Virtual Memory

The requirement that instructions of a process need to be in physical memory is necessary and reasonable, but limits the size of the program. They contain extra space which is not needed. For example

1. Programs have handling of unusual error conditions. Since errors seldom occur, these are never executed
2. Arrays, lists and tables allocated more memory than needed
3. Certain options and features may be used rarely

The ability of a program that is only partially in memory would confer many benefits

1. A program will no longer be constrained to fit entirely in physical memory
2. More programs can be run at the same time as the program loaded will take lesser memory
3. Less I/O required to load or swap these programs

Virtual memory involves the separation of logical memory from physical memory, allowing a large virtual memory to be addressed to a smaller physical address space.

Virtual addresses that include holes are called sparse address spaces. These are useful because the holes can be filled as the stack or heap segments grow or if we wish to use DLLs.

Virtual memory also allows files and memory to be shared by two or more processes

1. System libs can be shared through mapping a shared object into a virtual address space
2. Processes can share memory by creating virtual shared address spaces
3. Pages can be shared during process creation with fork(), thus speeding process creation

Demand Paging

Demand paged virtual memory allows page to be loaded only when they are demanded during program execution. Pages never accessed as never brought into memory.

This system is similar to a paging system with swapping where processes reside in secondary memory. Rather than swapping the entire process into memory, a lazy swapper is used. A lazy swapper never swaps a page into memory unless the page is needed. This is also called a pager.

To distinguish between pages in memory and pages on disk, we use a valid-invalid bit. If the process tries to access a page not in memory, it results in a page fault. The procedure for handling page faults is

1. Check an internal table for this process to validate the reference
2. If reference is invalid, terminate the process. If valid, bring the page in from memory.
3. Find a free frame
4. Schedule a disk operation to read page into the frame
5. Modify the internal table that the page is in memory
6. Restart execution

Pure demand paging assumes no pages in memory, faulting at every unique instruction.

Programs tend to have locality of reference, and demand paging uses this technique to bring in multiple pages for the same process.

The hardware required again is just a page table and secondary memory. The secondary memory is known as the swap device and the space used for the pages is known as swap space.

Sometimes a page fault may occur in between a move instruction, when some locations are modified. This problem is solved before the move instruction

1. Microcode computes and attempts to access both ends of both blocks. If page fault occurs, it occurs now and the OS corrects it.
2. Temporary registers hold the values of overwritten locations, and are used to restore context after the fault is handled.

The effective access time of a demand page system is (1 - p) x ma + p x page fault time.

Where p is the probability of page fault and ma is the memory access time.

A page fault causes this sequence to occur

1. Trap to the OS
2. Save user registers and process state
3. Determine that the interrupt was a PF
4. Check the page reference for validity and determine page location on disk
5. Issue a disk read to a free frame
    a. Wait in queue for this device till the request is served
    b. Wait for device seek and/or latency time
    c. Begin the transfer of the page to a free frame
6. While waiting, allocate the CPU to some other user.
7. Receive interrupt that I/O is completed

8. Save the registers and process state
9. Determine that the interrupt was from disk
10. Correct the page table to show the desired page is in memory
11. Wait for CPU to be allocated to the process
12. Restore user registers, process state and new table, and resume.

An additional aspect of demand paging is the handling and use of swap space. Disk I/O to swap space is usually faster than filesystem I/O. This block is large, so the whole file image can be copied into the swap space. Another approach is to write the pages into the swap space.

Some systems limit the amount of swap space used through demand paging of binary files. When page replacement is used, they can simply be overwritten, and the pages can be read in from the filesystem again if needed. The file system hence becomes the backing store.

Copy-On-Write

In this, upon a fork() system call, the parent and the child share the same page. These shared pages are marked as copy-on-write pages, meaning if any of the processes write to this page, the page will be copied and then the write will occur. Free pages in this case are allocated based on zero-fill-on-demand, where pages have been zeroed before being allocated.

vfork() or virtual memory fork allows the child to overwrite the parent page, and thus replaces it completely.

