COMP 3000
Lecture 2, Sep. 14th 2004
Notes by Rossen Atanassov

On the assigned reading from last time –"In the Beginning was the Command Line." By Neal Stephenson. Since many students were not too happy of the fact that such a long reading was assigned, a pool of different opinions was defined (below).

Cons.

1. Too subjective
2. too much unnecessary details
3. outdated
4. too much metaphors
5. too biased

Pros.

1. amusing (important)
2. good use of metaphor and analogies
3. good anecdotes
4. nostalgic
5. little bit of history
6. nice OS comparisons

It is rather easy to see most of the Cons. As the author obviously wrote the article in 1999, this right away makes the essay outdated today. Further, the fact the Stephenson was quite fed up with Apple and their OS made his arguments quite biased.
On the other hand we can't deny the amusement and language used by the author. Aside from his biased point of view he was still able to make good metaphorical comparisons between the different OS's.

Why did the professor assigned it?

1. spite? :-)
2. background on OS – the author actually goes through most of the operating systems used today, pointing out the key features of each.
3. appreciate the textbook – as we are used to read more technical literature, the line between the essay the textbook used in class becomes quite obvious.
4. ideas for term paper

End of assignment review.

First lets list some different OS's

- Free, Open BSD
- MS DOS
- UNIX family
- Applesoft Basic
- Linux family
- React OS
- Solaris
- Cisco IOS
- Windows '95 and up
- Windows NT and up
- Windows CE
- Windows 3.x
- Netware
- BeOS
- AIX
- Mac OS <10
- Mac OS X
- Vx works
- VMS (connection with Win NT)
- Palm OS
- HPIX
- QNX
- IRIX
- Amiga OS
- CP/M
- Plan 9

So it seems there are lots of different Operating System. What is the big deal about them, and how many of them are totally different?
Turns out that all of the ones in **green** are POSIX compliant, and all of the **blue** ones are win32 compliant. Almost every *NIX OS today implements POSIX in some extent.
What is also interesting to notice that the rest of the OS's that are not colored are now obsolete. Exceptions are Palm, VX and IOS because these are embedded and hence harder to replace.
Back in the days most computers had their own operating system and all applications had to be ported to them. Was this good or bad? Well it depends on the point of view.
Also worth mentioning is that all operating systems before were all character driven until the GUI based OS's came.

Solved and Unsolved problems in operating systems.

Solved

- Processes and Threads – multiple processes executed at the same time.
- Protected memory – this causes only an application causing a problem to be terminated instead of the entire OS to crash.
- Virtual memory – space assigned and used as if it was system memory in order to handle large applications.
- File systems.
- Interrupts and Exceptions (exceptions are more like Soft Interrupts)
- Cryptographic security – encrypted file systems
- Access control – password protection etc.
- Multiuser systems
- Parallel / SMP

Unsolved

- General code and data security
- What is the next user interface? Voice, gestures, something else?
- System administration – every user should be able to administer their system.
- Distributed fault tolerant systems – shared recourses among multiple systems used as one. Ex. If a hard drive goes down, that should not interrupt the performance of the operating system. A real life example that is close to that is what Google uses for their system. Currently they use over 100000 different systems as one. Google has solved the hardware side of this, but not the software side.
- Denial of service (DOS) – users should not be able to destroy the operating system by executing harmful commands or code (ex: rm –rf /).

Requirements of operating systems

- Be efficient – as we know, all operating systems are pure overhead! If we were writing everything directly in assembly and from scratch for every single application we write, sure it is going to be much faster, but who in their right mind would do that?
- Control access to resources *safely* – a user should not be able to destroy the hardware through code.

- Run forever – example is Turing machine, but the problem with that is that TM only executes functional computations.
- Recover from bugs and failures – this is very hard.
- Live cleanly – what this means is simply that maintenance should be handled with high priority. (ex: if you only have to live one day, you would care of you good or bad in your life)

Cruft is inevitable!

- Broken hardware – all kinds of hardware bugs (F00F – see Appendix A). If the chip executes these bugs at some point it could physically harm the hardware. The only way around these types of problems is pure cruft.
- Broken software – backwards compatibility for example causes cruft.
    - Applications
    - API's
- OS could be thought of as a cathedral. It is usually built over long periods of time, and as time progresses different people/generations work on it and the original ideas are usually not followed as close. The same goes for OS's – we are still living with the decisions that were made long time ago (back in the 60's). Things have to change!

End of class

Appendix A

The following is quoted from the article "The Intel Pentium F00F Bug: Description and Workarounds" by Robert R. Collins. The article is available on his website: http://www.rcollins.org/Errata/Dec97/F00FBug.html

Here's how the F00F bug works:

When the Intel processor encounters this instruction (F0 0F C7 C8, or anything from F0 0F C7 C8..CF), the F00F bug occurs. The processor recognizes that an invalid opcode has occurred and tries to dispatch the #UD handler. Because of the LOCK prefix, the processor is confused. When the processor issues the bus reads to get the #UD handler vector address, the processor erroneously asserts the LOCK# signal. The LOCK# signal can only be asserted for read-modify-write instructions which modify memory. When the bus is locked, a locked memory read must be followed by a locked memory write, lest unpredictable results may occur. But in this case, the LOCK# signal remains asserted for the two consecutive memory reads required to retrieve the #UD vector address. The processor never issues any intervening locked write, and then hangs itself. This behavior is shown in the logic analyzer trace in Listing 1. As you can see, the Pentium tries to retrieve the #UD vector with two locked reads. After that point, all processor activity stops.

```
Sequence Address  Data          Mnemonic                        Timestamp
-------------------------------------------------------------------------
T 524285 000000B2 ----7E--------(-IO-WRITE-)----------------------300-ns
  524286 00000018   ----E14C    ( LOCKED MEM READ )                440 ns
  524287 0000001A   F000----    ( LOCKED MEM READ )                100 ns
```