th letter in the alphabet and we have 4 addresses per page

Therefore, m is located at the 3rd page (4th entry in the page table) because 13/4 = 3

The binary address is 10100

101 = 5, which is the virtual address space

00 = 0, which is where in the virtual address space m is

Physical address = (page number \* page size) + offset = (5 \* 4) + 0 =20

The CPU executing a process generates (5,0,4) as a logical address in a system that uses an inverted page table to compute physical addresses. Let the following be the inverted page table: [(2,1),(5,2),(7,3),(5,0),(6,4),...]. Assuming that there are 16 frames in the memory and page size is 8 construct the binary version of the physical address corresponding to this logical address. Show the result and explain how did you derive it.

Page number = 3, (5,0) is at page index 3

Page size = 8

4 = 4th spot in the page

24+4 = 28

1 1 1 0 0

Let the logical memory of some process be a contiguous sequence of letters from a to z (26 letters, only lower case). Consider a computer system that uses 8-bit addressing scheme with six bits for two-level page indexing with the first level taking four bits, the second taking two, and the offsets another two. Assuming that the following is the content of the outer page table:

[7, -, 0, 5, -, -, -, 2, -, 9, -, 11, 13, -, 1, -]

and that the content of the third (counting from zero) frame is:

[8, 12, 10, 4]

and the content of the sixth frame is:

[3, 15, 6, 14]

show the binary version of the logical address of letter c knowing that the physical address of c is 18. Show the result and explain how did you derive it.

If the physical address of c is 18, then the physical address of a is 16

The page number of a-d is 16 (physical address) / 4 (number of bits/page) = 4

In binary, 18 is 0 0 0 1 0 0 1 0

The last two digits are transferred to the last two digits of the logical address

4 is found at index 3 of the third frame, which forms the 3rd and 4th digits of the logical address

2 is the index of the third frame, which is found at index 7 of the outer page table

The 7 is what makes up the remaining digits of the logical address, giving us an address of 0 1 1 1 1 1 1 0

How does a segmentation memory architecture differ from paging? How can they both be integrated in a hybrid architecture? Describe such a hybrid architecture with details on how logical addresses would be translated into their physical equivalents.

How does a segmentation memory architecture differ from paging?

Slide 29

Paging

Get a long linear address without consuming too much physical memory

Segmentation

2D physical addresses

Base contains the starting physical address where the segments are

Limits specify the length of the segment

Protection, including read/write/execute privileges

How can they both be integrated in a hybrid architecture? Describe such a hybrid architecture with details on how logical addresses would be translated into their physical equivalents.

Slide 30

The CPU generates the logical address

That logical address is given to the segmentation unit

The segmentation unit produces a linear address

The linear address is given to the paging unit

The paging unit generates a physical address in main memory and forms the equivalent of MMU

Describe the buddy memory allocation scheme. What are the benefits of using the buddy scheme for allocating memory for kernel needs?

Slide 33

Allocates memory from a fixed-size segment consisting of physically-contiguous pages

Memory is allocated using a power-of-2 allocator

The power of 2 is where the “buddy” portion of the name comes from

Each request is rounded up the next highest power of 2

When the needed amount of allocation is smaller than what’s available, the current chunk is split into 2 buddies of the next-lower power of two

The kernel needs to use a minimal amount of memory to minimize waste due to fragmentation

Buddy allocation keeps lowering the size of the memory chunk by powers of 2 until the request fits within the chunk

This minimizes the waste for the kernel

Describe the slab memory allocation scheme. What are the benefits of using the slab scheme for allocating memory for kernel needs?

Describe the slab memory allocation scheme.

Slide 34

A slab is made up of at least one physically contiguous page

A cache has at least one slab

Each cache has objects that are instantiations of the kernel data structure

A slab can be in one of three states:

Full

All objects in the slab are marked as used

Empty

All objects in the slab are marked as free

Partial

The slab consists of both used and free objects

The slab allocator first attempts to satisfy a request with a free object in a partial slab

If it doesn’t exist, a free object is assigned from an empty slab

If no empty slabs are available, a new slab is allocated from contiguous physical pages and assigned to a cache

Memory for this object is allocated from the slab

What are the benefits of using the slab scheme for allocating memory for kernel needs?

Slide 35

Little to no memory is wasted due to fragmentation

Each kernel gets an associated cache

Each cache is made up of slabs that get divided into chunks the size of the objects being represented

When the kernel request memory for an object, it gets the exact amount it needs

Memory requests can be satisfied quickly

The slab allocation scheme is effective for managing objects when objects are frequently allocated and deallocated

Frequent allocation and deallocation usually come with kernel requests

Objects get created in advance and can be quickly allocated from the cache

When the kernel has finished with an object, it’s marked as free and is returned to the cache

**Lecture 10 Virtual Memory**

Explain what a virtual memory is including the role of demand paging realized by the lazy swapper in memory virtualization. Furthermore, describe the benefits of memory virtualization.

