Boris Jurosevic

Homework Group 3

CS 3100

Chapter 8 NoteTaker (Please use a DIFFERENT COLOR in your responses)

1. **Definitions**
   1. Backing Store - fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
   2. Roll in – Roll out - swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.
   3. Contiguous Memory Allocation - is a classical memory allocation model that assigns a process consecutive memory blocks (that is, memory blocks having consecutive addresses). Main memory usually into two partitions: Resident operating system, usually held in low memory with interrupt vector. User processes then held in high memory.
   4. MMU - is a computer hardware component responsible for handling accesses to memory requested by the CPU. Its functions include translation of virtual addresses to physical addresses (i.e., virtual memory management), memory protection, cache control, bus arbitration and, in simpler computer architectures (especially 8-bit systems), bank switching.
   5. Internal Fragmentation - Main memory use is inefficient. Any program, no matter how small, occupies an entire partition.
   6. External Fragmentation - External fragmentation arises when free memory is separated into small blocks and is interspersed by allocated memory. It is a weakness of certain storage allocation algorithms, when they fail to order memory used by programs efficiently. The result is that, although free storage is available, it is effectively unusable because it is divided into pieces that are too small individually to satisfy the demands of the application. The term "external" refers to the fact that the unusable storage is outside the allocated regions. Eventually get holes in the memory.
2. **Memory Management Requirements**
   1. Relocation
      1. Programmer does not know where the program will be placed in memory when it is executed because…
      2. While the program is executing, it may be swapped out to disk and returned to main memory at a different location
      3. Logical Memory references in the code must be translated by the OS to physical memory addresses.

* 1. Protection

* + 1. Processes should not be able to reference memory locations in another process without permission
    3. Therefore, it’s impossible to check absolute addresses in programs since the program could be relocated at any time

So, memory must be checked during runtime by the Processor (Hardware).

* + - * + Operating system cannot anticipate all of the memory references a program will make.

* 1. Sharing
     1. Allow several processes to access the same portion of memory
     2. Better to allow each process (person) access to the same copy of the program rather than have their own separate copy

* 1. Logical Organization
     1. Memory is Linear, but
     2. Programs are written in modules
     3. Advantages achieved:
     4. Modules can be shared among processes
     5. Modules can be written and compiled independently – resolving memory references at runtime
     6. Different degrees of protection given to modules (read-only, execute-only)

* 1. Physical Organization
     1. Memory available for a program plus its data may be insufficient
     2. Programmer does not know how much space will be available
     3. Overlaying allows various modules to be assigned the same region of memory
     4. Moving seamlessly between memory locations is the key to Effective Memory Management

1. **Physically allocating processes to Memory**
   1. Dynamic Loading
      1. Routine is not loaded until it is called
      2. Better memory-space utilization; unused routines are never loaded.
      3. Useful when large amounts of code are needed to handle infrequently occurring cases.
      4. No special support from the operating system is required; implemented through program design.

* 1. Dynamic Linking Libraries
     1. Linking postponed until execution time.
     2. Small piece of code, called a *stub*, is used to locate the appropriate memory-resident library routine.
     3. Stub replaces itself with the address of the routine, and executes the routine.
     4. Operating system needed to check if routine is in processes’ memory address.
     5. Libraries can be easily updated.

* 1. Overlays
     1. Keep in memory only those instructions and data that are needed at any given time.
     2. Needed when process is larger than amount of memory allocated to it.
     3. Implemented by user, no special support needed from operating system, programming design of overlay structure is complex.

* 1. Swapping
     1. A process can be *swapped* temporarily out of memory to a *backing store*, and then brought back into memory for continued execution.
     2. Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
     3. Must be fast enough to keep up with CPU Scheduler demands.

