Back in a Flash! - Fast Recovery using Non-Volatile Memory

Jeffrey C. Carlyle, Francis M. David, Roy H. Campbell
Department of Computer Science
University of Illinois at Urbana-Champaign
201 N Goodwin Ave, Urbana, IL 61801
{jcarlyle,fdavid,rhc}@uiuc.edu

Abstract

In this paper we present a technique for improving the reliability and availability of services on mobile devices by utilizing flash memory for fast component restarts.

1 Introduction and Motivation

Users of mobile devices, such as cell phones, have come to expect a certain level of service from their devices. Many cell phone users have previously used land-line telephones and have grown accustomed to the availability and reliability offered by wired telephone services; however, cell phones have quickly grown in scope beyond being simple communications devices. Today, cell phones regularly include features such as cameras, audio and video players, and personal information managers. Indeed cell phones have become general purpose computing platforms.

As cell phones have grown in complexity, they have picked up some undesirable traits from their personal computer (PC) cousins. The reliability of the phone is dependent on the reliability of the operating system (OS) running on the phone. Current smart cell phones are adopting operating systems that are stripped down versions of desktop systems such as Linux and Windows. These operating systems have very little tolerance for errors within the kernel. Such operating system errors and errors within important out-of-kernel services can render a phone useless unless it is rebooted. Any phone calls or modem connections in progress during the reboot are dropped leading to possible loss of information. This can be also be frustrating, especially in emergency situations: a user needing to call 911 should not need to wait a minute or more for their phone to reboot.

Restarting components has been shown to be an effective recovery technique [4] that does not require complete rebooting. In the past, we have explored restarting arbitrary OS components by restructuring OS state [5]. In this work, we present techniques that can be used to rapidly restart OS and software components. If the phone cannot be recovered through component restarts, a complete reboot is used as a last resort. The reboot itself also benefits from fast component restart support.

2 Restart Overheads

One of the contributing factors to the amount of time it takes to restart a failed component or system is the amount of code and data that is accessed during initialization. Kernel modules may need to be read from disk, and user-space applications are usually dependent on one or more shared libraries which may also need to be reloaded during a restart. There is also a potential for overhead associated with run-time linking of shared libraries; this overhead can often be diminished by using techniques such as prelinking [7]. In addition to the code itself, some applications will need to load data resources such as icons, sounds, and wallpapers.

Another potential source of overhead lies in the choice of file system. Since embedded devices usually have constraints on storage space, developers often employ compressed file systems such as CramFS [1]. Compressed file systems are a significant source of execution overhead because they require an entire block to be read and decompressed into RAM before use. If the information used during the restart is stored across multiple blocks then this must be repeated for each of these blocks. A potential optimization to save time during a restart would be to organize the code and data required for the restart so that it is present in a small working set of file system blocks. Since the file systems containing code for embedded devices are usually read-only, implementing this optimization requires an analysis of code used during restart. With the cost of storage decreasing, even on embedded systems, developers should consider placing at least some code and data in uncompressed file systems to eliminate more of the overhead due to the file system.

3 Fast Restarts

We present a technique which aims to provide fast restarts with minimal changes to the code of the component being restarted. The core of our fast restart technique is to save a snapshot of the state (essentially the code and data) of the component immediately after initialization, when it is in a state which is known be valid. If the component needs to be restarted, the in-memory crashed version of the component is replaced with the saved snapshot. These snapshots are only taken once and could potentially be saved onto the device during manufacturing.

In principle, this method can be compared with hibernation [9]. In hibernation, the contents of RAM are saved to secondary storage at power down and loaded into RAM at power up; however, our focus is at the granularity of a single component instead of the entirety of RAM. Additionally, we only save component images once and not at every shutdown.

Mobile devices typically use flash memory for secondary storage. Flash memory is reprogrammable non-volatile memory. Some characteristics of flash memory may prove to be useful for fast restarts. In particular, reads to NOR-based flash memory can be carried out in the same way as reads to traditional ROM or RAM. That is, NOR-based flash memories can be mapped into the processor's address space and read from without any special access techniques. This characteristic allows code to be executed-in-place (XIP) from flash memory.[1] Some systems, like Linux, already use this technique to combat space constraints and decrease boot times. Such systems usually require that code be compiled and linked in order to support XIP; however, our design does not have such a requirement.

For fast recovery we leverage the fact that images saved in flash memory can be read as if they were in RAM. When we recover a crashed component, page tables in the system are updated to point into the known correct image stored in flash. This can be done for both executable information and data. Both read-only and read-write pages can remain in flash memory after a restart. When an attempt is made to write to a virtual address which points a physical address in flash memory, the operating system uses a copy-on-write technique to copy the page into RAM and update the page tables to point to the new physical address in RAM. Since reads from flash memory are usually slower than reads from RAM, it makes sense to consider moving some pages back into RAM even before a write occurs. Some experimentation is required to determine the proper threshold for this.

If such a system is to become commonly accepted, the implementation should also address potential security concerns. Care should be taken to protect any confidential information stored in the saved state. Another consideration is protection of the system from subversion by an attacker who maliciously changes the saved state information.

We are currently in the process of implementing this system on the ARM ports of Linux and Choices [3].

[1] XIP can also be emulated on NAND-based flash memory [8].

4 Related Work

Restarting OS components for reliability is the approach followed in the Minix3 microkernel OS [6]. It, however, lacks the fast restart optimizations discussed in this work. EROS [10] uses persistent systemwide checkpoints on disk to quickly recover from failures; however, it is difficult to assert that a checkpoint that is made at some arbitrary point in time is correct. In our design, individual components are checkpointed at an initial correct state.

Chorus provides persistent memory regions that are used to implement quick restarts [2]; however, unlike our design, components have complete access to their persistent memory regions and there is no mechanism in place to prevent data in the persistent memory region from being corrupted.

References

[1] Compressed ROM Filesystem for Linux. `http://lxr.linux.no/source/fs/cramfs/`.

[2] V. Abrossimov and F. Hemann. Fast Error Recovery in CHORUS/OS: The Hot-Restart Technology. Technical Report CSI-T4-96-34, Chorus Systems, Inc., August 1996.

[3] R. H. Campbell, G. M. Johnston, and V. Russo. "Choices (Class Hierarchical Open Interface for Custom Embedded Systems)". *ACM Operating Systems Review*, 21(3):9–17, July 1987.

[4] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox. Microreboot – A Technique for Cheap Recovery. In *Symposium on Operating Systems Design and Implementation*, San Francisco, CA, December 2004.

[5] F. M. David, J. C. Carlyle, E. M. Chan, P. A. Reames, and R. H. Campbell. Improving Dependability by Revisiting Operating System Design. In *Workshop on Hot Topics in Dependability*, Edinburgh, UK, June 2007.

[6] J. N. Herder. Towards a True Microkernel Operating System. Master's thesis, Vrije Universiteit Amsterdam, 2005.

[7] J. Jelínek. Prelink. `http://people.redhat.com/jakub/prelink.pdf`, March 2004.

[8] C. Park, J. Seo, S. Bae, H. Kim, S. Kim, and B. Kim. A low-cost memory architecture with NAND XIP for mobile embedded systems. In *CODES+ISSS '03: Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 138–143, New York, NY, USA, 2003. ACM Press.

[9] K. Reneris. United States Patent 6,209,088: Computer hibernation implemented by a computer operating system. March 2001.

[10] J. S. Shapiro. *EROS: A Capability System*. PhD thesis, University of Pennsylvania, 1999.