BACHELOR OF COMPUTER APPLICATIONS

SEMESTER 4

DCA2203

SYSTEM SOFTWARE

Unit 13

Memory and Process Management in Android

Table of Contents

1. INTRODUCTION

The Android Operating System, which is based on the Linux kernel 2.6, and its architecture were explained to you in the previous course. The android architecture is layered with various types of libraries and that the application framework is a set of basic tools for developing more complex tools the application is a dot APK file, which contains the whole application package. The Android permission model is built around the fundamental idea of permission, which you have covered.

Mobile phones are preferred to be smaller in shape and size. As a result, its processor and memory are restricted by the processor speed and memory capacity required for effectiveness and performance. In this unit, we will learn how the Android system manages its processes. The idea of processes and threads, which are used in Android applications, will be explained.

1.1 Learning Objectives:

After studying this unit, learners should be able to:

- *Justify when the processes are terminated in the android system*
- *Explain how processes and threads are used in android systems*
- *Discuss about the inter-process communication in the android system.*

2. UNDERSTANDING APPLICATION PRIORITY AND PROCESS STATES

Android Operating system are generally used for mobile handheld devices, which are designed to be smaller in shape and size. However, it should be energy efficient and provide better performance. Android should be able to manage applications, processes, and memory optimally to show its efficiency and performance to the user.

2.1 Application

In the Android architecture, applications can be a native application (like SMS programs, contacts, browsers, calendars etc.) or a third-party applications (like games, dictionaries etc.), which are built on the application layer using the same API libraries. The application layer runs within the Android run time using the classes and services made available from the application framework. Android apps may activate components of any other app if the other app allows it. Applications in Android OS consist of four kinds of components to provide entry points (an entry point is a point in a program, module, or function where the code begins in computer programming) for other apps. They are:

i) Activity
ii) Service
iii) Broadcast receiver
iv) Content provider.

i) **Activity:** It is a focused visual user interface. All the activities are maintained in a "task" stack. Activity that is running are at the top position. Activities always remain together in a stack and share priority. When all the activities are over, the stack can be cleared.

ii) **Service:** A task runs in the background without user interaction. Service runs in the background for an indefinite amount of time. Service persists in Android OS until it either finishes or has no active connections. Service can be stopped under extremely low memory conditions.

iii) **Broadcast receiver:** A broadcast receiver is a component that responds to an external stimulus. It receives and reacts to broadcast messages. It runs for a deterministic amount of time, and after time runs out, it is placed into the background to be killed. It is considered to have foreground priority for its short duration.

iv) **Content provider:** It is a set of data wrapped up in a custom API to read and write. It makes data available to other applications (no lifetime)

Note that all these components are the application's entry points – it has no main() functions. An application and its main components occur in a single process. An application may call components from other applications in their processes. Android usually does not kill an app. Android kills apps when the memory usage goes too high, but it saves the app state for a quick restart later. Note that Low-priority processes are killed as memory is needed. The priority of the hosted applications determines the processes to be killed to reclaim resources. An application's priority is equal to its highest-priority component. The process that has been at a lower and longest priority will be killed first when two applications have the same priority. Process priority is also affected by inter-process dependencies. If an application depends 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 remain running and are in memory until the system needs the resources for other applications.

Active processes using examples:

- They are additionally known as front-end processes.
- Currently, certain processes of a specific application interact with the user.
- Resources are given to these processes because they need to be active.

Examples are as follows:

- Texting using Whatsapp, and is said to be in the foreground
- It will be given the highest priority
- Only if insufficient memory to support its process is it killed.

Visible Processes have partial activity and lower priority than foreground processes:

- They are not an active process
- They are not interacting with users
- Ex
-]treme measures will be taken to kill them to liberate resources.

Examples are:

- The onPause()
- Putting an end to these processes could negatively impact user experience.

Let's learn about started services process:

- These are the running processes for the started services.
- The services do not require visible interfaces.
- They are less important than processes that are active or visible.
- Such as Bluetooth services, antivirus software, and background file uploading

SELF ASSESSMENT QUESTIONS – 1

1. The application layer runs within the Android run time using the classes and services made available from the" _____" framework.
2. Which of the following components of applications is a focused visual user interface?
   a) Activity
   b) Service
   c) Broadcast receiver
   d) Content provider
3. Which of the following components of applications is a set of data wrapped up in a custom API to read and write and to make data available to other applications?
   a) Activity
   b) Service
   c) Broadcast receiver
   d) Content provider

2.2 Processes and Threads

A process is a sequential program that is being executed, or you might say that a process is a program that is being executed. When an application's component in Android starts and the application does not have any other components running,, the Android system starts a new Linux process for the application with a single thread of execution.

