CS40 Operating Systems: Final Exam Study Guide

Chapter 7 Outline

Memory Management Requirements

Relocation - In a multiprogramming system, RAM is shared among a variety of processes who are constantly swapping in/out to/from disk. Thus, there is a possibility that a process won't be placed at the same physical address.

Protection - Many programs running concurrently have an inherent risk of writing into the address space of another program. Every process must be protected from this risk.

Sharing - It is advantous to allow several programs to share a portion of data loaded into main memory as opposed to each having their own copy. However, this should not compromise the protection of any processes.

Logical Organization - Main memory is organized linearly or in a one dimensional address space. Programs are written in modules (can be run independently), with varying degrees of protection (read, write, execute). The MMU must also facilitate the sharing of modules with multiple processors.

Physical Organization - The programmer should not be responsible for overseeing memory management. Flow of information from RAM and secondary memory should:

- Overlaying -
- Programmer does not know about the available memory at compiling time nor where that space may be located.

Chapter 8 Outline

Virtual Memory - a storage allocation scheme that addresses secondary memory as though it is a constituent of the RAM. This technique maps program-generated virtual addresses into physical addresses within the memory hardware through (1) address translation and (2) virtual address space management. All virtual management implementations have the following advantages:

1. Memory references are dynamically translated into physical addresses at run time. This allows the processor to occupy different memory addresses during the execution of a program. This allows a process to be swapped in and out as required.

2. A consequence of the above is that processes can be partition into several data blocks which need not be located contiguously in RAM.

A popular technique for virtual memory implementation is known as Paging. This technique partitions the virtual address space of a process and the allocated physical address space into equally fixed data blocks known as pages and frames, respectively. The MMU utilizes a page table entries to store mappings between physical and virtual addresses. Modern virtual memory schemes implement a high-speed cache, called the translation lookaside buffer (TLB) to store the recently used page table entries.

1. Given a virtual address, traverse through the TLB
2. If table entry is found, frame number is retrieved and physical address is formed.
3. If entry is not found, the virtual address' corresponding page number is indexed in the process page table.
4. If entry is neither in the TLB or Process Page Table, check main memory
5. Otherwise, we issue a page fault procedure.

