Operating Systems Programming Assignment Unit 4

Anonymous Student

University of the People

Operating Systems Programming Assignment Unit 4

This report discusses cache coherency, sparse addresses and virtual memory, various process states, approaches for load balancing, and virtual, physical, and logical memory.

**1. Cache Coherency**

In multi-core systems, each CPU has its own local cache, or a memory storage “space” that holds recently used values. These separate caches can become inconsistent as they are updated separately by the individual CPUs. The idea of cache coherency, then, is to try to keep the data synchronized across all of the CPU caches. A solution to keeping data consistent across the caches even as each CPU is potentially updating individual caches is to do bus snooping, where “each cache pays attention to memory updates by observing the bus that connects them to main memory”(Arpaci-Dusseau & Arpaci-Dusseau, 2012). There are two approaches to cache coherency which both involve writing the cached data to main memory so other caches can be updated (“Cache Coherence and Synchronization”, n.d.): The first is to write to the cache as well as main memory immediately, called write-through. The second is to write to the cache and then wait to write to main memory until the cache will be updated again or invalidated, called write-back.

**2. Sparse Addresses And Virtual Memory**

When an operating system assigns a virtual address space to a program, by convention the code will be at the top of the address space (see the illustration below). After the code, will be the heap. However, the stack will not be placed right after the heap. Instead the stack will be placed at the very bottom of the address space. Therefore, the heap and stack have free address space in between them, which is what is meant by a “sparse address”. The benefit of having the



(adapted from Arpaci-Dusseau & Arpaci-Dusseau, 2012)

address space organized this way is to allow both the heap and the stack room to grow (and recede) as the program runs without running into other memory addresses. The heap will grow down and the stack will grow up, giving the illusion of a larger address space than there actually is (Arpaci-Dusseau & Arpaci-Dusseau, 2012).

**3. Various Process States**

According to Arpaci-Dusseau & Arpaci-Dusseau (2012), processes can be in 1 of 3 different process states: blocked, ready, or running. In the blocked state, a process is waiting for an event to occur (like an I/O event) and, therefore, it has halted execution. During the ready state, a process is ready and waiting to run but it is also not currently executing. While in the running state, the process is executing instructions on the CPU.

**4. Approaches For Load Balancing**

In multi-core systems, there could arise a situation where one CPU has significantly more processes running than another CPU, an issue called load imbalance. There are two approaches for fixing this problem and attaining load balancing: push and pull migration (Krzyzanowski, February 18, 2015). In pull migration, an idle CPU will steal processes from a CPU that is busy. In push migration, all CPUs are periodically checked by the OS and rebalanced if an imbalance is found.

**5. Virtual, Physical, And Logical Memory**

When we talk about a program’s address space, we have to distinguish the program’s physical, virtual, and logical address memory. Physical memory is the memory space where a program can execute, usually on RAM chips (“Main Memory -- RAM”, n.d.). Virtual memory is a technique for using disk space to make limited physical memory for executing appear larger, sparse, and private to running programs (Arpaci-Dusseau & Arpaci-Dusseau, 2012; “Virtual Memory: Definition”, n.d.). Logical memory is the virtual memory space created by the CPU that a user/program can see, so that the physical address is not used directly (“Difference Between Logical and Physical Address in Operating System”, September 21, 2016).

**Conclusion**

We learned much this week about how to schedule multi-core systems and how the CPU virtualizes memory. This lab report specifically discusses cache coherency, sparse addresses and virtual memory, various process states, approaches for load balancing, as well as virtual, physical, and logical memory. All of this together gives us a better understanding of how much work the operating system is doing behind the scenes in order to give the illusion that multiple programs can run at the same time.

References

Arpaci-Dusseau, R. & Arpaci-Dusseau, A. (2012). *Operating Systems: Three Easy Pieces. Madison, WI: University of Wisconsin-Madison.* Retrieved from <http://pages.cs.wisc.edu/~remzi/OSTEP//>

“Cache Coherence and Synchronization” (n.d.). *Tutorialspoint*. Retrieved from <https://www.tutorialspoint.com/parallel_computer_architecture/parallel_computer_architecture_cache_coherence_synchronization.htm>

“Difference Between Logical and Physical Address in Operating System” (September 21, 2016). *Tech Differences*. Retrieved from <https://techdifferences.com/difference-between-logical-and-physical-address.html>

Krzyzanowski, P. (February 18, 2015). Process Scheduling. *PK.org.* Retrieved from <https://www.cs.rutgers.edu/~pxk/416/notes/07-scheduling.html>

“Main Memory -- RAM” (n.d.). *Red Hat*. Retrieved from <https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/4/html/Introduction_To_System_Administration/s2-memory-ram.html>

“Virtual Memory: Definition” (n.d.). *Illinois Institute of Technology.* Retrieved from <http://www.cs.iit.edu/~cs561/cs351/VM/Definiton.html>