CMPSCI 691ST Systems **Fall 2011**

Lecture 8

Lecturer: Emery Berger *Scribe: Ryan Hurley*

8.1 The Structure of "THE" - Multiprogramming System

8.1.1 Summary

"THE" Multiprogramming System was an operating system developed by Dijkstra while at the Technological University in Eindhoven. The operating system was designed to be a single user while executing concurrent programs. The system uses a virtual memory abstraction to allow for addressing passed the physical storage space. The systems also handles multiprogramming throught the use of semaphores to control process interactions.

8.1.2 Discussion

Besides the obvious system impacts of the paper there is this notion of software engineering being used throughout the implementation of "THE". It is important to note that systems people generally view software engineering as software developement while software engineering, according to those who study it, involve things including testing and model checking. There is much overlap between the two especially when correctness is the topic of conversation, but there are aspects of software engineering that Dijkstra would scoff at, including testing. Dijkstra did not believe in testing. If you had to do testing of a system, then you failed at developement. There was even a time when Dijkstra harrassed Nancy from MIT by smoking two packs during her presentation at the University of Texas because he hated her testing methods.

"THE" was designed using layers of abstraction. This allowed for the use of virtual memory and syncronized communication between different parts of the operating system. In addition, Dijkstra invented the use of semaphores. Semaphores created critical sections which helped to ease the analysis of code interleavings. The design of the system pioneered the use of modular design, and was a good initial design for multiprogramming systems. Linux is not very good at this, and its success seems to be attributed to a fortunate series of events. Linux is very unsafe with pointer operations, but since it employs the "Big Kernal Lock", these accesses are mostly "controlled". Unlike Linux, most micro-kernals dealt with sharing information by copying information instead of passing pointers. The latter is much slower but much more reliable. Microsoft Research attempted to develop an operating system known as "Singularity" with the goal of creating a more dependable operating system without compromising on speed. Singularity worked by passing objects through channels. As far as processes were concerned data was being copied, but behind the scenes a pointer is being passed. Since Microsoft developed using a new version of C-Sharp once you pass data you cannot use it again, and this is verified throught static checking at compile time. Pointer speed with copying protection.

8.2 The Working Set Model for Program Behavior

8.2.1 Summary

The working set was developed as a paging algorithm. It is based upon the idea that there is simply not enough memory for all processes to use, but a process usually needs all its pages in memory to perform its work efficiently. Denning invented the notion of a working set which is the number of pages an application needs to do its work over a sampling time of τ. In other words, an ideal working set is the amount of memory an application needs to do its job under normal use. The goal of the working set memory manager was to maximize the number of active processes with complete working sets. If a process can't fit then *all* the pages of that process are removed from memory. This could page an entire movie player out of memory if its consumption is too large.

8.2.2 Discussion

The working set was developed as an alternative to other paging algorithms like global LRU. The Prisims lab also tried developing another memory allocator known as Redline. Its driving mantra was that most users care about low latency, not throughput. Conversely, the working set design was focused on throughput not latency. Redline allows an application to specify the amount of cpu time it needs to perform its tasks. In the worst case, all applications say they need everything in which case we are back to using global LRU, butIn most cases Redline was able to run many more active applications than a normal linux machine. It's important to note that the notion of working set management only seems dismal according to our use of current operating systems. It simply has not stood up to the test of time.

Discussion ended by describing the evolution of browsers, and how they have become their own operating systems in a way. One of the most interesting aspects of Chrome and Firefox is how they perform memory management. Chrome has a memory pool reserved for its ”kernel” tasks, but also allocates separate memory based on the domain name of open tabs. Firefox on the other hand has one large pool of memory where information is distributed among the pages. Firefox would do terribly under the working set, since all the pages it has allocated would be needed for any task it might need to perform. Chrome's working set would involve the main pool memory in addition to any active domains, but it is entirely possible that many tabs could be paged out ”as God intented” (Emery Berger).