Page Replacement

Over-allocation of memory to processes is a major problem for the degree of multiprogramming. Since the number of pages stored in memory are limited, page replacement needs to be used to swap in pages that are required and swap out unused pages.

The page fault service routine is modified to include page replacement. This is done by

1. Finding the location of the desired page on disk
2. Find a free frame
    a. If there is a free frame, use it
    b. If there is no free frame, select a victim frame to be replaced
    c. Write the victim frame to disk, change the page and page frame tables
3. Read the desired page into the newly freed frame
4. Continue user process from where the page fault occurred.

If no frames are free, two page transfers are required. This can be brought down to 1 by using a modify or a dirty bit, indicating that the page has been modified. If the page is not modified, we do not need to write it back to the memory.

To implement demand paging, we need a frame allocation and a page allocation algorithm. The algorithms are evaluated using a reference string. Reference strings can be generated by using random number generators or by tracing a given system and recording the address of each memory reference. The second takes a lot of memory, so to reduce this, we say we consider only the page number as the page size is fixed, and if we have a reference to page p, any reference to a page after p will never cause a page fault.

As the number of free frames available increases, the number of page faults decreases.

1. **FIFO**
    a. Oldest page is replaced
    b. Make a FIFO queue to hold all pages
    c. Page fault rate may increase as the number of allocated frames increases, called Belady's anomaly.
2. **Optimal**
    a. Lowest page fault rate
    b. Replace the page that will not be used for the longest amount of time
    c. Does not suffer from Belady's anomaly
    d. Difficult to implement as future is not known
3. **LRU**
    a. Replace the page that has not been used for the longest time
    b. Can be implemented in two ways
        i. Counters : Associate a time of use field and add a logical clock, to find the time of the last reference to the page, and replace the page with the smallest time value.
        ii. Stack : Most recently used page at the top, least used on the bottom.
    c. Stack replacement algorithm, does not suffer from Belady's anomaly
    d. A stack algorithm is one where it can be shown that a set of pages in memory for n frames is always a subset of the set of pages that would be in memory for n + 1 frames.
    e. Updating of stack and clock must be done at each memory reference. Causes a lot of overhead.

4. **LRU-Approximation**
    a. A reference bit is used, with each page referenced set to 1. Replace the pages with bit = 0.
    b. Additional reference bits algorithm
        i. Store a bit for each time period and replace the page with the most 0s
        ii. If there is only one bit used, it becomes a second chance algorithm
    c. Second chance algorithm
        i. If value of bit is 0, replace. If value of bit is 1, skip it, but make the bit 0 and skip, giving a second chance.
        ii. Implemented by a circular queue
        iii. Degenerates to FIFO if all bits are set.
    d. Enhanced second chance algorithm
        i. Consider reference and modify bit
        ii. (0,0) - neither recently used nor modified - best page to replace
        iii. (0,1) - not recently used but modified - page needs to be written to memory before replacement
        iv. (1,0) - recently used but not modified - probably will be used again soon
        v. (1,1) - recently used and modified
        vi. Replace the page in the lowest priority class
5. **Counting Based**
    a. Keep counter of number of references to a page
    b. Least Frequently used
        i. Replace the page with the smallest count
    c. Most Frequently used
        i. Page with smallest count was just brought in, may need to be used again
6. **Page Buffering**
    a. Once a victim frame is chosen, desired page is written to a free frame before victim is written out.
    b. After victim is written out, frame is added to the free frame pool
    c. Maintains a list of modified pages, which writes these pages to memory when CPU is idle.
    d. Another is to remember what page was in what frame. If frame is not written, it can be used directly.

Sometimes, applications outperform virtual memory based processing, especially if the application knows how to manage its own memory. Because of this, some OS give special programs the ability to use the disk partition as a large sequential array of blocks, called a raw disk. Raw I/O bypasses all file-system services.

