VE482 — Introduction to Operating Systems

Introduction to Minix 3

Manuel — UM-JI (Fall 2019)

The goal of this document is to install MINIX 3, a POSIX conformant, open-source system freely available at http://www.minix3.org. The description appearing in this document is directly extracted from the textbook *Modern Operating Systems*, from A. Tanenbaum.

The MINIX 3 microkernel is only about 3200 lines of C and 800 lines of assembler for very low-level functions such as catching interrupts and switching processes. The C code manages and schedules processes, handles interprocess communication (by passing messages between processes), and offers a set of about 35 kernel calls to allow the rest of the operating system to do its work.

These calls perform functions like hooking handlers to interrupts, moving data between address spaces, and installing new memory maps for newly created processes. The process structure of MINIX 3 is shown in Figure 1, with the kernel call handlers labeled Sys. The device driver for the clock is also in the kernel because the scheduler interacts closely with it. All the other device drivers run as separate user processes.

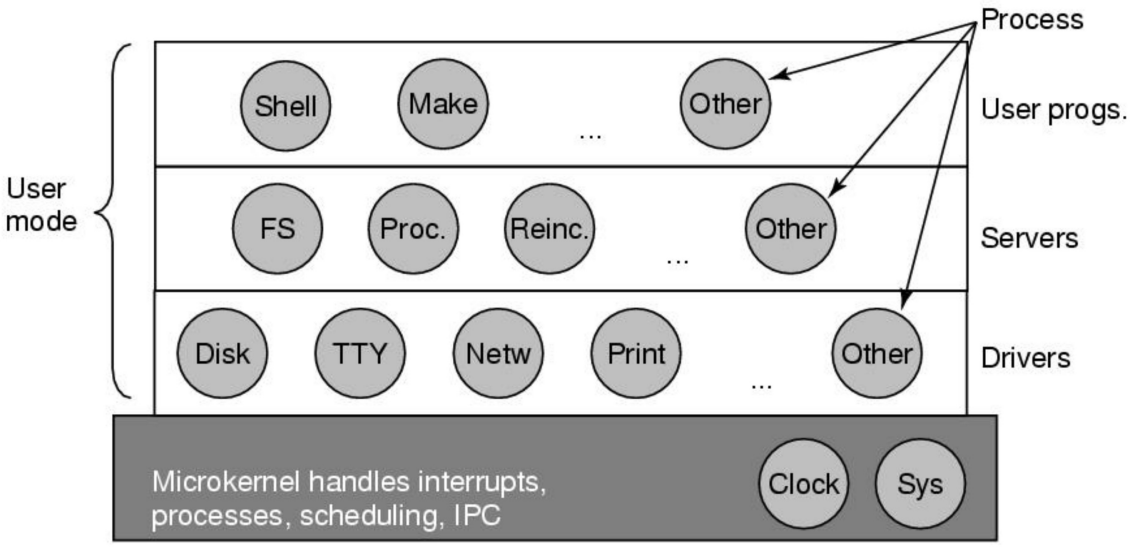

Figure 1: Structure of the Minix3 Operating System

Outside the kernel, the system is structured as three layers of processes all running in user mode. The lowest layer contains the device drivers. Since they run in user mode, they do not have physical access to the I/O port space and cannot issue I/O commands directly. Instead, to program an I/O device, the driver builds a structure telling which values to write to which I/O ports and makes a kernel call telling the kernel to do the write. This approach means that the kernel can check to see that the driver is writing (or reading) from I/O it is authorized to use. Consequently, (and unlike in a monolithic design), a buggy audio driver cannot accidentally write on the disk.

Above the drivers is another user-mode layer containing the servers, which do most of the work of the operating system. One or more file servers manage the file system(s), the process manager creates, destroys, and manages processes, and so on. User programs obtain operating system services by sending short messages to the servers asking for the POSIX system calls. For example, a process needing to do a read sends a message to one of the file servers telling it what to read.

One interesting server is the reincarnation server, whose job is to check if the other servers and drivers

are functioning correctly. In the event that a faulty one is detected, it is automatically replaced without any user intervention. In this way the system is self healing and can achieve high reliability.

The system has many restrictions limiting the power of each process. As mentioned, drivers can only touch authorized I/O ports, but access to kernel calls is also controlled on a per process basis, as is the ability to send messages to other processes. Processes can also grant limited permission for other processes to have the kernel access their address spaces. As an example, a file system can grant permission for the disk driver to let the kernel put a newly read in disk block at a specific address within the file system's address space. The sum total of all these restrictions is that each driver and server has exactly the power to do its work and nothing more, thus greatly limiting the damage a buggy component can do.

An idea somewhat related to having a minimal kernel is to put the mechanism for doing something in the kernel but not the policy. To make this point better, consider the scheduling of processes. A relatively simple scheduling algorithm is to assign a priority to every process and then have the kernel run the highest-priority process that is runnable. The mechanism – in the kernel – is to look for the highest-priority process and run it. The policy – assigning priorities to processes – can be done by user-mode processes. In this way policy and mechanism can be decoupled and the kernel can be made smaller.

Tasks to perform to install Minix 3:

1. Make sure hardware virtualisation is enabled on your computer (BIOS/UEFI setup).
2. Install a virtualisation solution, e.g. qemu.
3. Download Minix 3, version 3.2.1, from the official website: http://www.minix3.org.
4. Install Minix 3 in a virtual machine.
5. Some help and installation guidelines can be found here.
6. Check both the user-guide (can also be useful during the installation process), and developer-guide.