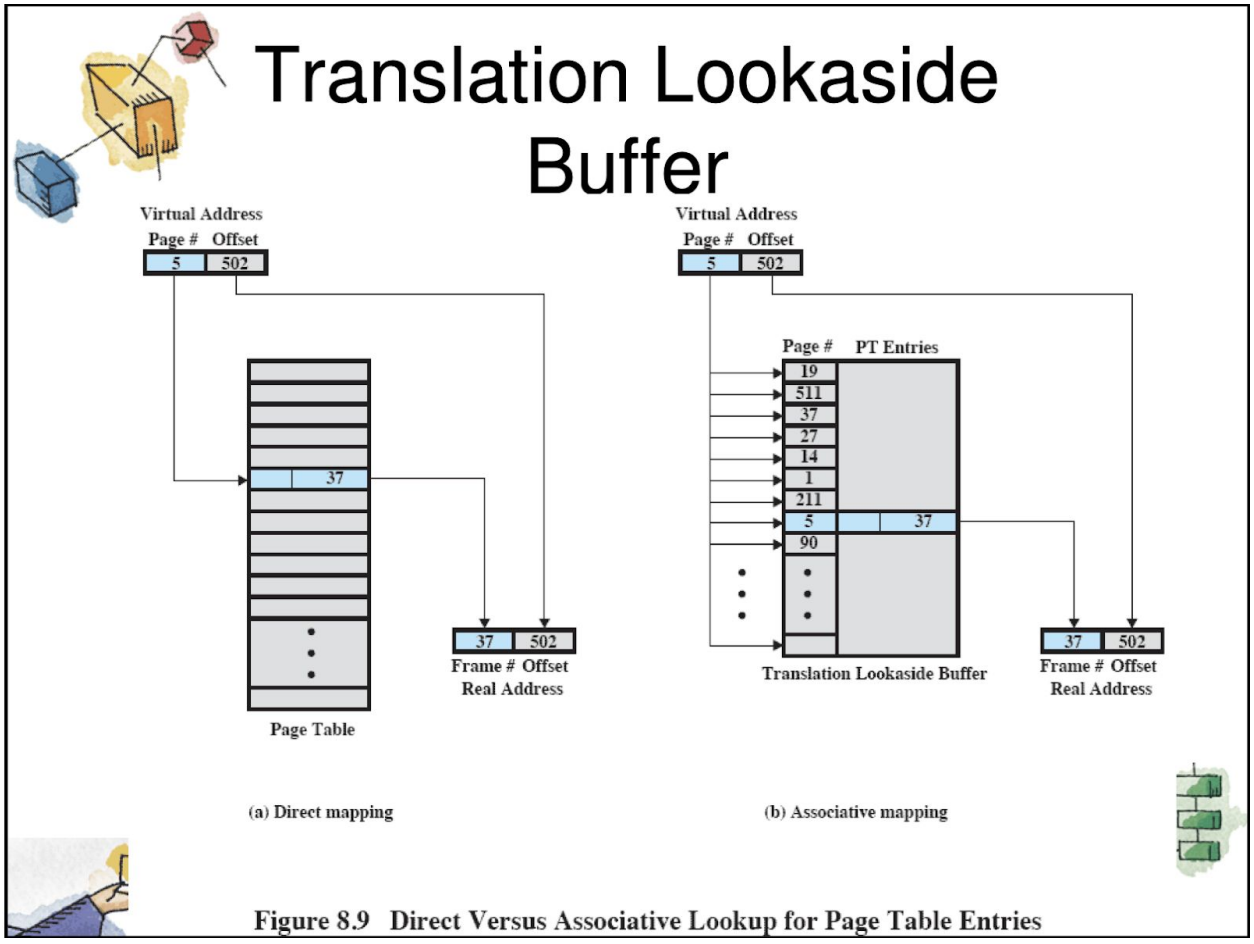

Figure 8.9 Direct Versus Associative Lookup for Page Table Entries

Steps 2 and 3 describe an associative mapping and direct mapping, respectively. The Sudo Code. Though the TLB is not a complete block of data and this cannot be simply indexed, the

process is able to simultaneously query numerous entries to find a match. In contrast, the direct mapping directly indexes the page table as the entries are ordered.

Execution of a Fault Process

- Operating System loads into main memory a few pieces of the program. The portion of the process in main memory is defined as a residence set.
- When a required address to execute the process is not found in main memory, an interrupt signal is generated and places the process in a Blocking State
- Piece of process that contains the logical address is loaded into memory
- Operating System issues a disk I/O read request
- Another process is dispatched to run while the read request is fulfilled.
- Once disk I/O is complete, an interrupt is generated and the affected process is placed in the Ready State

Residents Set Size - is a metric which reports the amount of virtual memory that is currently marked as resident in RAM. In other words, it is the portion of a process that is currently in main memory.

What are the Pros and Cons of a large residence set?

Pros

- Decreases the number of page faults, which ultimately allows the CPU to execute instructions and not waste time on swapping. This helps avoid thrashing.

Cons

- In a fixed partition allocation, a large resident set will likely result in internal fragmentation, leading to inefficient RAM utilization.
- A Large residency set invariably necessitates more RAM allocation which decreases the number of processes that can reside in main memory at once.

Small Number of Pages Per Process -> The Bigger the Number of Processes that can reside in memory. However, this increases the number of page faults.

Fixed Allocation - This gives a certain process a fixed and predetermined number of pages within which to execute.

Variable Allocation - # of pages allocated to a process vary over the lifetime of the process.

Internal vs External Fragmentation

Internal Fragmentation - occurs when more memory is allocated than what is needed for a process. Equivalently, the block of data loaded is smaller than the partition size, which results in wasted excess memory.

External Fragmentation - The main memory forms holes between portions of allocated memory that are too small to hold any process. It's the downside of storage allocation algorithms, when contiguous blocks of unused spaces cannot serve a new request because the spaces are too small for large memory application needs. In simple terms, the non-contiguous blocks create holes in the memory resulting in unused storage that are outside the allocated regions, meaning it cannot be used along with the main memory for larger memory tasks. They end up being isolated and cannot be totally eliminated from the memory space.

What Are The Pros and Cons of Having a Large Residence Set?

Thrashing

Occurs when processes on the system require more memory than what it has. If processes do not have enough pages, the page fault rate is high which leads to:

- Low CPU utilization
- Operating System spends most of its time swapping to disk

Linear - to - Physical Address Translation

Background: A logical address specified in an instruction is first translated to a linear address by the segmenting hardware. This linear address is then translated to a physical address by the paging unit.

Steps:

1) Determine the page number that the linear address falls in
2) Compute the offset by subtracting linear address by base address of corresponding page number
3) Determine frame # where the page currently resides.
4) Physical Address = Offset + Base Page Frame #

Example.
Linear Memory Address = 12287.