Explain what a virtual memory is including the role of demand paging realized by the lazy swapper in memory virtualization.

Slide 6

Virtual memory is a further separation of logical memory from physical memory

Only part of a program needs to be in memory for execution

This means that virtual logical addresses can be longer than physical address space

Each process has a memory map that maps process pages into physical memory frames

The difference from paging is that a frame might be either in the physical memory or on a disk

If a frame isn’t in physical memory, then the OS fetches it from the disk

Lazy swapper

Swaps a page into memory only when needed

Furthermore, describe the benefits of memory virtualization

Slide 7

Flexible process management

A process doesn’t need to wait for a chunk of memory to fit in

Only a portion of a process needs to be loaded into memory

This results in better system response

Total logical space for all processes can also be larger than the available physical address space

This results in more concurrent processes and users

Addresses can be shared by several processes, which increases the chances the frames are in the memory

Less I/O

Fewer bytes to swap around

Improved performance

Explain what a page fault is, how it is detected, and describe step-by-step the process of page replacement

Slide 9

A page fault is when the first reference to a page traps the OS

It’s detected by the OS constantly the validity of the reference for a given process

If the reference is invalid, then the process is aborted

If it’s valid but not found in memory, then the page is brought into memory

An empty frame is retrieved so that the page can be swapped into the frame

The page and frame tables are both reset

The validation bit is then set to valid

Finally, the instruction that caused the page fault is restarted

Page Replacement

Occurs when there’s no free frame for the missing page

First find a frame in memory that holds a page that’s currently not in use

Swap the page out and free the frame

Swap in the page that’s needed

The same page might need to be swapped in and out of memory multiple times

Use a modify bit to reduce the overhead of page transfers

Pages marked with the modify bit are written to the disk when replaced

Discuss local frame allocation dilemma; i.e., options for distributing frames between processes.

Slide 21

Global Replacement

Processes chooses a replacement frame from all of the frames available

One process may be given a frame that was taken away from another

Local Replacement

Each process only selects from its own set of frames allocated to it

Fixed Allocation

Either each process gets the same number of frames or the number is determined by its size

Dynamic Allocation

A proportional allocation scheme that uses priorities instead of size

If a process makes a page fault, a frame from a process with a lower priority number is used

Compute the memory effective access time in a system with the following characteristics:

* page faults happen once every 2000 memory accesses on average,
* disk access time is 5 ms,
* probability that the dirty bit is set on the vitctim page is 0.1,
* memory access time is 100 nanoseconds,
* page fault overhead is 7 nanoseconds, and
* restart overhead is 3 nanoseconds.

Slide 12

EAT = (1 – p) \* memory access + p \* [ page fault trap overhead + swap page out + swap page in + restart overhead ]

EAT = (1 – (1/2000)) \* 100 + (1/2000) \* [7 + 5000000 + 5000000 + 3]

EAT = 5099.955 nanoseconds = 5.1e-3 ms

Assume that an OS with 3 frames uses OPT replacement policy . For the following reference set

{4, 3, 6, 2, 1, 2, 5, 4, 3, 4, 2, 3, 6, 1, 2, 3}

1. Show with details all page replacements.
2. State clearly how many page faults have occurred.

Show frame content (which pages are loaded) for each page reference per line. Use "-" for empty frame; otherwise, use the page number of the page loaded into a particular frame. Precede the number corresponding to the page being referenced with "<". If there is a page fault, indicate which page was replaced by preceding the corresponding number with "\*".

For example, the following line:

- 4 2<

indicates that the first frame is free, the second frame holds page number 4, and the page number 2 held in the third frame has just been referenced. The following line:

6\* 4 2

indicates that page 6 was referenced and faulted; it indicates that page replacement occurred in the first frame.

For example, in FIFO, the following string:

1 1 5 2 4 2 0 2 5 1 0 5 0 2 3

will be processed as follows:

1\* - -  
 1< - -  
 1 5\* -  
 1  5 2\*  
 4\* 5  2  
 4  5 2<  
 4 0\* 2  
 4  0 2<  
 4  0 5\*  
 1\* 0  5  
 1 0< 5  
 1  0 5<  
 1 0< 5  
 1 2\* 5  
 1  2 3\*

4\* - -

4 3\* -

4 3 6\*

4 3 2\*

4 1\* 2

4 1 2<

4 5\* 2

4< 5 2

4 3\* 2

4< 3 2

4 3 2<

4 3< 2

6\* 3 2

1\* 3 2

1 3 2<

1 3< 2

Assume that an OS with 3 frames uses FIFO replacement policy. For the following reference set

{4, 3, 6, 2, 1, 2, 5, 4, 3, 4, 2, 3, 6, 1, 2, 3}

4\* - -

4 3\* -

4 3 6\*

2\* 3 6

2 1\* 6

2< 1 6

2 1 5\*

4\* 1 5

4 3\* 5

4< 3 5

4 3< 5

4< 3 5

4 3 2\*

4 3< 2

6\* 3 2

6 1\* 2

6 1 2<

6 1 3\*

Assume that an OS with 3 frames uses LRU replacement policy. For the following reference set

{4, 3, 6, 2, 1, 2, 5, 4, 3, 4, 2, 3, 6, 1, 2, 3}

4\* - -

4 3\* -

4 3 6\*

2\* 3 6

2 1\* 6

2< 1 6

2 1 5\*

2 4\* 5

3\* 4 5

3 4< 5

3 4 2\*

3< 4 2

3 6\* 2

3 6 1\*

2\* 6 1

2 3\* 1

Explain what Bellady's Anomaly is.