1. **Fixed Partitioning**
   1. Definition: any process whose size is less than or equal to the partition size can be loaded into an available partition if all partitions are full, the operating system can swap a process out of a partition because all partitions are of equal size, it does not matter which partition is used.
   2. Disadvantage: Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. Also a program may not fit in a partition. The programmer must design the program with overlays.
   3. Lessened (Not Solved by): Unequal-size partitions can assign each process to the smallest partition within which it will fit. There is a queue for each partition processes are assigned in such a way as to minimize wasted memory within a partition.
2. **Dynamic Partitioning**
   1. Definition: Partitions are of variable length and number. Process is allocated exactly as much memory as required. Must use compaction to shift processes so they are contiguous and all free memory is in one block.
   2. Algorithms:
      1. First-fit algorithm
         1. Searches from the beginning of the memory block
         2. Fastest algorithm, but
         3. May have many processes loaded in the front end of memory that must be searched over when trying to find a free block

* + 1. Best-fit algorithm
       1. Chooses the block that is closest in size to the request
       2. Since smallest block is found for process, the smallest amount of fragmentation is left, but memory compaction must be done more often

* + 1. Worse-fit algorithm

* + - 1. Scans memory from the last known location through the end and allocates the largest hole
      2. More often allocates a block of memory at the end of memory where the largest block is found
      3. The largest block of memory is eventually broken up into smaller blocks
      4. Compaction is required to obtain a large block at the end of memory

1. **Paging**
   1. Partition memory into small equal-size chunks and divide each process into the same size chunks
   2. The chunks of a process are called pages and chunks of memory are called frames
   3. Operating system maintains a page table for each process
   4. contains the frame location for each page in the process
   5. memory address consist of a page number and offset within the page

1. **Segmentation**
   1. Supports a user view of memory
   2. All segments of all programs do not have to be of the same length
   3. There is a maximum segment length
   4. Addressing consists of two parts - a segment number and an offset
   5. Since segments are not equal, segmentation is similar to dynamic partitioning

Chapter 9 NoteTaker

1. Definitions
   * Resident Set - portion of process that is in main memory
   * Principle of Locality - Program and data references within a process tend to cluster together on the hard drive. Only a few pieces of a process will be needed over a short period of time. Possible to make intelligent guesses about which pieces will be needed in the future
   * Belady’s Anomaly - is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern. This phenomenon is commonly experienced when using the First in First Out (FIFO).

1. Translation Lookaside Buffer
   * Each virtual memory reference can cause two physical memory accessesone to fetch the page table, one to fetch the data, To overcome this problem a high-speed cache is set up for page table entries
   * Contains page table entries that have been most recently used
   * Functions same way as a memory cache
   * First checks if page is already in main memory
   * if not in main memory a page fault is issued
   * The TLB is updated to include the new page entry
2. Demand Paging
   * Bring a page into memory only when it is needed.
   * Advantages
     1. Less I/O needed, Less memory needed
     2. Faster response, Supports More processes/users
   * Valid – Invalid Bit
     1. (1 ⇒ in-main memory, 0 ⇒ not-in-main memory)
   * Demand Paging works because of the principle of locality.
3. Page Size
   * Disadvantages of smaller page sizes
     1. Smaller page size = less internal fragmentation
     2. But, more pages are required per process
     3. More pages per process means larger page tables
     4. Larger page tables means large portion of page tables are stored in virtual memory
   * Disadvantages of larger page sizes
     1. Larger page size, more of the process will be found in main memory
     2. The pages in memory will all contain portions of the process near recent references (other portions of the program or process). Page faults low.
     3. BUT as time goes on during execution, Increased page size causes pages to contain locations further from any recent reference. Page faults rise.
   * Why have variable page sizes?
     1. Multiple page sizes provide the flexibility needed to effectively use a TLB
     2. Large pages can be used for program instructions
     3. Small pages can be used for threads
     4. However, most operating system support only one page size (typically 4kb).

