The Android Linux Kernel (Part 3):
Android Kernel Extensions

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Lesson

1. Recognize the two types of storage supported by Android Linux
2. Understand Android Linux’s local & remote communication mechanisms
3. Know how Android Linux’s processes & threads mediate access to one or more processor cores
4. Recognize extensions that Android Linux adds to the GNU Linux kernel
Learning Objectives in this Part of the Lesson

1. Recognize the two types of storage supported by Android Linux
2. Understand Android Linux’s local & remote communication mechanisms
3. Know how Android Linux’s processes & threads mediate access to one or more processor cores
4. Recognize extensions that Android Linux adds to the GNU Linux kernel, e.g.
   - Manage memory & power effectively on mobile devices
Learning Objectives in this Part of the Lesson

1. Recognize the two types of storage supported by Android Linux
2. Understand Android Linux’s local & remote communication mechanisms
3. Know how Android Linux’s processes & threads mediate access to one or more processor cores
4. Recognize extensions that Android Linux adds to the GNU Linux kernel, e.g.
   - Manage memory & power effectively on mobile devices
   - Accelerate performance for local inter-process communication (IPC)
Android Linux Extensions:
Memory Management
• The Android Linux kernel manages primary storage efficiently for memory-constrained mobile devices
• GNU Linux kernel shields programmers from primary storage constraints
GNU Linux kernel shields programmers from primary storage constraints, e.g.

- Linux’s virtual memory manager allows applications to access an address space larger than RAM

See Part 1 of this lesson on “Overview of the Android Linux Kernel”
• GNU Linux kernel shields programmers from primary storage constraints, e.g.
  • Linux’s virtual memory manager allows applications to access an address space larger than RAM
  • Swapping is used to automatically exchange the contents of primary storage with secondary storage

See stackoverflow.com/a/4420560/3312330
The Android Linux virtual memory manager behaves differently than GNU Linux.
The Android Linux virtual memory manager behaves differently than GNU Linux.

- e.g., to conserve power Android Linux doesn’t provide swap space for RAM.

See developer.android.com/training/articles/memory.html#Android
The Android Linux virtual memory manager behaves differently than GNU Linux.

- e.g., to conserve power, Android Linux doesn’t provide swap space for RAM.
• Android Linux defines a “low memory killer” extension
Android Linux defines a “low memory killer” extension

• Terminates app components when available RAM falls below a given threshold

See www.programering.com/a/MjNzADMwATE.html
App developers must know Android Linux kernel storage extensions.
App developers must know Android Linux kernel storage extensions, e.g.

Properly implement Activity & Service lifecycle methods

See [www.vogella.com/tutorials/AndroidLifeCycle/article.html](http://www.vogella.com/tutorials/AndroidLifeCycle/article.html)
App developers must know Android Linux kernel storage extensions, e.g.

- Properly implement Activity & Service lifecycle methods
  - e.g., cleanup dynamically allocated resources in the onPause(), onStop(), & onDestroy() methods

See www.vogella.com/tutorials/AndroidLifeCycle/article.html
App developers must know Android Linux kernel storage extensions, e.g.

- Properly implement Activity & Service lifecycle methods
- Apply common Android idioms for managing app RAM

See [developer.android.com/training/articles/memory.html](http://developer.android.com/training/articles/memory.html)
App developers must know Android Linux kernel storage extensions, e.g.

- Properly implement Activity & Service lifecycle methods
- Apply common Android idioms for managing app RAM
- e.g., use long-running services sparingly & reduce size of Android package kits (APKs)

See developer.android.com/topic/performance/reduce-apk-size.html
App developers must know Android Linux kernel storage extensions, e.g.

- Properly implement Activity & Service lifecycle methods
- Apply common Android idioms for managing app RAM
- Apply common Android tools for managing app RAM

Tools for analyzing RAM usage

Before you can fix the memory usage problems in your app, you first need to find them. Android Studio and the Android SDK include several tools for analyzing memory usage in your app:

1. The Device Monitor has a Dalvik Debug Monitor Server (DDMS) tool that allows you to inspect memory allocation within your app process. You can use this information to understand how your app uses memory overall. For example, you can force a garbage collection event and then view the types of objects that remain in memory. You can use this information to identify operations or actions within your app that allocate or leave excessive amounts of objects in memory.

   For more information about how to use the DDMS tool, see Using DDMS.

2. The Memory Monitor in Android Studio shows you how your app allocates memory over the course of a single session. The tool shows a graph of available and allocated Java memory over time, including garbage collection events. You can also initiate garbage collection events and take a snapshot of the Java heap while your app runs. The output from the Memory Monitor tool can help you identify points when your app experiences excessive garbage collection events, leading to app slowness.