1) This address falls at the highest boundary of Virtual Page #2.
2) Note that Linear Address = Base Address + Offset, therefore Offset = 12287 - 8193 = 4095.

3) Inspecting the memory snapshot, the Virtual Page #2 corresponds to Page Frame #6, whose base address is 24576.
4) Therefore :

Physical Address = Offset + Base Page Frame # = 4095 + 24576 = 28671

Physical - to - Linear Address Translation

Background: A logical address specified in an instruction is first translated to a linear address by the segmenting hardware. This linear address is then translated to a physical address by the paging unit.

Steps:

5) Determine the Page Frame # the physical address falls into
6) Compute the offset from base address of page frame,
    a) Offset = Physical Address - Page Frame Base
7) Map the Page Frame # to its corresponding Virtual Frame #
8) Linear Address = Offset + Base Virtual Page Address

Example.

Physical Memory Address = 28671.

5) This address falls within range of Frame # 6, with base address equal to 24576
6) Note that Physical Address = Offset + Base Page Frame #, therefore Offset = 28671 - 24576 = 4095.
7) Inspecting the memory snapshot, the Frame # 6 corresponds to Virtual Page #2, whose base address is 8192.
8) Therefore :

Linear Address = Offset + Base Virtual Page Address = 4095 + 8192 = 12,287

Virtual addresses are used by an application program. They consist of a 16-bit selector and a 32-bit offset. In the flat memory model, the selectors are preloaded into segment registers CS, DS, SS, and ES, which all refer to the same linear address. They need not be considered by the application. Addresses are simply 32-bit near pointers.

Linear addresses are calculated from virtual addresses by segment translation. The base of the segment referred to by the selector is added to the virtual offset, giving a 32-bit linear address. Under RTTarget-32, virtual offsets are equal to linear addresses since the base of all code and data segments is 0.

Physical addresses are calculated from linear addresses through paging. The linear address is used as an index into the Page Table where the CPU locates the corresponding physical address. If paging is not enabled, linear addresses are always equal to physical addresses. Under RTTarget-32, linear addresses are equal to physical addresses except for remapped RAM regions

Frame - Fixed length block of main memory.

Page - Fixed length block of secondary memory (ie. disks)

Segment - Variable-length block of data that resides in secondary memory.

Understanding Ext2 Filesystems

An Ext2 Filesystem partition begins with boot sector followed by a series of block groups in range [0, N] on disk with equal size defined by,
N = (Disk_Size / Block_Size) - 1
The -1 in the above equation is necessary since our blocks starts at 0.

Each Block Group contains the following sections:

1. SuperBlock - Size and Shape of filesystem
2. Group Descriptor Table - tells us the location of the block/inode bitmaps and of the inode table
3. Data Block Bitmap - identifies the free blocks within the group
4. Inode Bitmap - identifies free inodes inside the group
5. Inode Table - consist of a series of consecutive blocks, each which contain a predefined number of inodes. keep track of every file; their location, size, type and access rights are all stored in inode
6. Data Blocks - store the various files' content, including directory listing, extended attributes, symbolic links, etc.

Inodes

In Unix based operating system each file is indexed by an **Inode**. Inode are special disk blocks they are created when the file system is created. The number of Inode limits the total number of files/directories that can be stored in the file system.

The Inode contains the following information:

- Administrative information (permissions, timestamps, etc).
- A number of direct blocks (typically 12) that contains to the first 12 blocks of the files.
- A single indirect pointer that points to a disk block which in turn is used as an index block, if the file is too big to be indexed entirely by the direct blocks.

- A double indirect pointer that points to a disk block which is a collection of pointers to disk blocks which are index blocks, used if the file is too big to be indexed by the direct and single indirect blocks.
- A triple indirect pointer that points to an index block of index blocks of index blocks

The root directory is always the second entry of the inode table.. Any subdirectory from there can be located by looking at the content of the root directory file.

