International Journal of Hybrid Information Technology
Vol.7, No.3 (2014), pp.401-412
http://dx.doi.org/10.14257/ijhit.2014.7.3.38

Analysis on Process Code schedule of Android Dalvik Virtual Machine

Wen Hu, Yanli Zhao

School of Computer and Information Engineering
Harbin University of Commerce
Harbin, China
huwen1957@126.com

Abstract

Dalvik virtual machine's performance determines the performance of the Android platform. And each Dalvik virtual machine process is a Linux process, and each Dalvik virtual machine thread is a Linux thread. This paper analyses and researches process model of Android Dalvik virtual machine. Through research and analysis the role of Zygote in startup process of Dalvik virtual machine and the Zygote boot process, we analyze the relationship between the processes of the operating system and the Dalvik processes and we study the relationship between the Dalvik virtual machine processes. It answers the question why a Dalvik virtual machine process is actually a process of Linux.

Keywords: *Dalvik virtual machine; process; Android.*

1. Introduction

With the development of mobile network, mobile devices can develop rapidly. In the field of mobile devices, virtual machine technology plays an important role. Dalvik virtual machine is a Java virtual machine and Dalvik virtual machine is an important component of the Android platform. It is responsible for running applications of the Android platform, providing multi-threading support and memory garbage collection on Android platform [1]. And Dalvik virtual machine can optimize applications of Android platform in each step of executing applications. Instruction set of Dalvik virtual machine is based on registers. Virtual registers store operands, instructions from the virtual machine register get operands and calculate in the process of execution, and finally the data is back to the register [2]. Dalvik virtual machine is based on the Linux kernel, and it uses Zygote process model which is similar to fork process of Linux [3].

The Android platform can be divided into 4 layers, which are application layer, the application framework layer, library layer and the Linux kernel. Dalvik virtual machine is running on the Android runtime layer. Every Android program is executed in the Dalvik virtual machine. If the program needs to use the library content, it need to call the local program interface, and executive function in the library.

2. Problem analysis and related research

2.1. Problem analysis

We can discuss the virtual machine from the perspective of process and system point of view, the virtual machines can be divided into two types: system virtual machines and process virtual machines. System virtual machine owns a complete multi-process system environment. Multiple heterogeneous operating systems can be run on the system virtual machines [4]. In the process virtual machine, the ABI (Application binary interface) interface that runs on the operating system is replaced by virtualization software. The process virtual machine simulates a user-level instruction set and system calls. Dalvik virtual machine of the Android operating system belongs to the process virtual machine. Dalvik virtual machine is derived from Java virtual machine and is mainly used in the embedded system. Aiming at the characteristics of embedded system, it makes a great improvement in the compilation method, memory usage and stack usage [5]. So Dalvik has the feature of high efficiency, simple, save resources, and it is very suitable for limited resources of embedded systems.

Dalvik virtual machine's memory management and process management are dependent on the Linux kernel. Dalvik virtual machine makes the best of the Linux process management's features, so it can run multiple processes simultaneously [6]. And every Android application runs in a Dalvik virtual machine instances, and each virtual machine instance is an independent process space. Different applications run in different processes space, and the application of different sources use different Linux user to run. The way is the greatest degree to protect application secure and run independently.

Independent processes can prevent all programs from closing when virtual machine collapses. Each Dalvik virtual machine process is a Linux process, and each Dalvik virtual machine thread is a Linux thread [7]. The relationship between Dalvik virtual machine's process management and Linux process management is shown in Figure 1. It indicates Dalvik virtual machine's position in the process communication.

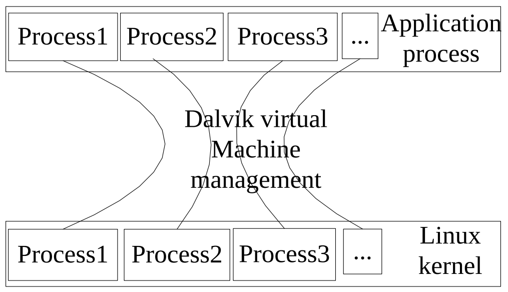

Figure 1. The relationship between Dalvik virtual process and Linux process

2.2. Related research

The Dalvik virtual machine directly provides own unique API on the basis of the Java API and most are achieved by calling the Linux system interface. And most of functions are achieved by calling the operating system interface of target machine, that is, them are realized by calling the interface of the Linux system. For example, when we call start function in android.os.Process class to create a process, the system will eventually call fork system to create a process. And Linux system provides fork system. Every Android application process has a Dalvik virtual machine instance. So an Android application process terminates unexpectedly, will not affect the normal operation of other Android application process.

