Assignment 3

By: Ian Edington
Student ID: 3236986
Date: July 31th, 2017

1. What are the advantages of using dynamic loading?

Loading is the process of moving a program from disk into memory. Loading is the last process a program undergoes before becoming a processes and being ready for execution. Prior to dynamic loading, once a program was slated for execution all the libraries would be linked and the resulting binary would be statically loaded into memory. This process was inflexible because it required the entire program be loaded into memory, when the majority of any program is often untouched during execution.

Dynamic loading solved this process by only loading instructions when they are needed. Dynamic loading relies on dynamic linking, which is the process of delaying the linking of libraries and code to runtime. By loading and linking programs dynamically a process benefits from faster startup times, reduced memory due to libraries being liked rather than copied, programs are not restricted by how much memory is available to them, and processes can access library updates without recompiling.

Dynamic loading requires the hardware support of virtual memory management.

2. Explain the basic method for implementing paging.

Paging is the process of mapping the CPUs view of logical memory to a systems physical memory.

A layer exists between the logical memory and physical memory that allows the redirection of a CPU address to a physical address. Logical memory in this scheme is called virtual memory due to the CPU having a virtualized view of the memory space. This process is orchestrated by the MMU.

Both physical and virtual memory is broken into a set of blocks of equal size. Each block of physical memory is called a frame, and each block of virtual memory is called a page. Each process has a virtual view of it's address from 0 to SIZE for the memory allocated to it. The page table in the PCB hold the block of the process. Each page block in the process page table, may or may not be mapped to a physical memory frame.

When a CPU tries to access a memory address, the page table is used to translate from the virtual address space to the physical address space.

This process is too slow to implement in software so hardware support is needed in the form of a TLB. When a page table has a page that is not linked to a frame, the system pager needs to be called in order to map a new frame for the requested page.

3. Briefly describe the segmentation memory management scheme. How does it differ from the paging memory management scheme in terms of the user's view of memory?

Before segmentation existed an older method called continual allocation was the norm. In this scheme one large block of memory was given to a process in order to meet all the needs of that process. This method was inefficient due to the external fragmentation caused by allocating large blocks of memory. It was also inflexible in that once a block of memory was allocated, there was no guarantee that the block could be expanded.

In order to allow more flexibility and reduce the impact of external fragmentation multiple segments of memory were given to process instead of large blocks. This scheme used the concept that programmers don't think in terms of memory as an array of addresses, but instead as chunks. Allowing a process to request multiple smaller blocks of memory, it helped to reduce the amount of external fragmentation and gave processes the ability to request memory as they needed it instead of all up front. Instead of a process using an address to access a piece of memory, each request contained both a segment and an address. Hardware acceleration in the form of an MMU was required in order to translate these segments and address into a physical address.

4. Explain the distinction between a demand-paging system and a paging system with swapping.

Paging is the process of virtualizing the memory that a process has access too. In many cases it allows an operating system to allocate more memory than is actually available on a system. This can provide large increases in processing capacity since the majority of the memory a process allocates is never used. Demand paging and paging with swapping are two ways of allowing an OS to over promise it's available memory while maintaining access to that memory.

Demand-paging happens when a process starts. Instead of brining all the process' memory pages into physical memory at the start of a process, demand paging allows a process to start executing without the required memory. Each time a process requests a page that doesn't have a backing frame, a page fault occurs and the frame is made available to the process. In this way only the pages that a process requests are ever loaded into physical memory.

When an OS is getting low on memory, it needs to reclaim frames in order to allow active processes using a demand paging scheme to continue executing. Page swapping is the process of taking a page and moving it to a backing store, in order to free the frame. The best way to decide which page to take is to determine which page hasn't been used in the longest amount of time (LRU). When a page isn't used for a while and the OS moves that page to disk it frees the frame to be used by a page that is needed for execution.

Both systems are complimentary and are often used together in order to reduce the physical memory

5. How does the second-chance algorithm for page replacement differ from the FIFO page replacement algorithm?

The second-chance algorithm is a suboptimal algorithm that attempts to be more efficient than a FIFO algorithm without the computational complexity of implementing Last Recently Used (LRU).

In a traditional FIFO queue, only the order of the page replacement is taken into account. This can result in recently used pages being reclaimed.

The second-chance uses a single access bit per page. Every time the page is accessed the access bit is set in the frame. When a frame needs to be reclaimed by the system the following happens:

- In order to find a suitable page to drop, the pager loops through the list of pages in FIFO order.
- If the second-chance access bit is set, the pager resets the bit and continues.
- if the second-chance access bit is not set, the pager removes that page from the page table and uses it. In this way the access bit acts as a naive LRU on bulk and only taking into account the last period.