Directory

size	start	end	Directory Entry
4	1	4	Inode number
2	5	6	This directory entry's length
1	7	7	File name length
1	8	8	File type (1=regular file 2=directory)
	9	?	File name

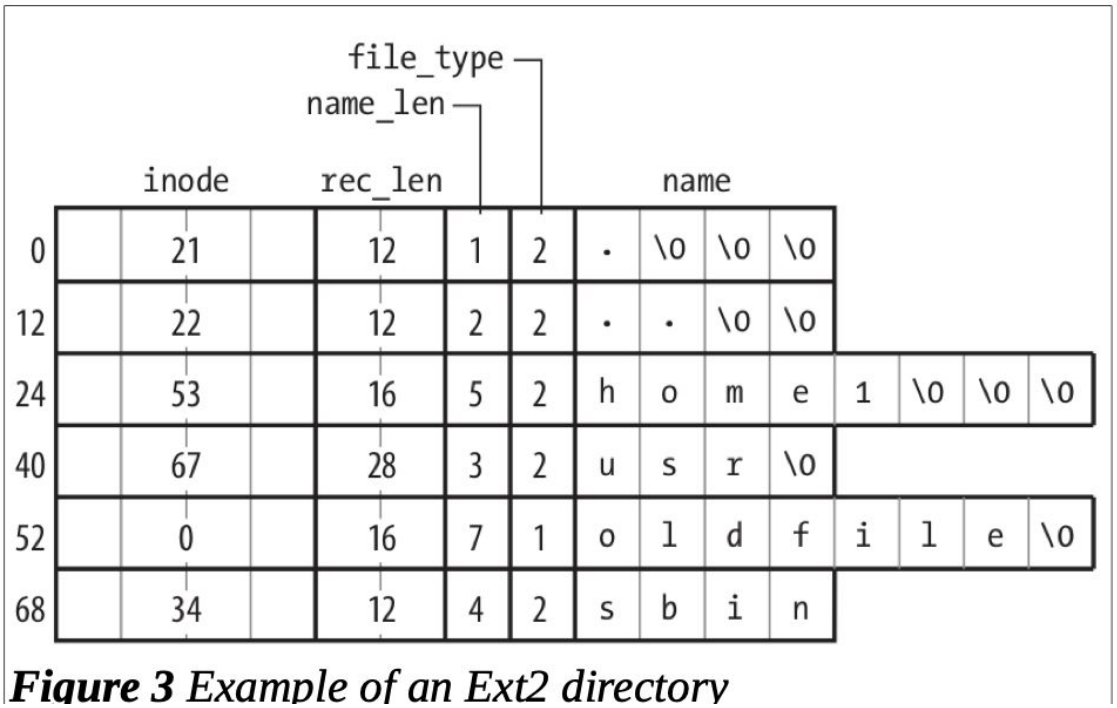

***Figure 3** Example of an Ext2 directory*

How To Find a File

1. Read the Superblock to find the size of each block, the number of blocks per group, number Inodes per group, and the starting block of the first group (Block Group Descriptor Table).
2. Determine which block group the inode belongs to.
3. Read the Block Group Descriptor corresponding to the Block Group which contains the inode to be looked up.
4. From the Block Group Descriptor, extract the location of the block group's inode table.
5. Determine the index of the inode in the inode table.
6. Index the inode table (taking into account non-standard inode size).

Directory entry information and file contents are located within the data blocks that the Inode points to.

How To Read the Root Directory

The root directory's inode is defined to always be 2. Read/parse the contents of inode 2.

How many Files can be stored in a file system? This is determined by the number of inodes.

Virtual Memory Page Replacement Algorithms

Page Fault - Occurs when (1) a program requires a memory page that has not been loaded to memory and (2) primary memory allocation has reached its full capacity, making a replacement necessary.

First In First Out (FIFO): Round Robin Style. Replaces the page that has been in memory the longest.

Least Recently Used (LRU): selects for replacement the page in memory that has not been referenced for the longest. By the Principle of Locality, this should be the page less likely to be referenced in the future.

Optimal: selects for replacement the page for which the time to the next reference is the longest. It is impossible to implement as the OS would require accurate future knowledge of events.

Clock:

Virtual Memory/Memory Exhaustion

$ free	**total**	**used**	**free**	**shared**	**buff/cache:**	**Available**
Mem:	**3951**	**601**	**1912**	**72**	**1438**	**2971**
Swap:	**4095**	**0**	**3978**			
$						

Slows down when all RAM has been used up.
It terminates when both RAM and SWAP memory have been used up.

The general pattern is that the amount of physical memory available to the program is as much as is installed in the box less how much of it is already in use (by the operating system and maybe other things). Then beyond that, an amount

of "pseudo-memory," apparent to the program as if physical and called swap space or virtual memory, is also available. When the program has consumed all the available physical memory and begins to consume swap, it slows down (because using swap is using the disk). When it consumes all of the box's available swap space, the operating system gracefully terminates it. Historically overconsuming memory resulted in a system crash requiring a reboot.