Illustrate the problem by computing page faults in an OS that uses memory with three frames and with four frames.

Use the following reference set:

{1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5}

Sometimes increasing the number of frames also increases the number of page faults

1 4 5

2 1 3 🡪 9 page faults

3 2 4

1 5 4

2 1 5 🡪 10 page faults

3 2 2

4 3 3

Assume the following page reference sequence:

{0, 3, 4, 3, 3, 0, 6, 5, 5, 6, 0, 2, 1, 2, 4, 3, 3, 4, 2, 1, 1, 1, 2, 3, 4, 5, 6, 4, 5, 6, 5, 5, 6, 6, 1}

Show all successive working sets using a window  Δ = 4. Use 4 reference bits that are set from the left for the pages that are referenced in the current window and then shifted right on every timeout. The "timeout" is simulated by counting page references; in this task assume a timeout every two page references.

Write C code that declares appropriate data structures and implements the algorithm using bit shifting to maintain the references bits. You do not need to use packing in this solution (i.e., you are allowed to utilize just 4 bits in a byte to hold the reference information

Step 1: Slide the window

Step 2: Do the right shift

Step 3: Look at everything within the window and mark the reference

Every 2 references is when the reference table gets updated

First working set is the first 2 in the window

Second working set takes into account all 4 in the window

When the window gets shifted, all 4 are taken into account

If the number of page references isn't a multiple of the window, then just use however many end up in the window

Can a stack the top of which always keeps the last referenced page be the foundation for implementing the LRU page replacement algorithm? How can you find the victim page? Is it an efficient approach to implementing the policy? Consider both array-based and linked-list-based implementations of a stack. For each, explain step-by-step the procedure to keep the last referenced page at the top of the stack. Next, discuss the complexity of each of the implementation.

Can a stack the top of which always keeps the last referenced page be the foundation for implementing the LRU page replacement algorithm? How can you find the victim page?

Slide 17

This follows the LRU Quasi-Stack implementation

To find the victim, you would reference the bottom of the stack

Is it an efficient approach to implementing the policy? Consider both array-based and linked-list-based implementations of a stack. For each, explain step-by-step the procedure to keep the last referenced page at the top of the stack. Next, discuss the complexity of each of the implementation.

This is an efficient approach because it requires no searching for a replacement

Array

Make a temp array

Place the last element of the stack array at the first entry of the temp array

Loop through the temp array starting at the second entry and copy each element from the stack array into the corresponding entry in the temp array

Copy the temp array into the stack array

Linked List

Make a temp node

Copy the data from the last node on the stack list into the temp node

Have the next of the temp node point to the head

Make the new head the temp node

Iterate through the list until the next of the current node points to the last node

Have the next of the second-to-last node point to null

Explain how to implement efficient page replacement algorithms that approximate LRU. What is a "second chance" approach?

The idea is to essentially approximate the age/usage of a page and use that to compute the LRU

Algorithm

Each page gets a reference bit

All of the reference bits get initialized to 0

When a page is referenced, the reference bit is changed to 1

To search for the LRU, check the reference bits in a cycle

If the bit is 1, set it to 0 and continue

If the bit is 0, set the bit to 1 and return the page as the LRU

Second Chance

Uses a second bit that acts as a clock

The first reference bit is shifted to the second one

If the page to be replaced has its first reference bit as 0 and second at 1

Set the second bit to 0

Leave the page in memory

Replace the next page (based on clock order) with bit 0 and apply the same rules

Let’s consider the following array:

char data[4][4];

on a trivial machine with the memory consisting of a single 4-byte frame.

1) How many pages does data encompass?

Consider two alternate ways to traverse data:

// A

for (i = 0; i < 4; i++)

for (j = 0; j < 4; j++)

data[i][j] = 0;

// B

for (j = 0; j < 4; j++)

for (i = 0; i < 4; i++)

data[i][j] = 0;

2) Assuming that the row major array order is used on this machine (i.e., arrays are stored by rows — like in C), calculate how many page faults will occur executing A and how many will occur executing B.

1)

Formula: (total bytes)/(bytes/page) = total pages

(4\*4)/(4) = 4 pages

2)

A will have four page faults since there will be a page fault each time a new row is started

For B, in memory each page will still be put in order of the current row

In other words, every time we want to access an element we have a page fault

Therefore, B will have 16 page faults since there are 16 elements total in data

**Lecture 11 File System Interface**

What kind of data are necessary for maintaining open files in operating systems. Where is the data kept?