1. Dealing with Large Page Table
   * Multi-Level Page Tables
     1. Page the page table
     2. Modern processors have a 3 or 4 level page table
   * Inverted Page Tables

* + 1. Map the physical frames into logical pages
    2. Relies heavily on the TLB

1. Thrashing
   * Definition: a process is spending more time swapping pages than executing
   * Solution: : Limit thrashing by using a local replacement algorithm
   * Prevent over-allocation of memory (too much multiprogramming) by modifying page-fault service routine to include page replacement.
   * Use *modify* (*dirty*) *bit* to reduce overhead of page transfers – only modified pages are written to disk.
2. Page Replacement Algorithms
   * Optimal
     1. Replace page that will not be used for longest period of time.
     2. 4 frames example 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

* + FIFO
    1. 3 frames (3 pages can be in memory at a time per process)

* + LRU
    1. Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
    2. Counter implementation. Every page entry has a counter; every time the page is referenced through this entry, the clock is copied into the counter. The page to be replaced is the one that has the smallest (LRU) value.
  + Counting Algorithms
    1. Keep a counter of the number of references that have been made to each page.
    2. LFU (Least Frequently Used) Algorithm: replace the page with the smallest count.
    3. MFU (Most Frequently Used) Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

1. Determining how many frames to allocate to a process
   * 1. Local replacement – each process selects from only its own set of allocated frames.
     2. Equal allocation – e.g., if 100 frames and 5 processes, give each 20 pages. - Remainder goes to Free Frame Buffer
     3. Proportional allocation – Allocate according to the size of process.

1. Explain the difference between internal and external fragmentation.

Internal fragmentation occurs when more storage is allocated than is actually requested. This left over space, known as slack space, causes a degradation of system performance. For example, space can be left over between the last byte of the file and the first byte of the next sector. This slack space can add up and eventually cause system performance to degrade.

External fragmentation occurs when free memory becomes divided into several small blocks over time. For example, this can cause problems when an application requests a block of memory of 1000bytes, but the largest contiguous block of memory is only 300bytes. Even if ten blocks of 300 bytes are found, the allocation request will fail because they are not contiguous. This can be caused by an excessive amount of clutter on a system.

So we can say that internal fragmentation is the area in a region or a page that is not used by the job occupying that region or page. This space is unavailable for use by the system until that job is finished and the page or region is released. The main difference is the allocation: internal fragmentation is allocated area and unused but external fragmentation is un allocated area and unused.

2. Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)?

First-fit algorithm

100KB -- 500KB – 212KB, 112KB

200KB – 300KB –

600KB – 417KB

process unallocated would be - 426KB .

unused memory would be -959KB.

Best-fit algorithm

100KB -- 500KB – 417KB

200KB – 112KB 300KB – 212KB

600KB – 426KB

memory unused would be -533KB

Worst-fit algorithm

100KB -- 500KB – 417KB

200KB – 300KB –

600KB – 212KB, 112KB

process unallocated would **-**426KB

unused memory would be -959KB

So we can say that:

Best-fit would be the best because there is no wait processes.

Which algorithm makes the most efficient use of memory?

3. Compare the main memory organization schemes of contiguous-memory allocation, pure segmentation, and pure paging with respect to the following issues:

contiguous-memory allocation

a. external fragmentation

There is external fragmentation (as address spaces are allocated contiguously and holes develop as finished processes release its space and new processes are allocated and the size of the new process is almost smaller than the old one)

b. internal fragmentation

There is no internal fragmentation

c. ability to share code across processes

It does not allow processes to share code.

pure paging

a. external fragmentation

There is no external fragmentation

b. internal fragmentation

There is internal fragmentation (it appears in the last frame because the process size almost not a multiplex of page size )

c. ability to share code across processes

Able to share code between processes

pure segmentation

a. external fragmentation

There is external fragmentation (fragmentation would occur as segments of finished processes are replaced by segments of new processes.

and the size of the new process is almost smaller than the old one)

b. internal fragmentation

There is no internal fragmentation

c. ability to share code across processes

Able to share code between processes

4. On a system with paging, a process cannot access memory that it does not own; why? How could the operating system allow access to other memory? Why should it or should it not?

