The Nanvix Operating System

Pedro H. Penna

August 4, 2015

1 Introduction

Nanvix is an operating system created by Pedro H. Penna for educational purposes. It was designed from scratch to be small and simple, and yet modern and fully featured, so that it could help both, novices and experienced enthusiasts in operating systems, to learn about kernel hacking. The first release of Nanvix came out in early 2011, and since then the system has gone through several changes. This paper details the internals of Nanvix 1.2. All previous and future releases are available at `github.com/ppenna/nanvix`, under the GPLv3 license.

In this section, we present an overview of Nanvix, starting with the system architecture, then presenting the system services, and finally discussing the required hardware to run the system. In later sections, we present a more detailed description of each part of the system.

1.1 System Architecture

The architecture of Nanvix is outlined in Figure 1. It presents a similar structure to Unix System V, and it has been intentionally designed to be so due to two points. First, several successful operating systems, such as Aix, Linux and Solaris, are based on this architecture [??]. Second, System V has earned Dennis Ritchie and Kenneth Thompson the 1983 Turing Award [??]. These points indicate that System V is a well-architected and reliable system, thus serving as a good baseline design for a new educational operating system, such as Nanvix.

Nanvix is structured in two layers. The kernel, the bottom layer, seats on the top of the hardware and runs in privileged mode, with full access to all resources. Its job is to extended the underlying hardware so that: (i) a more pleasant interface, which is easier to program, is exported to the higher level; and (ii) resources can be shared among users, fairly and concurrently. The userland, the top layer, is where all user software run in unprivileged mode, with limited access to the hardware, and the place where the user itself interacts with the system.

The kernel presents a monolithic architecture, and it is structured in four subsystems: the hardware abstraction layer; the memory management system; the process management system; and the file system. The hardware abstraction layer interacts directly with the hardware and exports to the other subsystems a set of well defined low-level routines, such as those for dealing with IO devices, context switching and interrupt handling. Its job is to isolate, as much as possible, hardware intricacies, so that the kernel can be easily ported to other compatible platforms, by simply replacing the hardware abstraction layer.

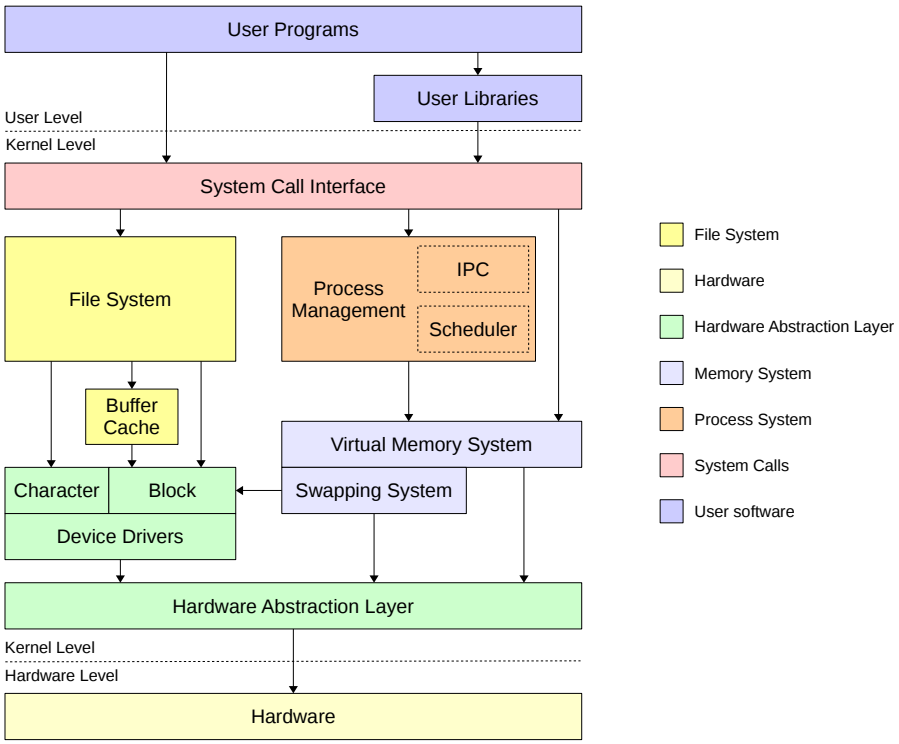

Figure 1: Nanvix architecture.

The memory management subsystem provides a flat virtual memory abstraction to the system. It does so by having two modules working together: the swapping and virtual memory modules. The swapping module deals with paging, keeping in memory those pages that are more frequently used, and swapping out to disk those that are not. The virtual memory system, on the other hand, relies on the paging module to manage higher-level abstractions called memory regions, and thus enable advanced features such as shared memory regions, on-demand loading, lazy coping and memory pinning.

The process management system handles creation, destruction, scheduling, synchronization and communication of processes. Processes are single thread entities and are created on demand, either by the system itself or the user. Scheduling is based on preemption and happens in userland whenever a process runs out of quantum or blocks awaiting for a resource. In kernel land, on the other hand, processes run in nonpreemptive mode and scheduling occurs only when a processes voluntarily relinquishes the processor. Finally, processes many synchronize their activities using semaphores, and communicate with one another through pipes and shared memory regions.

The file system provides a uniform interface for dealing with resources. It extends the device driver interface and creates on top of it the file abstraction. Files can be accessed through a unique pathname, and may be shared among several processes transparently. The file system is compatible with the one present in the Minix 1 operating system, it adopts an hierarchical inode structure, and supports mounting points and disk block caching.

The userland relies on the system calls exported by the kernel. User libraries wrap around some of these calls to provide interfaces that are even more pleasant to programmers. Nanvix offers great support to the Standard C Library and much of the current development effort is focused on enhancing it. User programs are, ultimately, the way in which the user itself interacts with the system. Nanvix is shipped out with the Tiny Shell (`tsh`) and the Nanvix Utilities (`nanvix-util`), which heavily resemble traditional Unix utilities.

1.2 System Services

1.3 Hardware Requirements