3.1 MEMORY MANAGEMENT

DEFINITION

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free.

When mobile operating system are considered, memory management plays the key role. As the mobile devices are constrained about hardware part, special care has to be taken for the memory management.

Android operating system is based on the linux kernel which mainly has the paging system. The memory is categorizes as internal memory, swap memory, external memory etc.,

- The Android Runtime (ART) and dalvik virtual machine use paging and memory-mapping (mmapping) to manage memory.
- This means that any memory an app modifies—**whether by allocating new objects or touching mmapped pages—remains resident in RAM and cannot be paged out.**
- The only way to release memory from an app is **to release object references that the app holds,** making the memory available to the garbage collector.
- That is with one exception: any files mapped in without modification, such as code, can be paged out of RAM if the system wants to use that memory elsewhere.

LINUX KERNEL VS ANDROID OS

- Android OS is nearly similar to the Linux kernel. Android OS has enhanced its features by adding more custom libraries to the already existing ones in order to support better system functionalities. For example last seen first killed design, kill the least recently used process first.
- Memory management is the hardest part of mobile development. Mobile devices, from cheaper ones to the most expensive ones, have limited amount of dynamic memory compared to our personal computers.
- The basic facilities run by the kernel are process management, memory management, device management and system calls. Android also supports all these features.

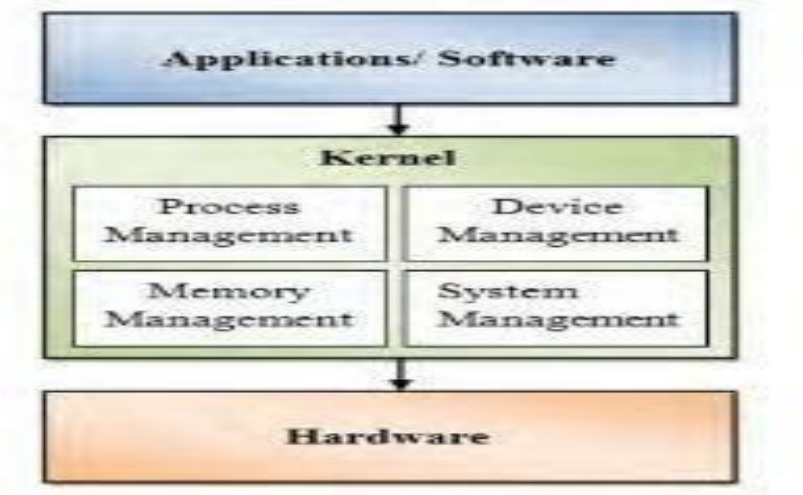


Fig 1: The Kernel Relation with Hardware and Applications

Poor memory management shows itself in various ways:

1. Allocating more memory space than application actually needs,
2. Not releasing the memory area retained by application,
3. Releasing a memory area more than once (usually as a result of multi-thread operations),
4. While using automatic memory solutions (ARC or GC), losing the tracks of your objects.

GARBAGE COLLECTION

- A managed memory environment, like the ART or Dalvik virtual machine, keeps track of each memory allocation.
- Once it determines that a piece of memory is no longer being used by the program, it frees it back to the heap, without any intervention from the programmer.
- *The Mechanism for reclaiming unused memory within a managed memory environment is known as garbage collection.*

Garbage collection has two goals:

1. Find data objects in a program that cannot be accessed in the future;
2. Reclaim the resources used by those objects.

Android's memory heap is a generational one, meaning that there are different buckets of allocations that it tracks, based on the expected life and size of an object being allocated. For example, recently allocated objects belong in the *Young generation*. When an object stays active long enough, it can be promoted to an older generation, followedby a permanent generation.