When Android system is startup, Dalvik virtual machine has been also started, and a system service process is created by the fork semantics. Dalvik virtual machine processes correspond to the processes of the operating system. When the Android system performs an application, Zygote will use fork mechanism of Linux to produce a child process to perform the application. Not only we can create Android application process quickly, but also all of the Android Application processes share the same set of core Java libraries which is beneficial to save memory space [8]. As shown in Figure 2 the relationship between Dalvik virtual machine between processes.

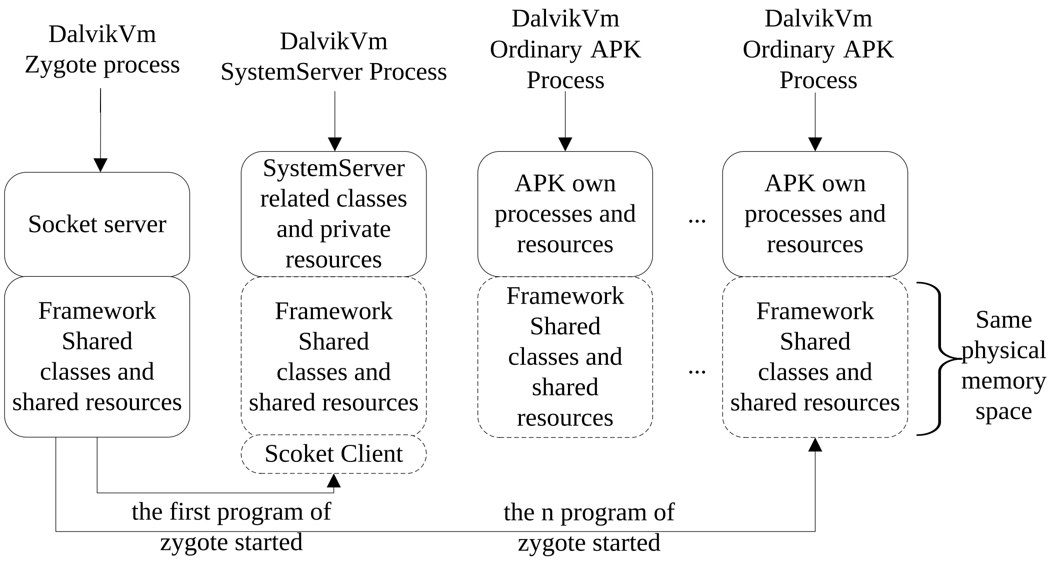
DalvikVm
Zygote process
DalvikVm
SystemServer Process
DalvikVm
Ordinary APK
Process
DalvikVm
Ordinary APK
Process
Socket server
SystemServer related classes and private resources
APK own processes and resources
...
APK own processes and resources
Framework Shared classes and shared resources
Framework Shared classes and shared resources
Framework Shared classes and shared resources
...
Framework Shared classes and shared resources
Same physical memory space
Scoket Client
the first program of zygote started
the n program of zygote started

Figure 2. The relationship between Dalvik virtual machine between processes

In figure 2, we know that through the Zygote in the Android system, Dalvik virtual machine processes are hatched out. All of system service process SystemServer and the application processes are bred by the Zygote process. So the zygote process can be treated as a generator of process. When applications in Dalvik virtual machine need to generate a new process, we can call zygote service. The zygote mainly consists of two modules: (1) Socket server. It is used to receive commands which start a new Dalvik process. (2) Shared classes and share resources in framework. When zygote process is started, it will load some shared classes and resources. Among them, shared classes are defined in preload-classes file, and shared resources are defined in preload-resources file.

3. Zygote startup and Dalvik startup

3.1. Zygote startup

Android system is based on the Linux kernel, and in the Linux system, all processes are descendant process of the init process. All processes are directly or indirectly fork out by the init process including Zygote process. When Zygote process starts, it will complete the initialization of the virtual machine, loading the library, loading the preset library and initialized operation and so on. And when the system needs a new virtual machine instance, Zygote can copy itself and provide instance system as rapidly as possible. In addition, for some read-only system libraries, all virtual machine instances and Zygote are sharing a memory region and saving the memory overhead.