Thread is a sequential execution stream within a process. Threads in a process share the same address space.

In the Android system, by default, all components of the same application run in the same process and the "main" thread. Suppose an application component starts and there already exists a process for that application (because another component from the application exists). In that case, the component is started within that process and uses the same thread of execution. However, you can set up distinct parts of your program to work independently in multiple processes and add extra threads to any process.

2.2.1 Processes

All components of the same application run in the same process by default; most applications are not supposed to change this. However, change the manifest file if needed to control which process a certain component belongs to.

In the manifest file, each type of component element like <activity>, <service>, <receiver>, and <provider>supports an ***android: process*** attribute. This attribute can specify a process in which that component should run. This attribute can be set so that each component runs in its process or that some components share a process while others do not. ***android: process*** can also be set so that components of different applications run in the same process, but the applications should share the same (Linux) user ID with the same certificates signed. The <application>element also supports an ***android: process*** attribute. It sets a default value that applies to all components.

In the case when memory is low and required by other processes that the user more immediately requires, Android may have to end a process. When the application components running in the process are terminated, then they are consequently destroyed

too. A process is started again to complete the work of those components that are ended without completion.

The Android system judges the relative importance of the process to the user application before they are terminated. For example, process hosting activities are terminated if they are no longer visible on screen, compared to process hosting visible activities. Hence, a judgment to terminate a process depends on the state of the components running in that process.

Process lifecycle

An Android activity is a self-standing component of an Android application that can be started, stopped, paused, restarted, or reclaimed depending on various events, including user-initiated and system-initiated ones. Figure 13.1 shows the process lifecycle in detail.

onCreate()

The activity's life cycle starts with this method. When onCreate is called, an activity may be in one of three states. The activity may have been a brand-new activity starting out its life for the first time. Alternatively, it might be a process that needs to be restarted automatically due to a configuration change, such a device rotating from one direction to another. Or it is an activity restarted following a previous process shutdown due to low-memory conditions and being in the background. In the onCreate callback, you should consider these scenarios if what you need to do in each scenario is different.

onStart()

After being created, this method pushes the activity into a visible state. In other words, this method starts the visible life cycle of the activity. This method is called right after onCreate(). This method assumes that the view hierarchies are loaded and available from the onCreate().

onRestoreInstanceState()

If the system chooses to close the activity because there is a change in orientation, then the user will expect that transitory (instance) state right back when the activity is restarted. To facilitate this, Android calls theonRestoreInstanceState method with a bundle that contains the saved instance state.

onResume()

The callback method onResume is the precursor to having the activity fully visible. This is also the start of the foreground cycle for the activity. In this foreground cycle, the activity can move between onResume() and onPause() multiple times as other activities, notifications, or dialogues with more urgency come on top and go.

onPause()

This callback indicates that the activity is about to go into the background. It would be best if you stopped any counters or animations that were running when the activity was fully visible. The activity may go to onResume or proceed to onStop. Going to onResume will bring the activity to the foreground. Going to onStop will take the activity into a background state.

onStop()

The callback method onStop() moves the activity from partially visible to the background state while keeping all view hierarchies intact. This is the counterpart of onStart. The activity can be taken back to the visible cycle by calling onStart. This state transition from onStop to onStart during the same activity life cycle goes through the onRestart() method.

onSaveInstanceState(bundle saveStateBundle)

The control goes to onDestroy() coming out of onStop if the process is still in memory. However, if Android realizes that the activity is being closed without the user's expectation, it would call onSaveInstanceState() before calling onDestroy(). Orientation change is a very concrete example of this. The SDK warns that the timing of onSaveInstanceState() is not predictable, whether before or after onStop().

onRestart()

This method is called when the activity transitions from the background state to a partially visible state, i.e., going from onStop to onStart. You can use this knowledge in onStart if you want to optimize code there based on whether it is a fresh start or a restart. The view and their state are fairly intact when it is a restart. .

onDestroy()

onDestroy() is the counterpart ofonCreate(). The activity is going to finish afteronDestroy(). An activity can finish for two primary reasons. One is an explicit close. The second reason an activity can close is involuntary.

