Android - Interprocess communication

Innopolis University
Course of Operating Systems

22nd September 2020

Sources

- These slides have been adapted from the original slides of the adopted book:
    - Tanenbaum & Bos, Modern Operating Systems: 4th edition, 2013 Prentice-Hall, Inc.

  and customised for the needs of this course.
- Additional input for the slides are detailed later

Content of the lecture

- Revising IPC and RPC
- Introduction to IPC in Android
- Android AIDL (Android Android Interface Definition Language)
    - Defining AIDL interface
        - Creating the .aidl file
        - Implementing the interface
        - Exposing the interface to clients/applications.
- Parcelable objects (Android 5.0)
- Reference list

Revising IPC

- Inter-process communication (IPC) is a mechanism which allows processes to communicate with each other and synchronize their actions.

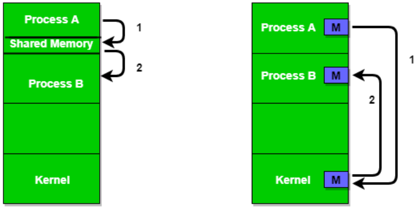
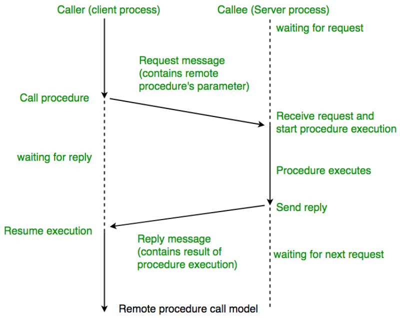


Figure 1 - Shared Memory and Message Passing

Revising RPC

- Remote Procedure Call (RPC) is a powerful technique for constructing distributed, client-server based applications. It is based on extending the conventional local procedure calling so that the called procedure need not exist in the same address space as the calling procedure.


Caller (client process)
Callee (Server process)
waiting for request
Request message (contains remote procedure's parameter)
Call procedure
Receive request and start procedure execution
waiting for reply
Procedure executes
Send reply
Resume execution
Reply message (contains result of procedure execution)
waiting for next request
Remote procedure call model

Introduction to IPC in Android

- Android's system design revolves significantly around process isolation, between applications as well as between different parts of the system itself by providing different levels of working with IPC

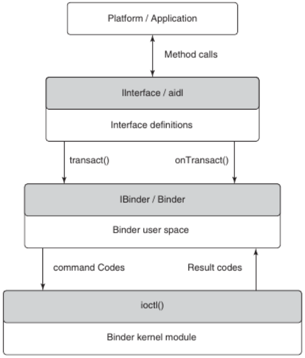


Figure 10-42. *Binder* IPC architecture.

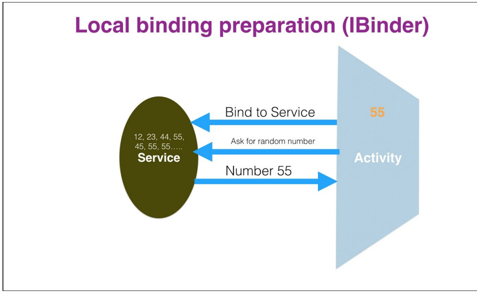
ioctl Unix

- In Unux, ioctl is mostly used for communicating on low-level with device drivers.

```
void ioctl_get_msg(int file_desc)
{
  int ret_val;
  char message[100];

  /* Warning - this is dangerous
     because we don't tell
   * the kernel how far it's allowed
       to write, so it
   * might overflow the buffer.
   */
  ret_val = ioctl(file_desc,
      IOCTL_GET_MSG, message);

  if (ret_val < 0) {
    printf("ioctl_get_msg failed:%d\n
        ", ret_val);
    exit(-1);
  }

  printf("get_msg message:%s\n",
      message);
}
```

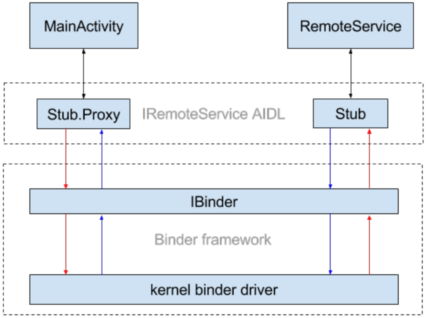
Binder/IBinder Android

- Base interface for a remotable object, the core part of a lightweight remote procedure call mechanism designed for high performance when performing in-process and cross-process calls.



Android Interface Definition Language

- Android Interface Definition Language (AIDL) allows to define the programming interface that both the client and service agree upon in order to communicate with each other using interprocess communication (IPC).

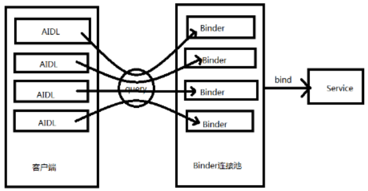


Binder IPC architecture.