Slide 5

Name

Only info kept readable for humans

Identifier

Unique tag that identifies a file within the file system

Is stored as a number

Type

Needed for systems that support different types (e.g. file, folder)

Location

Pointer to file location on device

Size

Current file size

Protection

Controls who can read, write, and execute

Time, Data, and User Identification

Data for protection, security, and usage monitoring

All of this stuff gets stored in the directory structure

Describe the ways for an operating system to tell the type of a file?

Slide 13

Some systems use file extensions (e.g. Windows)

Others have internal meta-data associated with files

Unix -> magic number stored at beginning of the file indicates the type

macOS -> Keeps a reference (capable of reassignment) to the app that created the file

Describe with details all types of directory structures that are discussed in the study notes.

Slide 19-22

Single-Level Directory

Single directory for all users

Problems

Searching -> lost of files to search through

Naming -> can’t have duplicates

Grouping -> No support for grouping other than by name

Security -> shared space

Two-Level Directory

Separate directory for each user

A path name is used in place of a file’s actual name

This allows for duplication so long as the files are from different users

Has efficient searching per user

The problem is that there’s no flexibility

Tree-Structured Directories

One root

Efficient searching

Grouping capability

Current (working) directory is associated with every process

The current directory of each user’s session starts at the user’s home directory in the login shell

That directory is also inherited by all children processes

Acyclic-Graph Directories

Allows for shared subdirectories and files

Two different names (aliasing)

The problem with this is if a file reference gets deleted and that file reference was pointing to an alias, then there is a dangling pointer

To solve this, we have back pointers and reference counts

Back Pointer

Points back to the original file reference

Doesn’t work well because the size of the supporting file data is variable this way

Reference Counts

Indicates the number of active pointers to a file

Much better solution

Allowing a general graph to represent a directory structure has a number of pitfalls. Describe them along with potential solutions.

A problem that comes with using a general graph is that we don’t want cycles to be introduced

To solve this, links should be restricted only to files and not include directories

Even with this, there is still the potential for dangling pointers

This would require the use of reference counts

Every time a new link is added, an algorithm for detecting cycles should be ran

The problem with that is that doing so is expensive

Thoroughly explain a difference between Unix soft links and hard links to files.

Soft Link (Symbolic)

New directory entry type

Link -> another name (pointer) to an existing file (alias)

Resolve the link -> follow the pointer to the directory to locate the file

Can cross the file system

Allows linking between directories

Has different inode number and file permissions than the original file

Permissions don’t ever get updated

Only has the path to the original file, not its contents

Hard Link

Points to the same file content, but also has fully-fledged directory entries

Hard link count shows how many links exist

The file isn’t removed until the hard link count reaches 0

Can’t cross file system boundaries (Only works in the same file system)

Can’t link directories

Has the same inode number and file permissions of the original file

Permissions are updated if the original file’s permissions are updated

Contains the contents of the original file

The contents can be viewed and live on after the original file is moved/deleted

1. Show Unix commands that change access modes to protect the following files:
   * A (you can do anything, your group can only read, nobody else can see)
   * B (everybody can execute, but only you can write)
   * C (everybody has total access to this non-executable)
2. Show a Unix command that changes the access mode to protect directory D, so only you can open it.

1)

A -> -rwxr-----

B -> --wx--x—x

C -> -rw-rw-rw-

2)

drw-------

Describe how to mount and unmount file systems on Unix. How to list the mounted file systems.

All files in a Unix system are arranged in a big tree known as the file hierarchy

The tree starts at “/” and can be spread out over several devices

The mount command attaches a filesystem to the file tree

The command goes like this:

mount -t *type device dir*

type is the type of device the filesystem you wish to mount is on

dir is the pathname that refers to the root of the filesystem

As long as the file system remains mounted, the previous contents, owner, and mode of dir are invisible

Unmount detaches the specified filesystem from the file hierarchy

The specified filesystem is given by the directory where it’s been mounted

A file system can’t be unmounted when a process is still running

To display a list of mounted file systems, run the findmnt command

What is the difference between the following command? Which one always requires super user access?