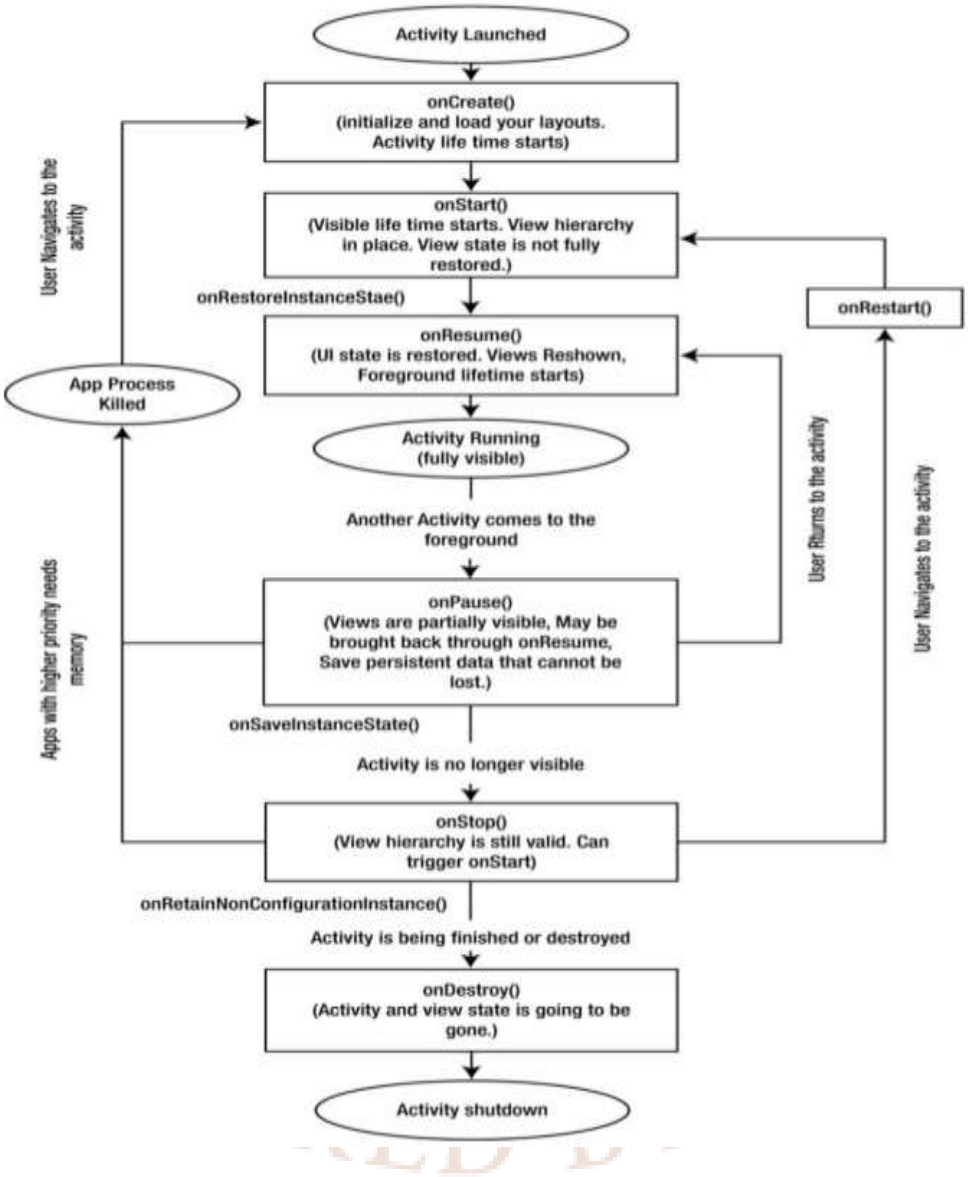

Fig 13.1: Process Life Cycle

The Android system tries to maintain the process of an application for as long as possible. However, it also must remove old processes so that new or more important processes can use memory. To determine which processes to keep and which to kill, the system places each process into an "importance hierarchy" based on the components running in the process and

the state of those components. Processes with the lowest importance are killed first, then those with the next lowest importance, and so on, as necessary to recover system resources.

There are five levels in the importance hierarchy. Figure 13.2 presents the different types of processes in order of importance, the application states of an application, and their priority. Note that the first process is the most important and is killed last. It is important to structure your application correctly to ensure it is on a priority basis for the work it is doing; otherwise. your application could be killed while it is in the middle of the process, which is very important.

