David Merrick

CS 411

4 June, 2013

Writeup for Final Exam

The Android operating system is built on top of a Linux foundation. We have spent the semester studying in-depth the ways in which the Linux kernel schedules processes, schedules I/O, handles memory allocation, encryption, and how the kernel interacts with devices through drivers. I will be comparing and contrasting the implementation we used for class with the Android operating system in terms of process scheduling and I/O scheduling.

Overview: The Android kernel is a modified version of Linux. Over the course of development, some changes were made. The number of changes is relatively small, however. There have been approximately 250 patches that take up about 3 megabytes of differences over 25,000 lines of code. Some of this code is used to implement a flash filesystem, and some of it is to patch the network security. [**?**]. Android is based on Linux 2.6.32. It has some power-saving features built in and excludes many libraries, shells, editors, and programming frameworks that ship with most distributions of Linux. On top of the kernel runs Dalvik, which comprises the Java virtual machine and many basic runtime essentials. Most Android apps run inside this VM. [**?**]

Process Scheduling in Android

Overview: A scheduler is the algorithm for sharing the CPU between multiple processes in an operating system. A policy is the behavior of the scheduler that determines what runs when. Changes in policy can result in operating systems that are optimized for specific tasks (such as user responsiveness). Therefore, this is a critical aspect of an operating system. [**?**] pg 43.

In the kernel we were working with, the default scheduler was the Completely Fair Scheduler (CFS). The idea behind CFS is to model scheduling as if the system had an ideal processor that could multitask perfectly. In this kind of system, each runnable process n would receive 1/n of the CPU time. In this way, CFS assigns each process a proportion of the CPU. For example, in pre-CFS UNIX systems, if two processes were to run, one would be run right after the other and each would receive 100% of the CPU power. In CFS, the two processes would run simultaneously at 50% of the CPU power. CFS performs this by running processes round-robin style for very small timeslices, so that in a given period of time it will appear as though many processes are running in parallel. [**?**] pg. 48-50

Real-time scheduling means to schedule processes within timing deadlines. The Linux scheduler provides soft real-time behavior, meaning that it tries to schedule processes within timing deadlines but doesn't make guarantees to always achieve this. Linux provides two real-time scheduling policies, FIFO (SCHED_FIFO) and round-robin (SCHED_RR). I will be using SCHED_RR and round-robin interchangeably in this writeup, likewise for SCHED_FIFO and FIFO. Both round-robin and FIFO make use of runqueues, which are essentially queues containing process descriptors for runnable processes. Round-robin scheduling cycles through processes, running each for a pre-specified amount of time known as a timeslice. On a more technical level, this means placing the process descriptor at the end of the runqueue after its timeslice, then running all subsequent tasks until it reaches the end of the queue, at which time it will start the cycle again. FIFO (First In, First Out) scheduling runs each SCHED_FIFO process of the same priority until it is finished, then runs the next process in the runqueue. It does this in order of when processes were placed in the runqueue. On a technical level, FIFO scheduling behaves almost identically to round-robin scheduling but with infinite timeslices. [**?**]

Real-time scheduling in the Linux kernel we were working with prioritizing processes. Processes with a higher priority run before processes with a lower priority, and processes of the same priority run one right after the other, round-robin style. Our Linux kernel used two different priority ranking systems. One system ranked processes based on their "nice value." This value ranged from -19 to 20 with 0 as the default. A larger nice value corresponds to a lower priority. Meaning that a process is being "nice" to other higher-priority processes by yielding control of the CPU. The second priority ranking system in Linux is the real-time priority. These numbers range from 0-99. Unlike nice values, higher priority values correspond to higher priority. [**?**]

In Android, by default, applications run as processes in a single thread of execution. By default, every component of the

application is run in the same process. However, different components of an application can be run as separate processes, and extra threads for each process can be created by the developer's specifications. This is all specified at runtime in the manifest file for each application. Only a certain amount of memory is able to be allocated for all processes, so the kernel may decide to put a process to sleep when memory is required for others. The kernel chooses which applications to put to sleep based on an algorithm that estimates their relative importance to the user. A process whose activities are visible on the screen, for example, takes a higher priority than one whose activities are not visible. [**?**]. When two applications have the same priority, the process that has been at a lower priority longest will be killed first. [**?**]

Android has separate process groups for background and foreground processes. Android uses a number of priorities so that background processes do not interrupt foreground ones. User interface threads run at default priority. Applications that are moved to the background have all their threads forced into background process priority. Background processes in Android are not allowed to take up more than 10% of the CPU time needed by foreground threads. This is implemented using a Linux algorithm called cgroups. [**?**]

The goal behind the Android scheduling policy is responsiveness to the user. Android, like the Linux foundation it is built on, uses time slices for scheduling processes. Processes can have a dynamic priority from 19 to -20, very high to very low, respectively. Higher priority processes will get more CPU time when needed. At startup, Service processes are run in the background process group unless this has been specified otherwise. Background threads are given a priority level of 10. [**?**]

References

[1] Daniel Bovet. Understanding the linux kernel. `http://oreilly.com/catalog/linuxkernel/chapter/ch10.html`.

[2] Tim Bray. What android is. `http://www.tbray.org/ongoing/When/201x/2010/11/14/What-Android-Is`.

[3] eLinux.org. Android kernel features. `http://elinux.org/Android_Kernel_Features`.

[4] gomo. Scheduling in android. `https://github.com/keesj/gomo/wiki/AndroidScheduling`.

[5] Android Developer Guide. Processes and threads. `http://developer.android.com/guide/components/processes-and-thread`

[6] Dianne Hackborn. Google+ post, dec 9 2011. `https://plus.google.com/u/0/105051985738280261832/posts/XAZ4CeVP6DC`.

[7] Robert Love. *Linux Kernel Development*. Pearson Education, Inc., third edition, 2010.

[8] StackOverflow. Android process scheduling. `http://stackoverflow.com/questions/7931032/android-process-scheduling`.