Allocation of Frames

The basic algorithm is

1. The OS allocates all its buffers and table space from the free frame list
2. When this system is not being used, user paging is implemented
3. Free frames are kept in a free frame list
4. When PF occurs, the user process is allocated to one of these free frames.

Minimum Number of Frames

In this, the OS allocates the minimum number of frames required to service a process. This, however increases page faults.

When multiple levels of indirection exist, where every indirection access every page in virtual memory. This is countered by limiting the number of indirections, and if the limit is reached, trap occurs.

Allocation Algorithms

The easiest way to split m frames among n process is equal allocation, where each process gets m/n frames. This is not good as some process may not require that many pages, and others may require more.

To solve this, we allocate frames proportionally, according to the process size. If S is the sum of all the virtual memory space for a process, the number of frames available is m, then the allocated frames would be s / S x m.

The allocation depends on the degree of multiprogramming inversely.

Global vs Local Allocation

Page replacement algorithms can be global or local. Global replacement allows a process to select a frame from a set of all frames, even if the frame is allocated to some process. Local allocation only allows it a frame from that process' set of allocated frames.

One problem with global replacement is that a process cannot control its own page fault rate. Local replacement may hinder a process by not making many pages available to it. Therefore, global replacement is chosen usually.

Non-Uniform Memory Access

Systems in which memory access times vary depending on where the architecture exists are known as non-uniform memory access (NUMA) systems. Memory with lesser latency is chosen for higher priority processes.

Thrashing

A process is said to be thrashing if it spends most of its time paging than executing.

A program thrashes because

1. If CPU utilisation is too low, degree of multiprogramming introduces a new process into the system.
2. A global page replacement algo is used, causing page replacement of active processes
3. The new process starts faulting due to more frames being required, taking frames from other processes
4. The other processes also start faulting
5. CPU utilisation decreases further.
6. CPU detects this, and increases the degree of multiprogramming again, further reducing CPU utilisation.

The effects of thrashing can be limited by using a local replacement or a priority replacement algorithm, where if one process starts thrashing, it cannot steal frames from another process. But this still causes a decrease in CPU utilisation, increasing effective access time.

To prevent thrashing, we need to allocate enough frames to a process, by looking at how many frames the process is using, called the locality model. Frames are allocated only for locality, that is, each function is one locality, etc. Thus a process only faults for its locality.

The working set model is based on the assumption of locality. It uses a parameter del to define the working set window. The most recent del page references is the working set. If a page is in use, it belongs to this working set. If not, it will drop from the set del time units after its last reference, approximating the program's locality. Selection of del is paramount to define the locality.

The total demand for frames is the sum of all the working set sizes of all the processes. If this total demand exceeds the number of available frames, thrashing occurs.

The algorithm is summarized as

1. The OS monitors the working set of each process and allocates to that working set enough frames to provide it with the working set size.
2. If there are enough extra frames, another process can be initiated.
3. If sum of WSS increases, the OS selects a process to suspend
4. The suspended process can start later

The only difficulty with this approach is to keep track of the working set. The working set window is a moving window, where old references are dropped when new references enter.

This can be approximated using a fixed interval timer interrupt and a reference bit, where the timer interrupts at half the del value. When an interrupt occurs, we clear the reference bit values for each frame. If a PF occurs, examine the current reference bit and two in memory bits to see if the page was used in the timer interrupt window size. If it was used, one of these bits will be set, and these pages are considered in the working set. This can be made more accurate by increasing the reference history bits and frequency of interrupts. However, this increases the cost to service these interrupts as well.

Page Fault Frequency

The PFF approach establishes lower and upper bounds on the desired PF rate. When the process exceeds the upper limit, the process is allocated a new frame. If the process drops below the lower limit, we remove a frame from the process.

The Memory API

In a running program, two types of memory are offered, the stack, the memory offered by the compiler, and the heap, for dynamic memory allocation.

The malloc() call allocates memory from the heap, with the size defined by the user It internally calls the system call sbrk(), making it a compile time operator.

The free() system call frees memory based on the allocation by malloc().