Snapshot System State Management

Pau Garcia i Quiles
http://www.elpauer.org

1. Motivation

These days it's usual to have several applications running at once in your computer, and switch from one to another one on a regular basis. For example, you can be running your mail application, your word processor, a MP3 player, an Internet browser and a spreadsheet.

What happens if lights go out and the computer turns off? You lose all your information since the last time you saved your work (and you have to save your work individually in each application, assuming your application lets you save your context –few browsers allow it–).

Currently, Windows and some other modern operating sytems allow you to shutdown your computer in a special model called "hibernation", but "hibernation" takes some time and is intended as a shutdown, not as a save-here-and-continue-working tool.

2. The idea

What I propose is a method for system state management in a way you *never* lose your work. RAM, CPU registers and so on should be presserved all the time.

If lights go out, when the computer reboots you will be offered the chance to do or a resume-reboot (restoring the previous system state) or a fresh reboot (useful if you restarted your computer because of a virus or a lock, for instance)

3. Some thoughts on how to do it

In a computer designed following the Von Neuman architecture, the essential elements to presserve to recover previous system state would be:

- CPU registers
- RAM contents
- List of open files (already in the RAM)
- System clock
- Maybe, Video RAM contents

An element we would not need to save its state would be cache memories. For one side, saving them would imply previous system functionality would be faster, giving more responsiveness. In my opinion, though, the expense would be hardly justified.

Of course, there would be some other external elements involved, such as the network concentrator (modem, router, switch or something like), but this document introduces a computer-related design. Each other element in the computer environment, but external to the computer itself ("the box"), would need to have its own method.

3.1. RAM

Having RAM contents always updated to the moment just before the system went off is the most difficult task in this proposal. Most systems today have 256 MB of RAM, and it’s easy to find computers with 512 MB or even 1 GB. Saving such a big amount of data takes time, and if lights go out, we are scarce of time.

3.1.1. A hardware-only solution

I think a hardware only solution would be possible, for example, using static RAMs.

When time comes for system restore, a program would load the system-state and RAM contents.

The biggest detrimental is it would be too expensive. This method would perfectly fit our needs and requires little software development, but it requires a big investment in hardware.

3.1.2. A software-only solution

A software-only solution (without any kind of newly-designed hardware) would also be possible, but information would be somewhat outdated.

For instance, you can save the entire contents of your RAM every 10 seconds in a file in the hard disk. Actually, it would work better if you have a dedicated hard disk in a dedicated IDE channel, because you would always have the headers over the surface to be written.

The same RAM-restorer I described before would be needed, of course.

3.1.3. A mixed software-hardware solution

A mixed SW/HW solution I have come up would be having a little power-suppy (some kind of UPS) for the RAM, the graphics card, the CPU, and the device you would write the information to.

For instance, imagine you would write the RAM contents in a hard disk. When lights go out, your computer would have a tiny internal UPS, directly cabled to the RAM, CPU, and so on, providing enough power for the CPU registers to be copied, the RAM to be dumped to the hard disk, and so on.

Again, we would need that RAM-restorer software.

3.2. Other elements’ data

Saving data contained in other elements of the computer is easier, because the amount of information we have to save is significantly less: CPU registers, the time of the clock, and so on, take up very few space (some Kilobytes, at most).

3.3. Not so easy, though

Some problems may arise with swap, journal file systems and network connections.

- Swap: if the swap space is erased when the system boots and we have not saved swap data (like RAM was), then recover the previous system state might be impossible
- Journal file systems[1]: if we lose data abour transactions that are not finished, it may be be impossible to recover the previous sytem state
- Network connections: it may be impossible or difficult to recover previous connections, due to several reasons (the other peer is no longer on line, use of stateless protocols, and so on).

A deeper study is needed to find other problems and inconveniencies that may appear.

3.4. Where to dump the data

For the hardware side, where could we dump to all this information (RAM, VRAM, CPU registers, and so on)?

An average Seagate or Maxtor SATA hard-disk has an access time of 9 ms, with a latency of 4.2 ms, and is able to write 48 MB per second[2]. This means saving system state would take:

System RAM	*Time to save system context*
256 MB	5.35 seconds
512 MB	10.67 seconds
1024 MB	16.1 seconds

As we see, times are very high for a hard disk.

In opposition to hard disks, we could use some kind of very fast and large capacity memory, for instance a PuRAM[3] (writing speed=6.4GB/s, more than 100 times faster than any hard disk) instead of static RAM.

3.5. Taking advantadge of ACPI

The ACPI[4] specification provides an API for the hibernation (and some other features) of PC-compatible computers. Linux has support for ACPI, although not included in the 2.4 series[5,6].

The SSSM could be implemented using calls to this API, dodging the computer: you tell the system it is going to hibernate, but you only save the current state and continue working.

For the restore side, calling the API functions to wake up the computer would do.

4. Colophon

For the desktop, implementing the SSSM would mean happier users and a big increase in the productivity of workers, because no work is lost.

For servers, it would mean the end of UPSs as we know them today. They would not be needed anymore for modern systems to be shut down in an ordered way. We would, then, need UPSs only for systems that need no disruption in service time.

Even though the SSSM could be implemented with current hardware (using an external UPS and saving to a hard-disk), this would definitely rule if integrated in PC motherboards, because it would be transparent to the user.

5. References

[1] Journal File Systems
http://www.linuxgazette.com/issue55/florido.html

[2] Serial ATA HDD Roundup
http://www.digit-life.com/articles2/sata-summer-2003/index.html

[3] PuRAM
http://www.go-l.com/laptops/producer/architecture/index.htm

[4] The ACPI Specification
http://www.acpi.info/

[5] ACPI4Linux
http://acpi.sourceforge.net/

[6] Software Suspend for Linux
http://softwaresuspend.berlios.de/