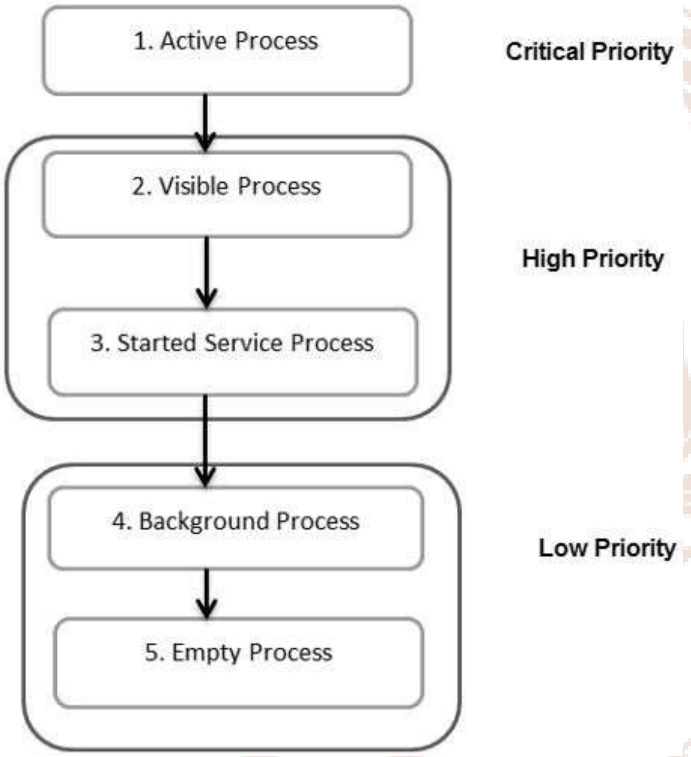

Fig 13.2: Application states

Now you will study how the application components determine the state:

1. **Active process:** This process is required for the user currently involved in some services. Android keeps this process responsive by reclaiming resources. Active processes include activities in an "active" state. This process is in the foreground and responds to user events.

Generally, only a few foreground processes exist at any given time. These processes are killed only at the last moment if memory is very low to run these processes. Generally, at that point,

the device reaches a memory paging state, and hence some foreground processes have to be killed to keep the user interface responsive.

A process is in the foreground if any of the following conditions are true:

- It hosts an Activity that the user is interacting with.
- It hosts a Service that is bound to the activity that the user is interacting with.
- It hosts a Service that's running "in the foreground."
- It hosts a Service that's executing one of its lifecycle callbacks.
- It hosts a Broadcast Receiver.

2. **Visible process:** This is a process that is visible but inactive. It hosts a "visible" activity. This process does not have any foreground components but can still affect what the user sees on screen. Visible activities are visible but are not in the foreground or do not respond to user events. There are generally very few visible processes, and are killed only in extreme circumstances to allow active processes to continue.

A process is visible if either of the following conditions is true:

- It hosts an activity not in the foreground but still visible to the user.
- It hosts a service bound to a visible (or foreground) activity.

A visible process is considered extremely important. It is killed only when all foreground processes are required for running. Otherwise it is not killed.

3. **Started service process:** This process processes hosting services that have been started. Services support ongoing processing that should continue without a visible interface. Although service processes are not directly tied to anything the user sees, they are generally doing things that the user cares about (such as playing music in the background or downloading data on the network), so the system keeps them running unless there is not enough memory to retain them along with all foreground and visible processes. Because services do not interact directly with the user, they receive a relatively lower priority than visible activities. They are still considered to be afore-ground processes and are not killed unless resources are needed for active or visible processes.
4. **Background process:** These kinds of processes carry out actions that are not visible and do not have any running services. This process holds an activity that is not

currently visible to the user. These processes have no direct impact on the user experience, and the system can kill them at any time to reclaim memory for a foreground, visible, or service process. Many background processes that are running are kept in an LRU (least recently used) list to ensure that the process with the activity that the user saw is the last to be killed. If an activity implements its lifecycle methods correctly and saves its current state, killing its process will not have a visible effect on the user experience because when the user navigates back to the activity, the activity restores all its visible state. Note that android kills the background process to obtain resources for foreground processes.

5. **Empty process:** This process does not hold any active application components. This process is kept alive for caching purposes to improve startup time when a component needs to run next time. To improve overall system performance, Android often retains applications in memory after they have reached the end of their lifetimes. Android maintains this cache to improve the start-up time of applications when they are restarted. The system often kills these processes to balance overall system resources between process caches and the underlying kernel caches. These processes are routinely killed as required.