Defining an AIDL interface

Steps to create a bounded service using AIDL:

1. Create the .aidl file
2. Implement the interface
3. Expose the interface to clients



1. Create the .aidl file

By default, AIDL supports the following data types:

- All primitive types in the Java programming language
- Arrays of primitive types such as int[]
- String
- CharSequence
- List
- Map

```
 // IRemoteService.aidl
package com.example.android;

// Declare any non-default types here
     with import statements

/** Example service interface */
interface IRemoteService {
    /** Request the process ID of this
          service, to do evil things
         with it. */
    const int VERSION = 1;
    int getPid();

    /** Demonstrates some basic types
         that you can use as parameters
     * and return values in AIDL.
     */
    void basicTypes(int anInt, long
         aLong, boolean aBoolean, float
          aFloat,
            double aDouble, String
                 aString);
}
```

2. Implement the interface

- To implement the interface .java generated from the .aidl, extend the generated Binder interface (for example, YourInterface.Stub) and implement the methods inherited from the .aidl file.

```
private final IRemoteService.Stub
     binder = new IRemoteService.Stub()
      {
     public int getPid(){
         return Process.myPid();
     }
     public void basicTypes(int anInt,
          long aLong, boolean aBoolean,
         float aFloat, double aDouble,
              String aString) {
         // Does nothing
     }
};
```

- To expose the interface for your service, extend Service and implement onBind() to return an instance of your class that implements the generated Stub

```
public class RemoteService extends
     Service {
    @Override
    public void onCreate() {
        super.onCreate();
    }

    @Override
    public IBinder onBind(Intent
         intent) {
        // Return the interface
        return binder;
    }

    private final IRemoteService.Stub
         binder = new IRemoteService.
         Stub() {
        public int getPid(){
            return Process.myPid();
        }
        public void basicTypes(int
             anInt, long aLong, boolean
              aBoolean,
            float aFloat, double
                 aDouble, String
                 aString) {
            // Does nothing
        }
    };
}
```

Creating the service based on .aidl interface

- Now, when a client (such as an activity) calls bindService() to connect to this service, the client's onServiceConnected() callback receives the binder instance returned by the service's onBind() method.

```
IRemoteService iRemoteService;
private ServiceConnection mConnection
    = new ServiceConnection() {
    // Called when the connection with
        the service is established
    public void onServiceConnected(
        ComponentName className,
        IBinder service) {
        // Following the example above
            for an AIDL interface,
        // this gets an instance of
            the IRemoteInterface,
            which we can use to call
            on the service
        iRemoteService =
            IRemoteService.Stub.
            asInterface(service);
    }

    // Called when the connection with
        the service disconnects
        unexpectedly
    public void onServiceDisconnected(
        ComponentName className) {
        Log.e(TAG, "Service␣has␣
            unexpectedly␣disconnected"
            );
        iRemoteService = null;
    }
};
```

Passing objects over IPC

- In Android, you can define Parcelable objects directly in AIDL. If custom accessors or other functionality is desired, Parcelable should be implemented instead.

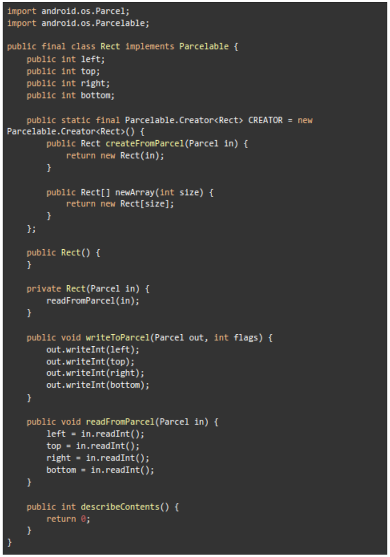
```
package android.graphics;

// Declare Rect so AIDL can find it
    and knows that it implements
// the parcelable protocol.
parcelable Rect {
    int left;
    int top;
    int right;
    int bottom;
}
```

Implement the Parcelable interface

To create a custom class that supports the Parcelable protocol, you must do the following:

- Make your class implement the Parcelable interface.
- Implement writeToParcel, which takes the current state of the object and writes it to a Parcel.
- Add a static field called CREATOR

Here is an example of how the Rect class implements the Parcelable protocol.



```
import android.os.Parcel;
import android.os.Parcelable;

public final class Rect implements Parcelable {
    public int left;
    public int top;
    public int right;
    public int bottom;

    public static final Parcelable.Creator<Rect> CREATOR = new
Parcelable.Creator<Rect>() {
        public Rect createFromParcel(Parcel in) {
            return new Rect(in);
        }

        public Rect[] newArray(int size) {
            return new Rect[size];
        }
    };

    public Rect() {
    }

    private Rect(Parcel in) {
        readFromParcel(in);
    }

    public void writeToParcel(Parcel out, int flags) {
        out.writeInt(left);
        out.writeInt(top);
        out.writeInt(right);
        out.writeInt(bottom);
    }

    public void readFromParcel(Parcel in) {
        left = in.readInt();
        top = in.readInt();
        right = in.readInt();
        bottom = in.readInt();
    }

    public int describeContents() {
        return 0;
    }
}
```

List of references

References:

- Android documentation
  https://developer.android.com/guide/components/aidljava
- AIDL (Android Interface Definition Language) (IPC)
  https://habr.com/ru/post/139432/ (russian)
- Talking To Device Files
  https://tldp.org/LDP/lkmpg/2.4/html/c854.html

End of lecture