5.3 MESSAGING AND COMMUNICATION MECHANISMS

In this section, we will introduce two mechanisms used in Android, including message and communication mechanisms.

5.3.1 Message Mechanisms

Android provides message mechanisms in three core classes: Looper, Handler, and Message. Similar to some other operating system, there is a Message Queue in Android. However, this Message Queue is packaged in Looper class. This class is mainly used to run a message loop for a thread. Threads by default do not have a message loop associated with them. We can call the prepare() method in the thread that is to run the loop, and then call loop() to process the message queue. The main function of prepare() method is defining the Looper object as a ThreadLocal object. After calling the loop() method, the Looper thread begins to work, and it continually processes the first message in the message queue. The working mechanism of the prepare() and loop() method, are shown in Fig. 5.18.

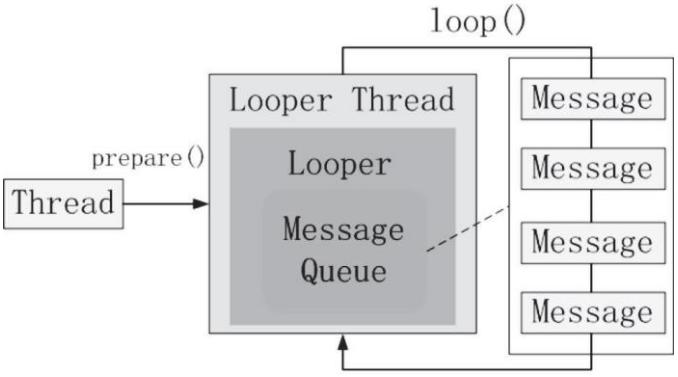

Figure 5.18 prepare() and loop() methods.

Furthermore, we will introduce how to add a message into the message queue. In Android, a Handler allows us to send and process Message and Runnable objects associated with the thread's Message-Queue. Each Handler instance is associated with a single thread and that thread's message queue. A Handler has two main functions: (1) to schedule messages and runnable to be executed as some point in the future, and (2) to enqueue an action to be performed on a different thread than our own.

Fig. 5.19 shows the process of adding a message into the message queue using Handler. First, Handler creates a message. Then, find the

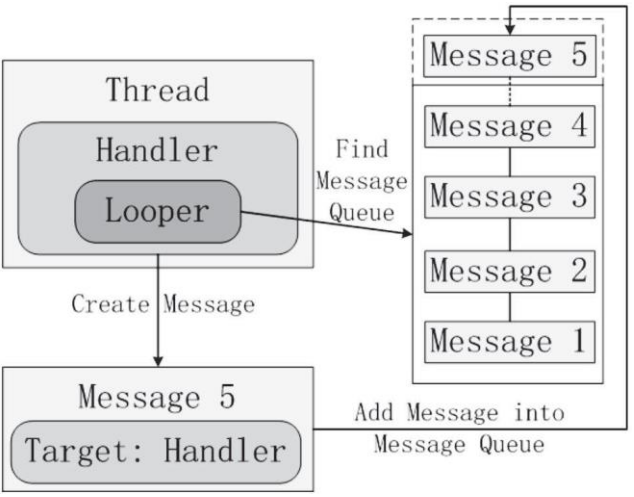

Figure 5.19 Use handler to add message into Message Queue.

related message queue based on the looper. Then add the new message to the end of the message queue.

Message class in Android defines a message containing a description and arbitrary data object that can be sent to a Handler. Although the constructor of Message is public, the best way to get one of these is to call *obtain()* method or *obtainMessage()* method of Handler to save resource costs.

5.3.2 Communication Mechanisms

Android provides the process-unit component model. All Android operations are expressed as Linux processes eventually. Android runs based on the Linux Kernel, and the memory, process, and file management

are controlled by the Linux Kernel. The system service is isolated by Linux processes for protection. To support mobile devices, all the default system functions of Android are provided as the server processes. Meanwhile, the functions realized by apps belong to application processes.

In Android, the server process and the application process are implemented by the class *Binder*. Binder is the most important part in Android, and it is the core part of a lightweight Remote Procedure Call (RPC) mechanism. We can derive directly from Binder to implement our own custom RPC protocol or simply instantiate a raw Binder object directly to use a token that can be shared across processes. RPC is a form of Inter Process Communication (IPC), which is a set of techniques for the exchange of data among multiple threads in one or more processes.

Android Interface Definition Language (AIDL) allows us to define the programming interface that both the client and service agree upon in order to communicate with each other using IPC. Normally, one process cannot access the memory of other processes. Processes need to decompose their objects into primitives that the operating system can understand and configure the objects across that boundary.

All system functions of Android are provided as a server process,

which makes the optimized communication method between processes extremely important. *Binder* refers to Kernel memory that is shared between all processes to minimize the overhead caused by memory copy. Furthermore, the RPC framework provided by Binder is written in C++, which is more efficient than Java.

In the RPC framework, the kernel space is a place where all processes can share and let each process refer to the memory address. In the Kernel space, a *Binder Driver* is implemented to use the kernel space to convert the memory address that each process has mapped with the memory address of the kernel space for reference. The Binder Driver supports the system call Input/Output Control (ioctl) and the file operations, including open, map, release, and poll. In computer science, ioctl is a system call for device-specific input/output operation and other operations which cannot be expressed by regular system calls.

Fig. 5.20 shows an example of the process of transmitting data from process A to process B. The first thing process A must do is to open the Binder kernel module, and this module uses the descriptor to identify the initiators and recipients of Binder IPCs. After defining the transmission (process A) and the reception (process B) of this operation, process A transmits the data to the Binder Driver first. Then,

the Binder Driver converts the memory address of the data to allow process B to access it.

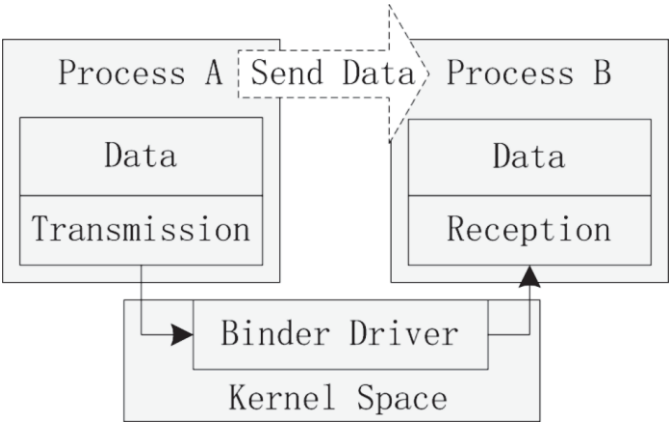

Figure 5.20 Transmit data via Binder Driver in RPC framework.