Android ranks a process at the highest level based on the importance of the components currently active in the process. For example, if a process hosts a service and a visible activity then the process is ranked as a visible process, not a service process. The ranking of process increases if other processes are dependent on it. A process serving another process can never be ranked lower than the process it is serving. For example, if a content provider in process A is serving a client in process B, or if a service in process A is bound to a component in process B, then process A is always considered at least as important as process B.

SELF ASSESSMENT QUESTIONS – 2

4. In the manifest file, each type of component element like<activity>, <service>, <receiver>, and <provider>supports an"_____" attribute.
5. The Android keep the active process responsive by reclaiming resources. (True/False)
6. Which of the following "_____"process does not have any foreground components but can still affect what the user sees onscreen?
   a) Active process
   b) Visible process
   c) Started service process
   d) Background process

2.2.2 Threads

A thread is a process without its own address space in memory. Thread shares the address space with the parent process. Processes can have child threads assigned to a process.

In Android, when an application is launched, the system creates a thread of execution for the application called "main." This thread acts as in charge of dispatching events to the appropriate user interface widgets, including drawing events. It is also the thread in which your application interacts with components from the Android User Interface (UI) toolkit (components from the android. Widget and android. view packages). The main thread is also sometimes called the UI thread. This type of Android system represents a single-thread model. The system does not create a separate thread for each component instance. All components that run in the same process are instantiated in the UI thread, and system calls to each component are dispatched from that thread. Consequently, methods that respond to system callbacks always run in the UI thread of the process. For example, when the user touches a button on the screen, the UI thread of the apps dispatches the touch event to the widget, which in turn sets its pressed state and posts an invalid request to the event queue. The UI thread dequeues the request and notifies the widget that it should redraw itself.

Your applications using a single-thread model may need to run better. Hence it should be implemented properly. Performing long operations such as network access or database queries will block the UI if everything is happening in the UI thread . No events can be

dispatched when the thread is blocked , including drawing events. From the user's perspective, the application appears to hang. If the UI thread is blocked for more than a few seconds (nearly about 5 seconds), the user is responded with the "application not responding" (ANR) dialog. The user might then decide to quit your application and uninstall it if they are unhappy.

Additionally, the Android UI toolkit is not thread safe. A code is thread-safe if it functions correctly during simultaneous execution by multiple threads. Therefore, must not manipulate your UI from a worker thread (explained below). All user interface manipulation must be done from the UI thread. Thus, there are simply two rules to Android's single-thread model to make any implementation thread-safe:

1. Do not block the UI thread.
2. Do not access the Android UI toolkit outside the UI thread.

Worker threads

In a single-thread model, you should not block the UI thread; otherwise, this can result in undefined and unexpected behavior. So, any long operations or operations that are not instantaneous should go in separate threads. Those threads can be regarded as "background" or "worker" threads. Oneshould also not access the Android UI toolkit from outside the UI thread. Android offers several ways to access the UI thread from other threads. It to maintain the kind of code that becomes complex as the operation grows. . Use a handler in your worker thread to process messages sent by the UI thread in order to manage more complicated interactions with a worker thread.. Perhaps the best solution is to extend the ***AsyncTask*** class, which simplifies the execution of worker thread tasks that need to interact with the UI.

Using AsyncTask

Work on the user interface can be done asynchronously thanks to the Async Task class. It performs the blocking operations in a worker thread and then publishes the results on the UI thread, without requiring to handle threads and/or handlers yourself.

Thread-safe methods

In some situations, the methods you implement might be called from more than one thread, and therefore methods used must be written to be thread- safe. That is to say that the methods should function correctly during simultaneous execution of multiple threads.

This is primarily true for methods that can be called remotely—such as methods in a bound service. There are few threads safe methods like query(), insert(), delete(), update(), and getType() that can be implemented to be thread afe.

SELF ASSESSMENT QUESTIONS – 3

7. In Android system, when an application is launched, the system creates a thread of execution for the application, called"_____".
8. In a single thread model, any long operations or operations that are not instantaneous should go in separate threads regarded as _____Or"_____"_____threads.
9. You can operate asynchronously on your user interface using"_____" class.

2.3 Inter-process Communication

Processes communicate with each other and with the kernel to coordinate their activities. If one process exchanges data with another process, it is called Inter-process Communication (IPC). Android supports several Inter-Process Communication (IPC) mechanisms using Remote Procedure Calls (RPCs). In Android systems, IPC is a set of methods for exchanging data among multiple threads in one or more processes. IPC methods are divided into message passing, synchronization, shared memory, and RPC. A reason for allowing process communication is to share information, speed, modularity, and security.