> chmod  
> chgrp  
> chown

chmod

Short for change mode

The mode is the mode of access

chmod *options* *permissions* *file name*

Example:

chmod u=rwx,g=rx,o=r myfile

chgrp

Short for change group

Changes the group ownership of the file

Applies to the specified file or the specified directory and all of its subdirectories

Example

chgrp -hR staff /office/files

chown

Short for change owner

Changes who owns the file

Can also change the group owner as well

The user can belong to multiple groups, but the command will use the current group

Example

sudo chown hope file.txt

A user executes the following commands in bash in an empty directory:

> umask 77  
> touch test\_file\_A  
> umask 27  
> touch test\_file\_B  
> umask 2  
> touch test\_file\_C

What will be the system response to the following command:

> ls -l

Explain how did you derive the answer by describing the results of each command.

umask 77 gives all access rights to the owner and group, but no access is given to everybody else

umask 27 gives write access to the owner, all access rights to the group, and no access to everybody else

umask 2 gives only write access to the owner and both the group and everybody else has no access

ls -l will yield:

-rwxrwx--- test\_file\_A

--w-rwx--- test\_file\_B

--w------- test\_file\_B

Explain the meaning and translate the following access modes to their letter counterparts:

777  
600  
400  
644  
666

The first digit represents owner permissions

The second digit represents group permissions

The third digit represents everybody else

Each digit is further split up into read, write, and execute permissions once converted to binary

For instance, 1 0 0 represents only read permissions since the read flag is flipped to 1

777 -> -rwxrwxrwx

600 -> -rw-------

400 -> -r--------

644 -> -rw-r--r--

666 -> -rw-rw-rw-

Explain the meaning and translate the following access modes to their octal counterparts:

rwx------  
rwxr--r--  
r--r--r--  
rw-rw-rw-  
rwxrwxrwx  
rw--w--w-

r=read

w=write

x=execute

Each group of 3 represents owner, group, and everybody else permissions respectively from left to right

A – means that the access right isn’t granted

rwx------ -> 700

rwxr--r-- -> 744

r--r--r-- -> 444

r--r--r-- -> 666

rwxrwxrwx -> 777

rw--w--w- ->622

**Lecture 12 File System Implentation**

Describe with details a typical memory-resident structures that support file system after it has been mounted. What is a File Control Block (FCB)? What is its purpose? Using the memory organization and data structures that you have just described, provide pseudocode for opening a file, and for reading from an opened file.

Describe with details a typical memory-resident structures that support file system after it has been mounted

Slide 8

Directory Structure

Indicates where file contents are on the disk

Open-File Tables

File descriptors of opened files

System-wide

Per process

What is a File Control Block (FCB)? What is its purpose?

Slide 5

File system data structure

Located on the disk

Contains the details about an individual file/directory

Using the memory organization and data structures that you have just described, provide pseudocode for opening a file, and for reading from an opened file.

Verify the file exists with the directory structure

If it doesn’t throw an error

Verify the FCB isn’t null

If it is, throw an error

If the file is already on the open-file table

Read the data blocks from the file

Else

Move the file to the open-file table

Mark that the file is open in its FCB

Read the file’s data blocks

Describe the organization of file system directories based on hashing functions; discuss its advantages and disadvantages.

Slide 10

File system directories arranged into a hash table

Direct access to a file is granted by applying a hash function to the file’s name

Advantages

Decreases directory search time

Disadvantage

Must deal with collisions

No efficient solution to these collisions exists

Fixed size

Describe the linear organization of a file system directory and analyze its advantages and disadvantages.

Slide 10

A linear list of file names with pointers to data blocks

Advantages

Simple to program

Variable size

Disadvantages

Sequential access only

Time consuming search

Could use a higher-level structure like a sorted tree

Caching in memory alleviates the problem

However, a complete copy can’t be kept if the system is too large

Describe the indexed allocation scheme for allocating space for files. What are the advantages of this organization?

Slide 16

Brings all pointers together into the index block

Needs an index table

Random access

Advantages

Dynamic access without external fragmentation

Overhead of index block

Multilevel indexing helps

Index table to index table until the data blocks are reached

Need to balance fragmentation and access time for many indices

Describe with details how to use linked lists for managing free space in a file system. What are the advantages and disadvantages of this approach?

Slide 13

Linked allocation

Each file is a linked list of disk blocks

The blocks themselves can be anywhere on the disk

Advantages

The simplicity of it means that only the starting address is needed

The free-space management system means that no space is wasted

Disadvantages

No random access

Must go through the whole list to get anywhere

Very difficult to recover from errors

Describe with details how to use a bit vector for managing free space in a file system. What are the advantages and disadvantages of this approach? *(i.e., compare to other methods to manage free storage)*

Slide 19-20

A bit vector contains bits that represent all of the blocks in the file system

Each bit indicates whether the block is free or not

Advantages

Efficient computation of the location of the bit for a specific block

Fast lookups for free blocks

Disadvantages

Requires extra space

Has a problem with increasing the size of the disk

A possible solution is to divide the blocks into groups and give each group its own bit vector

While doing this, try to keep files inside a single group

If you have a disk with 1,048,576 blocks, how many bytes do you need to allocate space for the bit vector in a file system using a bit vector-based management of free space assuming that a single bit in a byte indicates the status (free or taken) of one disk block? Assuming that bit 0 is used to indicate a free block, and bit 1 to indicate a taken block, write a fast C function that implements the algorithm to find, and to claim, the first (from the beginning of the logical address space) free block. The function should return the logical block number of the block and flip the corresponding bit to indicate that the block is taken (not free anymore).