   For more information about how to use Memory Monitor tool, see Viewing Heap Updates.

See developer.android.com/topic/performance/memory.html#AnalyzeRam
App developers must know Android Linux kernel storage extensions, e.g.

- Properly implement Activity & Service lifecycle methods
- Apply common Android idioms for managing app RAM
- Apply common Android tools for managing app RAM
  - e.g., Android Studio tools

See developer.android.com/studio/profile/investigate-ram.html
Android’s anonymous shared memory (ashmem) kernel extension allows multiple processes to share memory.

See [elinux.org/Android_Kernel_Features#ashmem](elinux.org/Android_Kernel_Features#ashmem)
• Android’s anonymous shared memory (ashmem) kernel extension allows multiple processes to share memory.
• e.g., the Binder framework uses ashmem to optimize transfer of “blobs” by avoiding data copying.
• ashmem is targeted for mobile devices with limited RAM
ashmem is targeted for mobile devices with limited RAM, e.g.

Shared memory regions can be discarded when RAM runs low

See notjustburritos.tumblr.com/post/21442138796/an-introduction-to-android-shared-memory
ashmem is targeted for mobile devices with limited RAM, e.g.

- Shared memory regions can be discarded when RAM runs low
- Memory allocated by ashmem is released when the process that creates it exits

See notjustburritos.tumblr.com/post/21442138796/an-introduction-to-android-shared-memory
Android Linux Extensions: Power Management
Android Linux provides kernel extensions & specialized device drivers tailored for the needs of mobile apps & services.
• Android Linux provides kernel extensions & specialized device drivers tailored for the needs of mobile apps & services
  • e.g., mobile devices require kernel-level power management capabilities

See developer.android.com/training/monitoring-device-state
Android Linux Extensions: Power Management

- Android Linux provides kernel extensions & specialized device drivers tailored for the needs of mobile apps & services
- e.g., mobile devices require kernel-level power management capabilities
- Common problems include keeping the screen on too long or running the CPU for extended periods of time
- Android Linux’s PowerManagement kernel extensions control when a device sleeps & wakes

See elinux.org/Android_Power_Management
• Android Linux’s PowerManagement kernel extensions control when a device sleeps & wakes
• Android middleware manages app power consumption via these kernel extensions
• Android Linux’s PowerManagement kernel extensions control when a device sleeps & wakes
• Android middleware manages app power consumption via these kernel extensions
  • e.g., the PowerManager system service provides wake locks to apps that can’t sleep when they are idle

See developer.android.com/training/scheduling/wakelock.html
Android Linux Extensions: Power Management

- Android Linux’s PowerManagement kernel extensions control when a device sleeps & wakes
- Android middleware manages app power consumption via these kernel extensions
  - e.g., the PowerManager system service provides wake locks to apps that can’t sleep when they are idle

Examples include Google Maps navigation
Android Linux Extensions: Local Inter-Process Communication
The TCP/IP protocol stack is designed for remote communication
• The TCP/IP protocol stack is designed for remote communication
  • e.g., handles network congestion via “slow start” & exponential back-off algorithms

See en.wikipedia.org/wiki/TCP_congestion_control
TCP/IP incurs gratuitous overhead when used for IPC between processes on a mobile device:

- e.g., unnecessary checksums & code that handles network congestion

See bhavin.directi.com/unix-domain-sockets-vs-tcp-sockets
• Android’s Binder driver is optimized for IPC on the same device

See [elinux.org/Android_Binder](http://elinux.org/Android_Binder)
Android’s Binder driver is optimized for IPC on the same device

- e.g. AIDL is used to invoke “remote” calls

See developer.android.com/guide/components/aidl.html
- Android’s Binder driver is optimized for IPC on the same device
  - e.g. AIDL is used to invoke “remote” calls
• Android’s Binder driver is optimized for IPC on the same device
  • e.g. AIDL is used to invoke “remote” calls
Android’s Binder driver is optimized for IPC on the same device
- e.g. AIDL is used to invoke “remote” calls

Binder driver is written in C/C++ & runs in the kernel to enhance performance
Android’s Binder driver is optimized for IPC on the same device
• e.g. AIDL is used to invoke “remote” calls

Binder driver collaborates with higher-layer Binder framework written in Java
Android’s Binder driver is optimized for IPC on the same device

- e.g. AIDL is used to invoke “remote” calls

These layers of software simplify/optimize object-oriented local IPC
End of the Android Linux Kernel (Part 3): Android Kernel Extensions