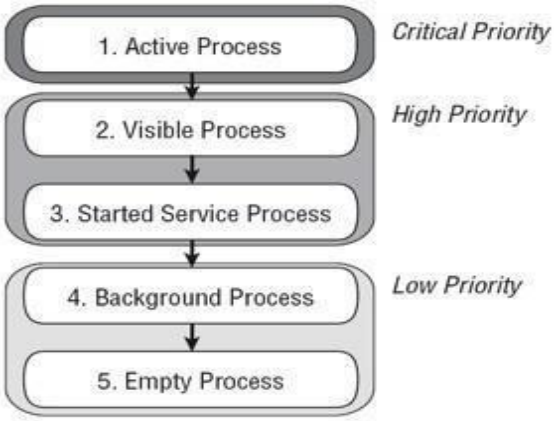
Each heap generation has its *own dedicated upper limit on the amount of memory* that objects there can occupy. Any time a generation starts to fill up, the system executes a garbage collection event in an attempt to free up memory. *The duration of the garbage collection depends on which generation of objects it's collecting and how many active objects are in each generation.*

- *The system has a running set of criteria for determining when to perform garbage collection. When the criteria are satisfied, the system stops executing the process and begins garbage collection.*
- If garbage collection occurs in the middle of an intensive processing loop like an animation or during music playback, it can increase processing time. This increase can potentially push code execution in your app past the recommended 16ms threshold for efficient and smooth frame rendering.

UNDERSTANDING APPLICATION PRIORITY AND PROCESS STATES

- The order in *which processes are killed to reclaim resources is determined by the priority of the hosted applications*. An application's priority is equal to its highest-priority component.

  Where two applications have the same priority, the process that has been at a lower priority longest will be killed first. **Process priority is also affected by interprocess dependencies; if an application has a dependency on a Service or Content Provider supplied by a second application, the secondary application will have at least as high a priority as the application it supports.**
- All Android applications will remain running and in memory until the system needs its resources for other applications.



Active Processes

- Active (foreground) processes are those **hosting applications with components currently interacting with the user.**
- These are the processes Android is trying to keep responsive by **reclaiming resources.**
  There are generally very few of these processes, and they will be killed only as a last resort.
- **Activities in an —active‖ state; that is, they are in the foreground** and responding to user events. You will explore Activity states in greater detail later in this chapter.
- Activities, Services, or Broadcast Receivers that are currently executing an onReceive event handler.
- Services that are executing an onStart, onCreate, or onDestroy event handler.

Visible Processes

- **Visible, but inactive processes are those hosting —visible‖ Activities**. As the name suggests, visible Activities are visible, but they **aren't in the foreground or responding to user events.**
-

Started Service Processes

- Processes hosting Services that have been started. **Services support ongoing processing that should continue without a visible interface.**
- Because Services don't interact directly with the user, they **receive a slightly lower priority than visible Activities.**
- They are **still considered to be foreground processes and won't be killed unless resources are needed for active or visible processes.**

Background Processes

- **Processes hosting Activities that aren't visible and that don't have any Services that have been started are considered background processes.**

- There will generally be a large number of background processes that Android will killusing a **last-seen-first-killed pattern** to obtain resources for foreground processes.

Empty Processes

- To improve overall system performance, Android often retains applications inmemory after they have reached the end of their lifetimes.
- Android maintains this cache to improve the start-up time of applications whenthey're relaunched. These processes are routinely killed as required.

DDMS

- Android Studio includes a debugging tool called the **Dalvik Debug Monitor Service (DDMS).** DDMS provides services like screen capture on the device, threading, heap information on the device, logcat, processes, incoming calls, SMS checking, location,data spoofing, and many other things related to testing your Android application.

- DDMS connects the IDE to the applications running on the device. On Android, every application runs in its own process, each of which hosts its own virtual machine (VM).
  And each process listens for a debugger on a different port.
- When it starts, DDMS connects to **ADB** (**Android Debug Bridge**, which is a command-line utility included with Google's Android SDK.).
- An Android Debugger is used for debugging the Android app and starts a device monitoring service between the two. This will notify DDMS when a device is connected
  or disconnected.
- When a device is connected, a VM monitoring service is created between ADB andDDMS, which will notify DDMS when a VM on the device is started or terminated.

-