1,048,576/8 = 131072 bytes

Unsigned short findAndClaimFirstFreeBlock

{

unsigned int i = 0;

while (bitvector[i] == 0xFF)

i += 1;

register unsigned char mask = 0x80;

CIFS\_INDEX\_TYPE j = 0;

while (bitvector[i] & mask)

{

mask >>= 1;

++j;

}

inline freeBlockIndex = (i \* 8) + j;

unsigned short blockIndex = freeBlockIndex/8;

unsigned short bitShift = freeBlockIndex%8;

register unsigned char mask = 0x80;

bitvector[blockIndex] ^= (mask >> bitShift);

return freeBlockIndex;

}

Describe what Virtual File System is. What's its purpose? How is it used?

Slide 9

A VFS provides an object-oriented virtualization layer for implementing file systems

Allows for the same API to be used for different types of file systems

The API is to the VFS interface rather than any specific kind of file system

VFS requests have to be interpreted by specific file system modules

Key Virtual Objects in VFS

Inode -> an object that represents an individual file

File -> an object that represents an open file

Superblock -> an object representing a file system

Dentry -> an object representing a directory entry

Each of these objects have a number of operations defined for it

Describe with details how a log-structured file system works.

Slide 24

Write info to media into new blocks

There is no rewriting and the info gets written along with meta data

The structure itself is a circular list buffer

New/modified info is added at the head

Old data is marked as obsolete when new data is written

A garbage collector is used to make room at the tail

If the info is obsolete, move on the tail and make the obsolete blocks available

If the info is valid, rewrite to the head and then move on to the tail

A disk is considered full if there are no obsolete blocks in the disk

If the system crashes, then the system is guaranteed to be consistent

This is because the old data will still remain and will only be obsolete if the new data is successfully written

Describe with details how journaling file system provides data protection from system crashes.

Slide 25

Provides data consistency protection

Records each update to the file system as a transaction

Has a circular structure similar to log-structured, but only the transactions are logged, not data

All transactions are written to a log that acts as a to-do list

The log is part of the medium committed for that purpose

A transaction is ready to be processed once it’s written to the log

At this point, as far as process requesting the operation is concerned, the operation is finished

The file system itself isn’t updated until after the transaction is processed

Transactions in the log are processed asynchronously and the data associated with the transactions are written to the file system

The transaction is removed form the log only after the file system has been confirmed modified

Slide 26

If the system crashes, whatever transactions in the log will still be processed upon restart

If the transactions are committed, then the system is guaranteed to remain consistent

Transactions that are partially logged aren’t executed

They may need to be rolled back depending on if the requested actions did something damaging to the system

Describe with details how copy-on-write file system provides crash protection.

If multiple processes need the same resources, then they each get pointers to the same resource

If a process tries to modify the resource, then a copy of that resource is made to be modified

This helps in providing snapshots of systems

Should the computer crash or need to revert back to an older state, there will be snapshots that can be used to restore the system to a previous state

**Lecture 13 Mass Storage Structure**

Explain what NAS and SAN are. Describe the difference between the two.

Slide 5

Network Attached Storage (NAS)

File-level storage made available over a network rather than over local connections made through hardware

Uses network protocols

NFS historical

CIFS/SMB are common protocols

AFP in Macs

Implemented with remote procedure calls (RPCs) between local host and storage host

iSCSI protocol uses IP network to carry the SCSI protocol

Slide 6

Storage Attached Network (SAN)

Common in large storage environments

Rather than files, blocks are read and written to

Multiple hosts are attached to multiple storage arrays

This allows for great flexibility, such as providing storage on-demand

The attached devices appear as local, which means that local protocols are used rather than network access

Implement two functions that translate a logical block address (LBA) into a cylinder-head-sector (CHS) address, and reverse, translating a CHS address into its LBA equivalent

typedef struct chs {

int cylinder;

int head;

int sector;

} CHS\_TYPE;

typedef long LBA\_TYPE;

CHS\_TYPE lba2chs(LBA\_TYPE lba);

LBA\_TYPE chs2lba(CHS\_TYPE chs);

Using the algorithms, compute the location of the logical block 162656 in a hard drive with 200 cylinders, 128 sectors, and 5 two-sided platters. Two-sided platters are served by two heads; one for each side. Assume that a logical block is the same size as a sector and that there are no bad sectors on this perfect disk.

SIM\_DEV\_RET\_CODE lba2chs(lba\_t lba, chs\_t \*chs)

{

if (lba >= MAX\_LOGICAL\_BLOCK)

return SIM\_DEV\_ADDRESS\_ERROR;

// todo: implement

chs->cyl=lba/(NUM\_OF\_HEADS\*NUM\_OF\_SECTS);

chs->head=(lba/NUM\_OF\_SECTS)%NUM\_OF\_HEADS;

chs->sect=(lba%NUM\_OF\_SECTS)+1;

return SIM\_DEV\_SUCCESS;

}

SIM\_DEV\_RET\_CODE chs2lba(chs\_t \*chs, lba\_t \*lba)

{

// todo: implement

lba=(chs->cyl\*NUM\_OF\_HEADS+chs->head)\*NUM\_OF\_SECTS+(chs->sect-1);

return SIM\_DEV\_SUCCESS;

}

Explain why some aspects of IO scheduling are addressed by the operating system rather than the disk controller. Give concrete examples. What aspects of optimization would be more appropriate in the disk controller?