Zygote process is responsible for follow-up other processes to create and start in Android application framework layer. Firstly, a SystemServer process is created (forked) by Zygote process. Secondly, SystemServer process is responsible for booting the system critical services, such as packet managed services PackageManagerService and application components management services ActivityManagerService. When we need to start an Android application, ActivityManagerService notifies Zygote process of creating a new process for the application through interprocess communication mechanism of Socket. Zygote process hatched the first Dalvik called SystemServer, and SystemServer is just the process alias. And the concrete process corresponding program is still an app_process, because SystemServer is hatched from app_process. These functions are completed in Zygote startup.

In the Dalvik virtual machine, Zygote supplies zygote interface to access to the Dalvik virtual machine. The interface wraps the fork function Linux system, it is used to create a new virtual machine instance processes. And ZygoteConnection is a socket connection to management and analytical parameters. When other Actvitiy establish process request, the socket will send command parameters to Zygote. The ZygoteInit is the main function of the Zygote entrance. As shown in Figure 3 the zygote process model. The Dalvik provides dalvik.system.Zygote class, the class provides three static methods for creating processes:

(1) fork(),create a zygote process,

(2) forkAndSpecialize(), create a non-zygote process,

(3) forkSystemServer(), create a system service process.

The fork () method will re-generate a new zygote process. The new process replicates the father process resources, including memory contents task_struct content. During the copying process, the child process copies the parent process task_struct, system stack space and page table. And when the child process changes father process variables, the program will create a new copy of the involved pages by copy on write, and the new zygote processes can also generate a child process. The Dalvik virtual machine threads are corresponding to the operating system threads, and the Dalvik processes are corresponding to operation processes.

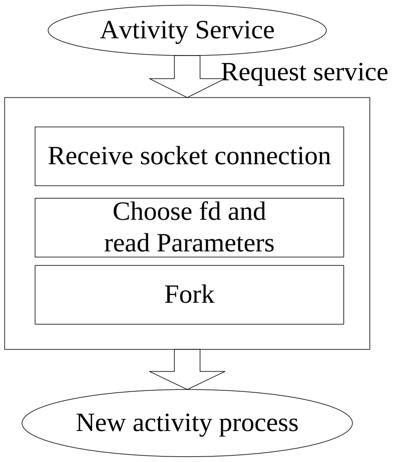

Figure 3. The zygote process model

The forkAndSpecialize () differs from fork (). The forkAndSpecialize () produces child which is not a zygote process. This is, forkAndSpecialize function’s child process can’t produce a new process.

forkSystemServer()'s child process is not also zygote process. When forkSystemServer()'s child process is end, its father is also end. The following steps are process code schedule of Zygote promoter.

(1) Through Zygote.forkSystemServer function the Zygote process creates a new process in order to start SystemServer components.

(2) The startVM function is called AndroidRuntime function to start Dalvik virtual machine, and AndroidRuntime function calls the startReg to register JNI method.

(3) Calling main function of com.android.internal.os.ZygoteInit class creates a socket interface and starts SystemServer component. Running ZygoteInit.runSelectLoopMode function into an infinite loop and earlier created socket interface will wait ActivityManagerService request to create a new application process. Now the program waits for ActivityManagerService to connect the Socket, then calls ZygoteConnection.runOnce function to create a new application.

3.2. Dalvik startup

The Zygote process will create a Dalvik virtual machine instances during boot time in the Android system. Whenever Zygote hatched a new application process, and the Dalvik virtual machine instances are copied to the new application process. That is, each application process has a separate the Dalvik virtual machine instance. In Zygote starting, function AndroidRuntime.start will start Android system runtime library [9]. The process completes four things. Shown in Figure 4, we can see the Dalvik virtual machine startup's process code schedule analysis.