The second chance algorithm differs from the FIFO page replacement algorithm in that there is an additional factor in deciding if the frame should be reclaimed. This is a small move towards a LRU algorithm that is also very fast to maintain.

6. Explain how copy-on-write operates.

Paging allows a number of memory optimization techniques. One of these techniques is mapping a single page into multiple processes. This technique allows two or more processes to access the same physical address, and is especially useful for libraries and processes from the same program. However, these blocks of memory need to be read only so that one process can't write to the memory and cause another process to have unexpected results. When a process is forked a large amount of both read only and read-write memory is duplicated. Copy-on-write (COW) was developed so that both the read only and the read-write memory could be copied.

Pages can be marked as copy on write. This allows new processes to be created without copying private memory block. If either process edits the block a copy will be made before editing. This is usually stored in a bit from the page table address. Only modifiable blocks need this. Read only blocks can just use the original read-write bit.

7. If you were creating an operating system to handle files, what are the six basic file operations that you should implement?

Most file systems have a large number of function that need to be implemented in order to provide all the required file system functions that are expected from a modern operating system. However, all the required operations can be implemented using six basic file operations.

1. Creating a file - Before this operation happens a suitable quantity of memory needs to be found in the disk. Depending on the type of File System Organization this might be a continuous block of memory, a linked list of blocks, or an indexed array of block. Once the memory is found this operation creates a file in a directory structure and allocate the memory on the disk for that file.
2. Writing a file - The process of writing to a file varies greatly depending on the File System Organization. In general the file name and write location is used to find the file in the directory. The physical location of the write is determined based on how the file pointer is translated to the physical location of the disk. Once the location is determined, the write is performed and the pointer is updated.

3. Reading a file - A read system call usually takes the name of the file and the amount of data to be read. Using the file pointer and the virtual file system layer the block to be read is found and requested from the disk
4. Repositioning within a file - Takes a file name and a position. Generally a pointer is kept in the operating system, and read/write operations use that pointer to access the file. However, often the location to be read or written to needs to be updated. This operation finds the file and updates the pointer location in the open file table of the PCB.
5. Deleting a file - Takes a file name. This process finds the file using the directory structure and based on the file system organization deallocates the blocks associated with that file. The file is then removed from the directory structure. This process usually leave the file data intact which can lead to a data breach.
6. Truncating - Takes a file name. Truncation is the process of deleting all the content of the file but leaving the directory listing. This is accomplished by following the directory to the file, and like with deletion deallocating the memory. However, unlike deletion the file is not removed from the directory.

8. To create a new file, an application program calls on the logical file system. Describe the steps the logical file system takes to create a file.

A OSs file system is often very complicated an has many layers of abstraction. Each layers is responsible for a different transformation in order to go from a single unified view of a file system to many separate devices with potentially different file systems.

The application calls the OS syscall using the virtual path to the file. The syscall is part of the file-system interface that the operating system exposes to a user program. From the file system interface the OS passes the file name to the Virtual File System. The Virtual File System uses the path to look up the mounted device in the device mount table. The device mount table contains a map of mount paths to devices. The mount table finds the longest matching file prefix and returns that device, along with the remaining file name. Using an OOP like abstraction the Virtual File System is then able to call the create command in the devices logical file system using the remaining path. Using the file path, the logical file system looks up the file in the directory structure for that device. The logical file system identifies the blocks that need to be modified in order to create and allocate the file and sends the write command to the Basic File System. The basic file system uses it's knowledge of the block to send detailed instructions to the device driver. The device driver with intimate knowledge of the device, sets the bits in the controller to write the blocks to the device. The controller initiates the IO transfer with the device. Once the controller has finished it sends an interrupt to the CPU, which then updates the open file list for the system and the process.

Although very complicated, all these layers of abstraction make it possible to have a unified view of a systems files, while having vastly differing underlying architecture.

9. How is a hash table superior to a simple linear list structure? What issue must be handled by hash table implementation?

A linear list is often used by a file system in order to implement a directory structure. This is an extremely easy and compact way of creating a list of files. However, it makes it difficult to find a file, since it requires a

linear search which is proportional to the number of files in the list. By implementing a sorted list using a balanced binary tree it is possible to greatly reduce the search time for a particular file. However, because of the frequency of file lookups a system generally does, the cost of maintaining and using the binary tree still isn't fast enough.

In order to increase access time many OSs use a hash table for directories. By maintaining a hash table the average lookup speeds are greatly decreased and often approach constant time. However, the effectiveness of a hash table relies on the hashing algorithm and the size of the table. When two file have the same hash code a collision occurs. Many different collision handling strategies are possible, however, the most common one for directory structures is chained entries. In a chained entry scheme each entry has a link to another entry. If the link is empty than it means there is only one entry with that hash code. However, when more than one file hashes to the same hash code multiple entries will be linked together. In the wost case, all entries hash to the same hash code and are linked together as a singly linked list. This results in a linear runtime which is as bad as the list scheme. However, this scenario is extremely improbably when hash functions and table sizes are appropriate for the data being stored.

