 Unit-4 MEMORY MANAGEMENTMain Memory, Swapping, Contiguous Memory Allocation, Segmentation, Paging, Structure of the Page Table. Virtual Memory: Introduction, Demand Paging, Copy-on-Write, Page Replacement, Allocation of Frames, Thrashing, Memory-Mapped Files, Allocating Kernel Memory. Storage Management: Disk Structure, Disk Attachment, Disk Scheduling, Disk Management, Swap-Space Management, RAID Structure. File-System Interface: File Concept, Access Methods, Directory and Disk Structure, File-System Mounting, File Sharing, Protection Memory management

 Main MemoryMain Memory

 Main MemoryBackgroundProgram must be brought (from disk) into memory and placed within a process for it to be run Main memory and registers are only storage CPU can access directly Memory unit only sees a stream of: addresses + read requests, or address + data and write requests Register access is done in one CPU clock (or less) Main memory can take many cycles, causing a stall Cache sits between main memory and CPU registers Protection of memory required to ensure correct operation When processor accesses memory then it spends a significant amount of time waiting for the data to become available. This situation is called MEMORY STALL.

 Main MemoryProtectionNeed to ensure that a process can access only those addresses in its address space. We can provide this protection by using a pair of base and limit registers define the logical address space of a process

 Main MemoryHardware Address ProtectionCPU must check every memory access generated in user mode to be sure it is between base and limit for that user the instructions to loading the base and limit registers are privileged

 Main MemoryLogical vs. Physical Address SpaceThe concept of a logical address space that is bound to a separate physical address space is central to proper memory management Logical address – generated by the CPU; also referred to as virtual address Physical address – address seen by the memory unit Logical address space is the set of all logical addresses generated by a program Physical address space is the set of all physical addresses generated by a program

 Main MemoryMemory-Management Unit (MMU)Hardware device that at run time maps virtual to physical address

 Main MemoryMemory-Management Unit (Cont.)Consider simple scheme. which is a generalization of the base-register scheme. The base register now called relocation register The value in the relocation register is added to every address generated by a user process at the time it is sent to memory The user program deals with logical addresses; it never sees the real physical addresses

 Main MemoryDynamic Loading and Dynamic linkingThe entire program does need to be in memory to execute Routine is not loaded until it is called Better memory-space utilization; unused routine is never loaded Useful when large amounts of code are needed to handle infrequently occurring cases No special support from the operating system is required Implemented through program design OS can help by providing libraries to implement dynamic loading Static linking – system libraries and program code combined by the loader into the binary program image Dynamic linking –linking postponed until execution time

 Contiguous Memory AllocationMain Memory

 Main MemoryContiguous AllocationMain memory must support both OS and user processes Limited resource, must allocate efficiently Contiguous allocation is one early method Main memory usually into two partitions: Resident operating system, usually held in low memory with interrupt vector User processes then held in high memory Each process contained in single contiguous section of memory

 Main MemoryContiguous Allocation (Cont.)Relocation registers used to protect user processes from each other, and from changing operating-system code and data Base register contains value of smallest physical address Limit register contains range of logical addresses – each logical address must be less than the limit register MMU maps logical address dynamically Can then allow actions such as kernel code being transient and kernel changing size

 Main MemoryHardware Support for Relocation and Limit Registers

 Main MemoryVariable PartitionMultiple-partition allocation Degree of multiprogramming limited by number of partitions Variable-partition sizes for efficiency (sized to a given process’ needs) Hole – block of available memory; holes of various size are scattered throughout memory When a process arrives, it is allocated memory from a hole large enough to accommodate it Process exiting frees its partition, adjacent free partitions combined Operating system maintains information about: a) allocated partitions b) free partitions (hole)

 Main MemoryDynamic Storage-Allocation ProblemFirst-fit: Allocate the first hole that is big enough Best-fit: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size Produces the smallest leftover hole Worst-fit: Allocate the largest hole; must also search entire list Produces the largest leftover holeHow to satisfy a request of size n from a list of free holes?First-fit and best-fit better than worst-fit in terms of speed and storage utilization

 Main MemoryFragmentationProblem-01:  Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB and 250 KB. These partitions need to be allocated to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order. Perform the allocation of processes using- First Fit Algorithm Best Fit Algorithm Worst Fit Algorithm

 Main MemoryFragmentationThe main memory has been divided into fixed size partitions as- Let us say the given processes are- Process P1 = 357 KB Process P2 = 210 KB Process P3 = 468 KB Process P4 = 491 KB

 Main MemoryFragmentation Let us say the given processes are- Process P1 = 357 KB Process P2 = 210 KB Process P3 = 468 KB Process P4 = 491 KBBest Fit AlgorithmFirst Fit Algorithm – P4 can’t be allocatedWorst Fit Algorithm – P3 and P4 can’t be allocated

 Main MemoryFragmentationExternal Fragmentation – Total memory space exists to satisfy a request, but it is not contiguous Internal Fragmentation – Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