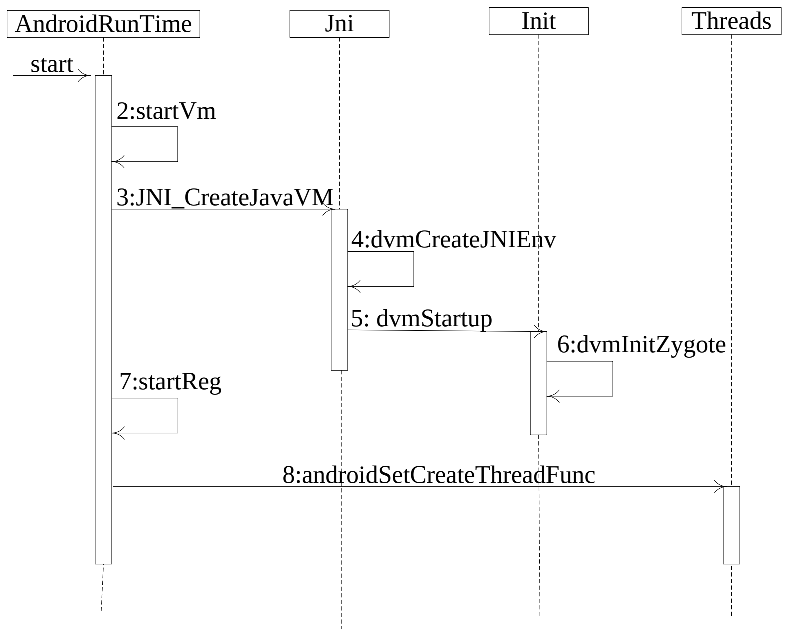

Figure 4. Analyze Dalvik startup process

(1)The startVM function is called by AndroidRunTime to starts the virtual machine.

(2)The startVm function calls JNI_CreateJavaVM function to create (create function: dvmCreateJNIEnv) and initialization (init function: dvmStartup) a Dalvik virtual machine instance.

(3) Calling the startReg function registers JNI methods, loading the Java core classes and JNI methods and they can make the main thread set up a JNI environment.

(4)Calling main function in com.android.internal.os.ZygoteInit class, finding Entrance that can make Zygote process enter java layer, and pre-loaded a large number of Android core classes and system resource files. The above is startup process of the Dalvik virtual machine in Zygote process.

From the above analysis, when Zygote process creates the Android application process, it can finish the follow process. It can will its own Dalvik virtual machine instance copy to the newly created Android application process, it can also share the core Java classes, Android core classes(dex file) and their JNI method(so file) with a new creating Android application process. Dex files and so files read-only code segment. They will always be shared by Zygote process, System processes and Android application process. Thus, a large number of pre-loaded behaviors conducted obtained value in Zygote process. So the way can speed up the boot process in System process and Android application process, and system can save memory consumption.

4. Creation process of the Dalvik virtual machine process analyzes and Instance

4.1. Analyze creation process of the Dalvik virtual machine process

The Android application process is created through ActivityManagerService services calling a static member function start of android.os.Process class to request Zygote process. Eventually Zygote process will create the Android application process through a static member function forkAndSpecialize of dalvik.system.Zygote class. So we start to analyze the creation process of Dalvik virtual machine process through a static member function forkAndSpecialize [10]. Shown in Figure 5:

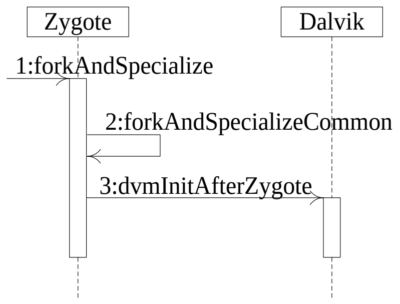

Figure 5. The creation process of Dalvik virtual machine process

(1)The static member function forkAndSpecialize of Zygote class is a JNI method [11]. In C++ layers the forkAndSpecialize function is implemented by Dalvik_dalvik_system_Zygote_forkAndSpecialize function. And the

Dalvik_dalvik_system_Zygote_forkAndSpecialize function can create a Dalvik virtual machine process by calling forkAndSpecializeCommon function.

Static member function forkAndSpecialize of Zygote class is a JNI method, which is achieved by Dalvik_dalvik_system_Zygote_forkAndSpecialize function in C++ layer. Its code is shown below:

```
/* native public static int forkAndSpecialize(int uid, int gid,
 *     int[] gids, int debugFlags);
 */
static void Dalvik_dalvik_system_Zygote_forkAndSpecialize(const u4* args, JValue*
 pResult)
{
     pid_t pid;
     pid = forkAndSpecializeCommon(args, false);
     RETURN_INT(pid);
}
```

In above code, the args parameter points to a u4 array, which it contains all parameters from the Java layer and it is encapsulated by the Dalvik virtual machine. Another parameter pResult is used to save the return result from JNI method calling, which is achieved through macro RETURN_INT.