10. What are the factors influencing the selection of a disk-scheduling algorithm?

Disk drives are the main secondary storage unit for computers. They are made up of magnetic disk with a small head that can read from and write to the disk. Disks are very slow because of the amount of time it takes for the head to move from one cylinder of the disk to another and the time it takes the disk to spin. Because of the large latency in disk operations, it is often worth it for many CPU cycles to be used in order to increase the speed on one IO operation.

In order to optimize the retrieval speed a number of algorithms have been developed. The Shortest seek time first (SSTF) and the C-LOOK algorithms are generally the most effective algorithms. With the C-LOOK algorithm the head looks to see if there is a block in the direction it is currently going. If so it goes to that one next, if not it find the furthest block in the other direction and goes to that block.

Either SSTF or LOOK is a reasonable choice for the default algorithm. These are the main factors affect the selection of a disk-scheduling algorithm:

- The number of IO operations in the queue.
- If the allocation method is Continuous or linked/indexed.
- Where the directory, file attributes, and index blocks are in relation to the file blocks.
- Sometimes priority (page caching is more important than file IO) C-LOOK tends to be better when there is a large number of IO operations and with continuous allocation. SSTF tends to be better with smaller queues.

11. Explain the disadvantage(s) of the SSTF scheduling algorithm.

The Shortest seek time first (SSTF) is one of the most effective algorithms for disk operations, and is one of the two suggested algorithms suggested by the text. The SSTF algorithm uses a priority queue to organize IO block requests based on their current seek distance from the disk head. At any given time the disk will go to the next closest block.

Because the SSTF only looks at the next closest IO block to retrieve it does not consider the optimal path but instead the next optimal step. This can cause the SSTF to take a sup-optimal path through the queue, especially when the closest points will have to be passed over twice to get first. For example with points 5,2,9 if the head is at 4 the order will be 4->5, 5->2, 2->9 with a total seek distance of 11, where an optimal path would be 4->2, 2->5, 5->9 with a total seek distance of 9.

Because the SSTF uses a priority queue it suffers from possible starvation of processes waiting in the queue. This scenario can only occur when the disk queue is never empty, so enough requests need to be occurring to keep the disk busy. Starvation will occur when many IO requests happen on one side of the disk an keep the head on that part of the disk. Since the other side of the disk is always the furthest from the current head position it will always be lower in priority. The larger the number of IO requests the more likely this scenario is to happen. It is possible to add an aging factor to this algorithm to prevent starvation.

12. Explain the concepts of a bus and a daisy chain. Indicate how these concepts are related.

Both a bus and a daisy chain are sets of wires that transport data from one component to another.

A bus is a set of parallel wires that connect hardware devices. Generally there will be a control bus, an address bus, and a data bus. A single controller usually sends instructions over the control bus and an address in the address bus. This signals the targeted device to read from or write to the data bus. It is however possible to have a bus that uses serial communication though, like PCI express, and therefore doesn't need a separate control, address, and data bus. With the serial bus, all the same sections exist, however, instead of being sent in parallel they are serialized, sent over the bus, and deserialized at the other side.

A daisy chain is the concept of stringing together devices. Each device has at least two port and a cable travels from one device to another in a line. Usually a daisy chain uses a bus to communicate between all the devices on the chain, and again one controller is used to send commands.

13. What are the three reasons that buffering is performed?

A buffer is a section of high speed memory that holds data when it is between two entities (devices, processes, people, ect.). The three reasons for buffers are as follows:

1. Speed mismatch between producer and consumer - When one device is really slow and the other device is really fast a buffer is used in order to not waste the fast devices time with small intermittent interrupts. For example if a disk drive had two buffer, it would take a stream of requests and fill the first buffer. It would signal the CPU, and the CPU would move all the requests from the first buffer into memory. While the CPU was doing this the disk would start filling the second buffer. In this way the CPU is only interrupted once instead of every time there is a block of data available.

2. Data block size mismatch between producer and consumer - When a producer and consumer require data to be transfered in different sizes a buffer can be used in order to hold the data on the larger side until it is the right size for the operation. For example one process accepts each persons resume as they become available, and another process looks at 10 resumes at a time to find the top candidate.

The producer can add one resume at a time to a buffer, and the consumer can wait until there are 10 resumes in the buffer before it looks at any of them.

3. Support "copy semantics" for application I/O - Copy semantics guarantees that an I/O operation performed on memory will affect the memory at the time of the operation. If a write to memory occurs after the I/O operation it will not affect what is written to disk. The memory for the operation is copied into a buffer that waits for the I/O device to be ready. In this way there's a separation of the memory from the IO operation.