The basic idea of RPC is that programs call functions or methods on remote programs (identified by program numbers), and the remote programs return a result code or message. Since different processes cannot access each other's address space, methods on remote objects are called on proxy objects provided by the system. These proxy objects decompose (marshal) the past arguments and hand them over to the remote process. The method call is then executed within the remote app component and the results marshaled are returned to

the calling process. The app programmer only defines and implements the interface. The system generates the entire RPC functionality based on the defined interface and transparent to the application. Interfaces for inter-process communication are defined using the Android interface definition language (AIDL). To perform IPC, theapplication must bind to a service using the bind Service () method.

IPC is an inter-process communication that describes the mechanisms used by different types of android components to communicate with one another.

1) **Intents** are messages which components can send and receive. It is a universal mechanism of passing data between processes. With the help of the intents, one can start services or activities, invoke broadcast receivers, and so on.
2) **Bundles** are entities of data that are passed through. It is similar to the serialization of an object but much faster on Android. Bundles can be read from intents via the getExtras() method.
3) **Binders** are the entities that allow activities and services to obtain a reference to another service. It allows not simply sending messages to services but directly invoking methods on them.

SELF ASSESSMENT QUESTIONS – 4

10. Android supports several Inter-Process Communication (IPC) mechanisms using""_____".
11. Toperform IPC, your application must bind to a service using _____Method.

3. SUMMARY

Let us recapitulate the important concepts discussed in this unit:

- In the Android architecture, applications can be anative applications or a third-party applications which are built on the application layer using the same API libraries.
- An application and its main components occur in a single process.
- Android usually does not kill an app. Android kills apps when the memory usage goes too high, but it saves the app state for a quick restart later.
- In the Android system, by default, all components of the same application run in the same process and the “main” thread.
- The Android system judges the relative importance of the process to the user application before they are terminated.
- To determine which processes to keep and which to kill, the system places each process into an "importance hierarchy" based on the components running in the process and the state of those components.
- The simple rules to Android's single thread model to make any implementation thread-safe is (i)Do not block the UI thread (ii) Do not access the Android UI toolkit from outside the UI thread.
- Android supports several Inter-Process Communication (IPC) mechanisms using remote procedure calls (RPCs).

4. GLOSSARY

- **Background process:** The type of process that host activities that are not visible and do not havSe any services that have been started.
- **Inter-process communication (IPC):** It is a mechanism that allows processes to communicate with each other and synchronize their actions.
- **Life cycle methodology:** The use of any one of several structured methods to plan, design, implement, test, and operate a system from its conception to the termination of its use.
- **Multithreading:** Used for programming and execution models that enable several threads to live within the context of a single process, independently and concurrently in multitasking operating systems.
- **Process:** A process is sequential program execution.
- **Software life cycle:** Period beginning when a software product is conceived and ending when the product is no longer available. The software life cycle is typically broken into phases denoting requirements, design, programming, testing, installation, and operation and maintenance.
- **Thread:** It is a sequential execution stream within a process or an execution of smallest sequence of programmed instructions.

5. TERMINAL QUESTIONS

Short Answer Questions

1. Write short notes about the different kinds of components of applications in Android OS.
2. Explain the processes of an application state of an Android System.
3. Write short notes on: (i) Worker threads (ii) Thread-safe method

5.1 Answers

Self-Assessment Questions

1. application
2. Activity
3. (d) Content provider
4. android: process
5. True
6. (b) Visible process
7. "main"
8. "background","worker"
9. AsyncTask
10. Remote Procedure Calls (RPCs)
11. bindService()

Short Answer Questions

1. Applications in Android OS consist of four kinds of components. They are: (i) Activity (ii) Service (iii) Broadcast receiver (iv) Content provider. (Refer to sub-section 2.1 for more details)
2. The various processes of an application are: Active process, visible process, started service process, background process, and empty process. (Refer to sub-section 2.2.1 for more details)
3. Any long operations or operations that are not instantaneous should go in separate threads regarded as "background" or "worker" threads. Methods that are implemented should function correctly during the simultaneous execution of multiple threads. (Refer to sub-section 2.2.2 for more details)

6. REFERENCES

- Processes and Threads. (n.d.). Retrieved 10 01, 2012, from developer.android.com: http://developer.android.com/guide/components/processes-and-threads.html
- Welcome to Mobile World. (n.d.). Retrieved from mobworld.wordpress.com: http://mobworld.wordpress.com/2010/07/05/memory-management-in-android/