(2)The function forkAndSpecializeCommon can be used to create a common Android application process and be used to create the system process. In Android system, only Zygote process has permission to create the system process and Android application process. When Zygote process is up and running Dalvik virtual machine, the gDvm.zygote value will be equal to true. Then the function forkAndSpecializeCommon can use the system call fork to create a new process.

The gDvm.zygote value will be set to false in the newly process, so that it can know that it is not Zygote process. When a process uses system to call the fork to create a new process, the former is called a parent process, the latter is called a child process. At this time the child and parent processes share the same address space. But when an address space is written by the parent process or child process, written address space will become independent between parent process and child process. This mechanism is called COW (copy on write) [12]. So, when the gDvm.zygote value of newly created process is set to false by the functionforkAndSpecializeCommon, the gDvm.zygote values of Zygote process is kept true.

When the Dalvik virtual machine starts, it will share the core Java classes, Android core classes(dex file) and their JNI method(so file) with the System process and Android application process. But due to the above COW mechanism, when necessary, System process and Android application process will still make a copy of these resources, so that t they have a separate instance of the Dalvik virtual machine.

(3)Finally, the forkAndSpecializeCommon will call the dvmInitAfterZygote function and initialize further the newly created process running on Dalvik virtual machine.

From the above analysis, the Dalvik virtual machine process is created to complete. And we draw a conclusion: a Dalvik virtual machine process is actually a Linux process.

4.2. Instance of creating a process

Through the above, we learn that the startup process for the Android application process. Shown in Figure 6, we can know that it is each step of process startup.

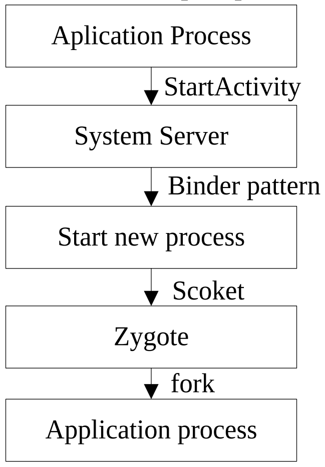

Figure 6. Step of process startup

Now we will create and start an activity which is named com.jsj.example.HelloActivity, and com.jsj.example is package name. We can have three parts analysis.

(1)In the Android system, Launcher is also a Activity, the application is activated by Launcher. When you click the screen icon, Launcher will start correspond to the application.

a) Launcher inherits from the Activity class, and Activity class implements the startActivity function, so the program calls the Activity.startActivity function. Through the Binder driver program enters the startActivity function of ActivityManagerService.

b) Through calling the topRunningActivityLocked function to obtain Activity in the top of the stack, this is HelloActivity. This code is shown below:

```
resumeTopActivityLocked(ActivityRecord prev) {
        ......
        // Find the first activity that is not finishing.
        ActivityRecord next = topRunningActivityLocked(null);
        ......
}
```

c) Firstly, the ActivityThread function will use Binder to reference token into remote interface ActivityClientRecord of ActivityRecord and notify ActivityManagerService that the Activity has entered Paused state. ActivityManagerService can now start HelloActivity.

d) In configuration file AndroidManifest.xml of Activity application, the system default using package’s name as process attributes, such as "com.jsj.example". Each application has its own uid, therefore, the combination of uid and process can create a ProcessRecord for each application.

e) Create a ProcessRecord variable, and stored value in the member variable mProcessNames. Finally, call startProcessLocked method of ActivityManagerService class for the further operation. This code is shown below:

```
ProcessRecord app = getProcessRecordLocked(processName, info.uid);
......
if (app == null) {
        app = new ProcessRecordLocked(null, info, processName);
        mProcessNames.put(processName, info.uid, app);
 }
```

(2)The ActivityManagerService operation

a) The ActivityManagerService starts a new process for the application by Process.start function and it will pass the first parameter’s value "android.app.ActivityThread". This is the process initialization to load Java classes.

b) When the Zygote starts, Dalvik virtual machine will start at the same time. Through openZygoteSocketIfNeeded function Process.start class connects to Zygote, starts Zygote and creates a Socket connection.

c) Through the Zygote.forkSystemServer function the Zygote process creates a new process to start the SystemServer component. And the Zygote calls the runSelectLoopMode function into an infinite loop, and earlier created socket interface will wait ActivityManagerService request to create a new application process.