 Main MemoryFragmentation (Cont.)Reduce external fragmentation by compaction Shuffle memory contents to place all free memory together in one large block Compaction is possible only if relocation is dynamic, and is done at execution time I/O problem Latch job in memory while it is involved in I/O Do I/O only into OS buffers

 PagingMain Memory

 Main MemoryPagingPhysical address space of a process can be noncontiguous; process is allocated physical memory whenever the latter is available Avoids external fragmentation Avoids problem of varying sized memory chunks Divide physical memory into fixed-sized blocks called frames Size is power of 2, between 512 bytes and 16 Mbytes Divide logical memory into blocks of same size called pages Keep track of all free frames To run a program of size N pages, need to find N free frames and load program Set up a page table to translate logical to physical addresses Still have Internal fragmentation

 Main MemoryAddress Translation SchemeAddress generated by CPU is divided into: Page number (p) – used as an index into a page table which contains base address of each page in physical memory Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit For given logical address space 2m and page size 2n

 Main MemoryPaging Hardware

 Main MemoryPaging Model of Logical and Physical Memory

 Main MemoryImplementation of Page TablePage table is kept in main memory Page-table base register (PTBR) points to the page table Page-table length register (PTLR) indicates size of the page table In this scheme every data/instruction access requires two memory accesses One for the page table and one for the data / instruction The two-memory access problem can be solved by the use of a special fast-lookup hardware cache called translation look-aside buffers (TLBs) (also called associative memory).

 Main MemoryTranslation Look-Aside Buffer Some TLBs store address-space identifiers (ASIDs) in each TLB entry – uniquely identifies each process to provide address-space protection for that process Otherwise need to flush at every context switch TLBs typically small (64 to 1,024 entries) On a TLB miss, value is loaded into the TLB for faster access next time Replacement policies must be considered Some entries can be wired down for permanent fast access

 Main MemoryPaging Hardware With TLB

 Main MemoryMemory ProtectionMemory protection implemented by associating protection bit with each frame to indicate if read-only or read-write access is allowed Can also add more bits to indicate page execute-only, and so on Valid-invalid bit attached to each entry in the page table: “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page “invalid” indicates that the page is not in the process’ logical address space Or use page-table length register (PTLR) Any violations result in a trap to the kernel

 Main MemoryValid (v) or Invalid (i) Bit In A Page Table

 Structure of the Page TableMain Memory

 Main MemoryStructure of the Page TableMemory structures for paging can get huge using straight-forward methods Consider a 32-bit logical address space as on modern computers Page size of 4 KB (212) Page table would have 1 million entries (232 / 212) If each entry is 4 bytes  each process 4 MB of physical address space for the page table alone Don’t want to allocate that contiguously in main memory One simple solution is to divide the page table into smaller units Hierarchical Paging Hashed Page Tables Inverted Page Tables

 Main MemoryHierarchical Page TablesBreak up the logical address space into multiple page tables A simple technique is a two-level page table We then page the page table

 Main MemoryTwo-Level Paging ExampleA logical address (on 32-bit machine with 4K page size) is divided into: a page number consisting of 20 bits a page offset consisting of 12 bits Since the page table is paged, the page number is further divided into: a 10-bit page number a 10-bit page offset Thus, a logical address is as follows: where p1 is an index into the outer page table, and p2 is the displacement within the page of the inner page table Known as forward-mapped page table

 Main MemoryAddress-Translation Scheme

 Main Memory64-bit Logical Address SpaceEven two-level paging scheme not sufficient If page size is 4 KB (212) Then page table has 252 entries If two level scheme, inner page tables could be 210 4-byte entries Address would look like Outer page table has 242 entries or 244 bytes One solution is to add a 2nd outer page table But in the following example the 2nd outer page table is still 234 bytes in size And possibly 4 memory access to get to one physical memory location

 Main MemoryThree-level Paging Scheme

 Main MemoryHashed Page TablesCommon in address spaces > 32 bits The virtual page number is hashed into a page table This page table contains a chain of elements hashing to the same location Each element contains (1) the virtual page number (2) the value of the mapped page frame (3) a pointer to the next element Virtual page numbers are compared in this chain searching for a match If a match is found, the corresponding physical frame is extracted Variation for 64-bit addresses is clustered page tables Similar to hashed but each entry refers to several pages (such as 16) rather than 1 Especially useful for sparse address spaces (where memory references are non-contiguous and scattered)

 Main MemoryHashed Page Table

 Memory management Inverted Page TableRather than each process having a page table and keeping track of all possible logical pages, track all physical pages One entry for each real page of memory Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs Use hash table to limit the search to one — or at most a few — page-table entries TLB can accelerate access But how to implement shared memory? One mapping of a virtual address to the shared physical address

 Main MemoryInverted Page Table Architecture

 SegmentationMain Memory

 Main MemorySegmentationMemory-management scheme that supports user view of memory A program is a collection of segments A segment is a logical unit such as: main program procedure function method object local variables, global variables common block stack symbol table arrays