Slide 17

The OS knows about constraints that the disk doesn’t

Example

Demand paging takes priority over application I/O

Writing is more important than reading if the cache is almost full

OS may need guaranteed write ordering

Rotational latency from the OS is something that is hard to address

Disk controllers perform some optimization in the form of request ordering, coalescing, and sector number schemes

Explain the difference between a single-level-cell-based SSD and multi-level-cell-based SSD.

How the cells are organized represents the data stored on the SSD

Single

One bit per cell

0 or 1

More expensive

More reliable

Only available in smaller capacities due to the amount of real estate it takes up on the drive

Multi

Usually takes 4 levels, which is equivalent to 2 bits

Lower cost per bit

Has higher capacities

Not as durable

Define and explain the difference between RAID 1+0 and RAID 0+1. Which is better?

RAID = redundant array of independent disks

RAID schemes improve performance and reliability by storing redundant data

Mirroring/shadowing keeps a duplicate of each disk

Block interleaved parity uses much less redundancy, but still allows for recovery from errors

RAID specifies 6 ways to arrange multiple disks

Disk striping uses a group of disks as one storage unit

Doing this improves disk usage and efficiency

RAID 1+0

Mirror, then stripe

RAID 0+1

Stripe, then mirror

RAID 1+0 is slightly better when a disk fails

This is because the whole stripe is available, since each of the elements work as a pair

With the disk array online, replication to a new disk might be done

Also resistant to some two-disk failures

In RAID 0+1, if a disk fails, then the whole stripe is unusable

It can be used again if the disk is replicated to a new disk

The entire RAID is lost if two disks fail on each side of the mirror

Describe:

1. advantages and disadvantages of solid-state drives (SSDs) in comparison with disk hard drives,
2. what wear leveling is,
3. why is wear leveling used in the current SSDs, and
4. why wear leveling is a problem for ensuring information security.

1)

Slide 20

Advantages

Faster start, read, and write

No noise because there isn’t any spinning

Has mechanical reliability due to less moving parts

File fragmentation isn’t an issue because it has direct access to any location

Problems usually occur when writing, which can be repeated in good cells

Not as hungry for power

Disadvantage

More expensive

Comparatively lower storage capacity

Limited write cycles

2)

A process some SSD controllers use to increase the lifetime of memory

Essentially distributes all writing on all blocks of an SSD evenly across the drive

All cells receive the same number of writes to avoid writing too often to the same blocks

3)

Increases the lifetime of memory

Essentially reduces the use of all the cells, which increases the time before a cell fails due to overuse

4)

Assume that there are 256 cylinders in a disk and the current disk controller request queue is as follows:

67, 223, 45, 183, 233, 12, 238, 54, 29, 218, 156

and that the current position of the disk head is 85.

Compute how many cylinders the disk head will have to cross using FCFS.

Show sequences of targets for each scheduling method.

|85-67| + |67-223| + |223-45| + |45-183| + |183-233| + |233-12| + |12-238| + |238-54| + |54-29| + |29-218| + |218-156|

13 + 156 + 178 + 138 + 50 + 221 + 226 + 184 + 25 + 189 + 62 = 1442

Assume that there are 256 cylinders in a disk and the current disk controller request queue is as follows:

67, 223, 45, 183, 233, 12, 238, 54, 29, 218, 156

and that the current position of the disk head is

85

Compute how many cylinders the disk head will have to cross using SSTF.

Show sequences of targets for each scheduling method.

|85-67| + |67-54| + |54-45| + |45-29| + |29-12| + |12-156| + |156-183| + |183-218| + |218-223|

= 284

Assume that there are 256 cylinders in a disk and the current disk controller request queue is as follows:

67, 223, 45, 183, 233, 12, 238, 54, 29, 218, 156

and that the current position of the disk head is

85

Compute how many cylinders the disk head will have to cross using SCAN.

Show sequences of targets for each scheduling method.

Assume that the head starts to move in the direction of increasing cylinder numbers.

|85-156|+|156-183|+|183-218|+|218-223|+|223-67|+|67-54|+|54-45|+|45-29|+|29-12|+|12-0|

=361

Assume that there are 256 cylinders in a disk and the current disk controller request queue is as follows:

67, 223, 45, 183, 233, 12, 238, 54, 29, 218, 156

and that the current position of the disk head is

85

Compute how many cylinders the disk head will have to cross using C-SCAN.

Show sequences of targets for each scheduling method.