d) Because the Zygote has returned to the connection of socket, so the program will obtain a ZygoteConnection object and connect the socket. Then the program can call the ZygoteConnection.runOnce function, and execute forkAndSpecialize. Now start creating process. This code is shown below:

```
class ZygoteConnection {
    boolean runOnce() throws ZygoteInit.MethodAndArgsCaller {
         ......
    pid = Zygote.forkAndSpecialize(parsedArgs.uid, parsedArgs.gid,
                    parsedArgs.gids, parsedArgs.debugFlags, rlimits);
         ......
     }
 }
```

e) Initializing runtime library for the newly created process. The parameter className is assigned to "android.app.ActivityThread" by ClassLoader.loadClass and load value of className into a process.

f) Execute main function of android.app.ActivityThread class. Firstly, create the ActivityThread object in the process, and enter the message loop. Then you can start Activity or the Service in the process.

(3) Run Activity in onCreate method

a) Retrieve previous created ProcessRecord and assigned to the app. The ActivityManagerService calls mMainStack.realStartActivityLocked to get need to bootable Activity, and the return value is HelloActivity.

b) The ActivityThread calls performLaunchActivity function to load the Activity class which is com.jsj.example.HelloActivity and call HelloActivity’s onCreate function.

According to the Activity life cycle, complete HelloActivity class loading and object creation and so on. The HelloActivity startup is complete.

5. Conclusion

With respect to the terms of the Linux kernel, the Android system is merely a Linux application. In Android system, each android application has a separate instance of the Dalvik virtual machine. The Dalvik virtual machine can regard the local operating system process as own process and it can use the process scheduling mechanism of the Linux kernel to dispatch its processes and threads. So it can realize high efficiency and fairness. In this paper, we analyze process code schedule of Android Dalvik Virtual Machine, Dalvik virtual machine startup process and Zygote startup process. We can know that the Dalvik work and run process. It is beneficial to study and improved the model of processes and threads of the Dalvik virtual machine.

6. ACKNOWLEDGEMENTS

The author would like to thank the anonymous referees for their valuable suggestions. This work was supported by Graduate Innovation Fund of Heilongjiang Province (YJSCX2012-232HSD).

References

[1] Y. Pengxiang, "Research on the structure and performance of Dalvik virtual machine", JinLin university**, (2011).**
[2] D. Ehringer, "The Dalvik Virtual Machine Architecture [EO/OL]" http://www.kiddai.com/NCTU/ebl/dex/The_Dalvik_Virtual_Machine.pdf. **(2010).**
[3] Z. Yimin, C. Rong, "Analysis about Process in Dalvik Virtual Machine", computer technology and development, vol. 20, no. 2, **(2010).**
[4] J. E. Smith and R. Nair, "Virtual Machines: Versatile Platforms for Systems and Processes", China Machine Press, **(2009).**
[5] L. Shengyang, "Android system source code Scenario Analysis", Publishing House of Electronics Industry, **(2012)**, pp. 611-641.
[6] N. Matthew and R. Stones, "Beginning Linux Programming 2nd Edition", China Machine Press, **(2002).**
[7] Y. Fengsheng, "Android Internals: System", China Machine Press**, (2011).**
[8] K. Yuandan, "Android kernel analysis", Publishing House of Electronics Industry, **(2011).**
[9] L. Shengyang, "The boot process analysis of Dalvik virtual machine [EB/OL]", http://blog.csdn.net/luoshengyang/article/details/8923484, **(2013).**
[10] L. Shengyang, "Analysis about creation process of processes and threads of Dalvik virtual machine [EB/OL]", http://blog.csdn.net/luoshengyang/article/details/8885792, **(2013).**
[11] B. Cheng and Bill Buzbee, " A JIT Compiler for Android's Dalvik VM [EB/OL]", http://www.kandroid.org/board/data/board/AndroidBeginner/file_in_body/1/android-jit-compiler-androids-dalvik-vm.pdf, **(2010).**
[12] Wikipedia. Copy-on-write [EB/OL], http://en.wikipedia.org/wiki/Copy-on-write, **(2013)**.

Authors

Wen Hu, master instructor, president of School of the Computer and Information Engineering, Harbin University of Commerce, backup leader of “Electronic Commerce” provincial key discipline echelon and academic leader of secondary doctoral discipline “electronic commerce and information service” in first-class doctoral discipline “business administration”. His main research fields include computer network and communication, electronic commerce, embedded technology.

Yan li Zhao, Master, student of School of the Computer and Information Engineering, Harbin University of Commerce. Her main research fields include embedded technology.