 Main MemoryUser’s View of a Program

 Main MemoryLogical View of Segmentation1324user space physical memory space

 Main MemorySegmentation Architecture Logical address consists of a two tuple: <segment-number, offset>, Segment table – maps two-dimensional physical addresses; each table entry has: base – contains the starting physical address where the segments reside in memory limit – specifies the length of the segment Segment-table base register (STBR) points to the segment table’s location in memory Segment-table length register (STLR) indicates number of segments used by a program; segment number s is legal if s < STLR

 Main MemorySegmentation Architecture (Cont.)Protection With each entry in segment table associate: validation bit = 0  illegal segment read/write/execute privileges Protection bits associated with segments; code sharing occurs at segment level Since segments vary in length, memory allocation is a dynamic storage-allocation problem A segmentation example is shown in the following diagram

 Main MemorySegmentation Hardware

 SwappingMain Memory

 Main MemorySwappingA process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution Total physical memory space of processes can exceed physical memory Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images Roll out, roll in – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped System maintains a ready queue of ready-to-run processes which have memory images on disk

 Main MemorySwapping (Cont.)Does the swapped out process need to swap back in to same physical addresses? Depends on address binding method Plus consider pending I/O to / from process memory space Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows) Swapping normally disabled Started if more than threshold amount of memory allocated Disabled again once memory demand reduced below threshold

 Main MemorySchematic View of Swapping

 Main MemorySwapping with Paging

 Virtual Memory: IntroductionVirtual Memory

 Virtual MemoryBackgroundCode needs to be in memory to execute, but entire program rarely used Error code, unusual routines, large data structures Entire program code not needed at same time Consider ability to execute partially-loaded program Program no longer constrained by limits of physical memory Each program takes less memory while running -> more programs run at the same time Increased CPU utilization and throughput with no increase in response time or turnaround time Less I/O needed to load or swap programs into memory -> each user program runs faster

 Virtual MemoryVirtual memory Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as a large virtual memory is provided for user programs when a very small physical memory is there. Thus Virtual memory is a technique that allows the execution of processes that are not in the physical memory completely. Virtual memory – separation of user logical memory from physical memory Only part of the program needs to be in memory for execution Logical address space can therefore be much larger than physical address space Allows address spaces to be shared by several processes Allows for more efficient process creation More programs running concurrently Less I/O needed to load or swap processes

 Virtual Memory Virtual memory (Cont.)Virtual address space – logical view of how process is stored in memory Usually start at address 0, contiguous addresses until end of space Meanwhile, physical memory organized in page frames MMU must map logical to physical Virtual memory can be implemented via: Demand paging Demand segmentation

 Virtual Memory Virtual Memory That is Larger Than Physical Memory

 Demand PagingVirtual Memory

 Virtual Memory Demand PagingCould bring entire process into memory at load time Or bring a page into memory only when it is needed Less I/O needed, no unnecessary I/O Less memory needed Faster response More users Similar to paging system with swapping Page is needed  reference to it invalid reference  abort not-in-memory  bring to memory Lazy swapper – never swaps a page into memory unless page will be needed Swapper that deals with pages is a pager

 Virtual Memory Valid-Invalid BitWith each page table entry a valid–invalid bit is associated (v  in-memory – memory resident, i  not-in-memory) Initially valid–invalid bit is set to i on all entries Example of a page table snapshot: During MMU address translation, if valid–invalid bit in page table entry is i  page fault

 Virtual Memory Page Table When Some Pages Are Not in Main Memory

 Virtual Memory Steps in Handling Page FaultIf there is a reference to a page, first reference to that page will trap to operating system Page fault Operating system looks at another table to decide: Invalid reference  abort Just not in memory Find free frame Swap page into frame via scheduled disk operation Reset tables to indicate page now in memory Set validation bit = v Restart the instruction that caused the page fault

 Virtual Memory Steps in Handling a Page Fault (Cont.)

 Virtual Memory Aspects of Demand PagingExtreme case – start process with no pages in memory OS sets instruction pointer to first instruction of process, non-memory-resident -> page fault And for every other process pages on first access Pure demand paging Actually, a given instruction could access multiple pages -> multiple page faults Consider fetch and decode of instruction which adds 2 numbers from memory and stores result back to memory Pain decreased because of locality of reference Hardware support needed for demand paging Page table with valid / invalid bit Secondary memory (swap device with swap space) Instruction restart

 Virtual Memory Free-Frame ListWhen a page fault occurs, the operating system must bring the desired page from secondary storage into main memory. Most operating systems maintain a free-frame list -- a pool of free frames for satisfying such requests. Operating system typically allocate free frames using a technique known as zero-fill-on-demand -- the content of the frames zeroed-out before being allocated. When a system starts up, all available memory is placed on the free-frame list.

 Virtual Memory Performance of Demand PagingThree major activities Service the interrupt – careful coding means just several hundred instructions needed Read the page – lots of time Restart the process – again just a small amount of time