An address on a paging system is a logical page number and an offset. The physical page is found by searching a table based on the logical page number to produce a physical page number. Because the operating system controls the contents of this table, it can limit a process to accessing only those physical pages allocated to the process. There is no way for a process to refer to a page it does not own because the page will not be in the page table.

To allow such access, an operating system simply needs to allow entries for non-process memory to be added to the process’s page table.

This is useful when two or more processes need to exchange data—they just read and write to the same physical addresses (which may be at varying logical addresses). This makes for very efficient interprocess communication.

It's useful also to implement sharing.

5. Compare paging with segmentation with respect to the amount of memory required by the address translation structures in order to convert virtual addresses to physical addresses.

Paging requires more memory overhead to maintain the translation structures. Segmentation requires just one (or more) entry for code and one entry for data. Paging on the other hand requires multiple entries for code and data depending on page size. So , paging almost requires more memory for page table than the memory space required to segment table.

6. A simplified view of thread states is **Ready**, **Running**, and **Blocked**, where a thread is either ready and waiting to be scheduled, is running on the processor, or is blocked (i.e. is waiting for I/O.) This is illustrated in Figure 9.31. Assuming a thread is in the Running state, answer the following questions: (Be sure to include an explanation of your answer.)

a. Will the thread change state if it incurs a page fault? If so, to what new state?

Yes, a tread changes from the Running state to the Blocked state when a page fault occurs.

When a page fault occurs, the process starts to wait for I/O operation to finish. The OS does several things while the process is waiting. It checks whether the page is really invalid or just on disk, finds a free memory frame, schedules a disk access to load the page into t the frame, updates the page table with the new logical-physical mapping, updates the valid bit of that entry, and eventually restarts the process by change its state from Blocked to Ready.

b. Will the thread change state if it generates a TLB miss that is resolved in the page table? If so, to what new state?

Not necessarily. If a page table entry is not found in the TLB (TLB miss), the page number is used to index and process the page table. If the page is already in main memory, then TLB is updated to include the new page entry, while the process execution continues since there is no I/O operation needed. If the page is not in the main memory, a page fault is generated. In this case, the process needs to change to the Blocked state and wait for I/O to access the disk. This is the same procedure as in the first question.

c. Will the thread change state if an address reference is resolved in the page table? If so, to what new state?

No, because no I/O operation is needed is the address reference is resolved in the page table, which indicates the page needed is loaded in the main memory already.

7. Discuss situations under which the least frequently used page-replacement algorithm generates fewer page faults than the least recently used page replacement algorithm. Also discuss under what circumstance the opposite holds.

Consider the following sequence of memory accesses in a system that can hold four pages in memory: 1 1 2 3 4 5 1. When page 5 is accessed, the least frequently used page-replacement algorithm would replace a page other than 1,and therefore would not incur a page fault when page 1 is accessed again. On the other hand, for the sequence “1 2 3 4 5 2,” the least recently used algorithm performs better

8. Discuss situations under which the most frequently used page-replacement algorithm generates fewer page faults than the least recently used page replacement algorithm. Also discuss under what circumstance the opposite holds.

Consider the sequence in a system that holds four pages in memory: 1 23 4 4 4 5 1. The most frequently used page replacement algorithm evicts page 4while fetching page 5, while the LRU algorithm evicts page 1. This is unlikely to happen much in practice. For the sequence “1 2 3 4 4 4 5 1,” the LRU algorithm makes the right decision.

9. What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate this problem?

Thrashing is a situation where whenever a process needs to run, it swaps out some other processes working set to disk. This happens when all the processes working set sizes sum to larger then the amount of physical memory available in the system.

Thrashing can be detected by the system when the CPU utilization starts decreasing and the number of page faults increases considerably.

To eliminate the problem, the system can either decrease the degree of multiprogramming or can use a local (or priority) replacement algorithm.