Assume that the head starts to move in the direction of increasing cylinder numbers.

|85-156|+|156-183|+|183-218|+|218-223|+|223-255|+|255-0|+|0-12|+|12-29|+|29-45|+|45-54|+|54-67|

=611

Assume that there are 256 cylinders in a disk and the current disk controller request queue is as follows:

67, 223, 45, 183, 233, 12, 238, 54, 29, 218, 156

and that the current position of the disk head is

85

Compute how many cylinders the disk head will have to cross using LOOK.

Show sequences of targets for each scheduling method.

Assume that the head starts to move in the direction of increasing cylinder numbers.

85->156->183->218->223->238->67->54->45->29->12

Assume that there are 256 cylinders in a disk and the current disk controller request queue is as follows:

67, 223, 45, 183, 233, 12, 238, 54, 29, 218, 156

and that the current position of the disk head is

85

Compute how many cylinders the disk head will have to cross using C-LOOK.

Show sequences of targets for each scheduling method.

Assume that the head starts to move in the direction of increasing cylinder numbers

85->156->183->218->223->233->238->12->45->54->67

**Lecture 14 IO Systems**

Using bit fields in a C struct design a type for variables that can be used to encode the following controls for an imaginary serial port:

* a choice between full-duplex and half-duplex communication,
* enabling or disabling parity checking with selection for no parity, odd parity, or even parity,
* selection of the word of data length that can range from 5 to 8 bits, and
* selection of  one of the 8 speeds supported by the port.

For each field, provide comments that describe the value choices for each of the options.

When the struct is ready, declare a variable of its type, and then using the hex format set the fields to configure the port for communicating over a full-duplex, odd-parity, 6-bit-word, and the highest speed available channel.

typedef struct serial\_r{  
  
unsigned duplex: 1;  
  
unsigned parity: 2;  
  
unsigned word\_len: 2;  
  
unsigned speed: 3;  
  
} SERIAL\_R  
  
SERIAL\_R register= {0x0, 0x1, 0x3, 0x5};

Describe what a device port is and how it is used to communicate with a device. Describe a sample layout of the port.

A physical connection between a system and other systems/peripheral devices

Uses 1-2-1 communication policy

Uses port mapped IO (PMIO) addressing

This is done using specialized IO instructions which communicate with the devices

The same addresses can be used to access memory

A layout of a port gives a range of addresses and what the addresses are reserved for

Describe what are block devices and character devices. How do they differ? What's the difference in how the OS needs to handle each of them?

Block Device

Disk drives

read (), write (), seek () commands

Raw IO or file-system access

Memory-mapped file access possible

Character Device

Keyboards, mice, serial ports

Commands include get and put

Libraries layered on top allow line editing

What is polling? How does a polling-based I/O differ from interrupt-based I/O?

Polling is a method of communicating with a device by constantly "polling" it. It will determine which state the device is in (command ready, busy, error). And with this, the OS can then determine which steps to take with the device.  
  
Polling differs from interrupts in that polling will "interrogate" the system every time even if the device is not ready and will thus create the busy-wait cycle. This is seen as very inefficient. While, interrupts are created directly by the hardware itself via the interrupt controller or in a memory subsystem through the memory mapping controller. With this in mind, the device will let the system know of its status. This eliminates any need to "poll" a device within every cycle.

Describe in general terms what buffering is? Why is buffering needed in dealing with I/O from and to devices? How is buffering different from caching? How is buffering different from spooling?

In general terms, buffering is when we store data in memory while transferring between devices.  
  
Buffering is needed in dealing with I/O to and from devices for several reasons. One is to cope with any device speed mismatch. Another is to cope with transfer size mismatches between devices. A third is to maintain "copy semantics" (collecting coherent chunks).  
  
Buffering is different from caching in that caching is fast memory holding copy of data and there is always a copy in the cache. A buffer on the other hand will hold the only copy of data for a set amount of time.  
  
Buffering is different from spooling in that spooling is holding output for a device. Spooling is also able to resume its operations after interruptions.

Describe what a device driver is and how it helps the operating system to operate with multitude of devices out there.

A device driver is a software module that can be plugged into an operating system to handle a particular device. It acts as sort of this interface between the system/OS and a device. It mainly helps the OS in that a device driver contains an interface and device-dependent code such that the code had specific register or read/writes specifically for the device it is written for.

Compare port-based communication with bus-based communication.

Port

1-2-1 communication channel

Bus

Many-2-many communication channel

Daisy chain or shared direct access

Describe how CPU communicates with devices using Memory-Mapped I/O (MMIO). Describe how CPU communicates with devices using Port-Mapped I/O (PMIO). Explain the key difference between MMIO and PMIO.

PMIO

Use special IO instructions to communicate with devices

Doesn’t carve from memory

Same addresses can be used to access memory

MMIIO

Same instructions and same address bus used to communicate with both devices and memory

Part of the memory carved for devices

The addresses used to communicate with devices can’t be used to access memory