Monday, December 3, 2018

CH 15, 16, 17, 18

CH 15 – Address Translation

- The address translation would take place with the help of both Hardware and OS. Hardware would provide the low-level mechanisms to help with the translation, where as OS would help with the memory management.
- The challenge is to virtualize the memory in such a way that even though the process sets the start of the code segment at position 0, the actual address in the physical memory would be way different, and the OS needs to make sure that translation takes place with ease such that the process feels that it has its own private virtual memory but, it would still use the physical memory to address the core segments – code, stack and heap.
- **Dynamic Hardware Relocation**: In this method, there are two physical registers (**memory management unit**) residing on the (one pair per CPU) CPU – **Base** & **Bound** that would help with the translation of the virtual address into physical address @ *runtime*. The Base would be the address, from which the program's code segment starts, and Bounds would help in protection of the memory from going overboard than what is allocated to the program. Hence we get; $physical\ address = virtual\ address + base$; where base would be the value in the base register. For example, if the physical address of the starting point of code segment is *32KB*, and a particular instruction in the code segment let's say *15KB* (virtual address needs to be addressed), then we need to access *(32 + 15) KB* of the physical address. The bounds register would be used to just verify whether the memory address to be accessed in a particular instruction is within the range of a particular program.
- Operating System Involvement. **First**: When the process is first created, OS must make sure it allocates a position on the memory for the process. To do this, OS initializes a free list, keeping a track of all the physical memory slots that are free and available to be used by a process. **Second**: OS must clean up the memory once the process is terminated and add the cleared up memory back to the free list to enable it to be allocated to other processes. **Third**: During Context switches, it is the responsibility of the OS to store and restore the base and bound registers for each of the processes, because there is only one pair of base and bound registers available per CPU. Even when the a process is stopped, it can be descheduled by the OS and moved to another location on the physical memory, while storing and restoring the base and the bound register on the CPU. **Fourth**: OS should provide exception handlers to deal with exceptions that take place during runtime associated with the memory. This could be faulty use of privileged operation by the user in the user mode, or could even be accessing a memory out of the bounds from the particular processes' limits.

CH 16 – Segmentation

- When a program is divided into three continuous segments for code, stack, and heap, a lot of memory is wasted. Hence, the previous method of just using one base and bound register per program would prove to be wasteful, hence now we describe a process of having **per-segment base and bound register** which would allow us to keep each of the segments in **different physical memory** location, and still make the program thing that a continuous private memory is allocated to it.
- Therefore for the code segment, the virtual address would be added to the base register to get the physical address; however, for the heap, we would first calculate the offset that would be equal to the **virtual address - the offset register + base register value**. If a reference to an illegal address is made, it would result in a segment violation.
- To determine which segments we are referring to, **EXPLICIT WAY**: we can simply divide the 14 bits used to define a virtual address into two parts. The first two bits can help us whether we are referring to the Code Segment (**00**), Heap Segment (**01**) or Stack Segment (**11**). The remaining 12 bits would then help us get the offset. For example the virtual address 4200 in binary is represented as follows
  - **0 1** *0 0 0 0 0 1 1 0 1 0 0 0*
  - **01** –> Represents that the segment is Heap
  - *0 0 0 0 0 1 1 0 1 0 0 0* –> Gives the offset, that is, 104.
  - Also, in some cases, the code segment is stored along with the heap, hence in some systems only 1 bit is used to identify the segment.
- **IMPLICIT WAY**: In this method, the segments are identified implicitly based on where the address is generated. **Code Segment**: Address was generated from the program counter. **Stack Segment**: Address is based off of the stack or base pointer. **Heap Segment**: Any other address generated.
- However, Segments grow negatively, hence we need an additional bit specifies whether a segment grows positively or not. Once we get that, for segment we simply would have to subtract the offset from the base segment or add the negative segment to the base register value to get the physical memory location of the segment.
- Also there is an availability of a protection bit in case of memory sharing. In addition to checking whether the address is legal or not there would be an additional check whether the memory is now shareable or not.
- The issues that are raised with segmentation is that the OS get to get more and more fragmented, due to the variable size of each of the segment being allocated. This results in an issue called **EXTERNAL FRAGMENTATION**, which simply means that even though enough (total) memory is available to be allocated to a new process, the OS cannot do it because it is not available in a continue segment. To solve this we can perform something that is called **COMPACTION**, which simply accumulates the free memory available in a single continuous block. Another way to proceed with this issue is to have a free-memory list, that the OS can use to keep track of all the free memory and an algorithm that would manage the free memory.
- Unfortunately, though no matter how smart the algorithm is, external fragmentation will still exist; but will simply work towards minimizing it.

SOLVE PROBLEMS