Advances in Intelligent Systems Research (AISR), volume 142
2017 International Conference on Information Technology and Intelligent Manufacturing (ITIM 2017)

# Third-part software calls and related process management based on Eclipse

MingCheng Qu[1,2],XiangHu Wu[1,2], YongChao Tao[1,2],QiGang Hu[1,2]
[1]School of Computer Science and Technology, Harbin Institute of Technology; Harbin 150001 Heilongjiang China
[2]Shenzhen Academy of Aerospace Technology, Shenzhen 518057 Guangdong China
[*]E-mail:qumingcheng@hit.edu.cn

***Abstract*—As embedded hardware and software become increasingly complex, the traditional command based development debugging operations have been unable to meet the daily needs. Graphical debugging tools have become the most popular choice. As a good open source software, Eclipse provides a large number of expansion nodes for us to develop our own functions. The embedded integrated development environment needs to be integrated with other related software on the basis of Eclipse. This article is to study how Eclipse invokes other software and manages related processes. This article uses the WorkspaceJob class and the Process class provided by Eclipse to complete the related functions.**

***Keywords—Embedded debugging, Eclipse, WorkspaceJob, Process manage***

## I. INTRODUCTION

As a good open source software, many embedded integrated development environments are implemented based on Eclipse. On the basis of Eclipse, the embedded integrated development environment is developed. The core is to call other related software and manage the related acquired process. This article will tell you how to implement the above functions through the function flow chart and the related kernel code.Eclipse based calls to other tools for more research, such as abroad on Github has an open source project: GNU, ARM, Eclipse. The project includes a series of Eclipse plug-ins and related tools developed specifically for multi platform embedded ARM. The entire project is based on the GNU tool chain. Hosted on Github and SourceForge. It provides integration of the three debugging tools, OpenOCD, JLink, and QEMU. However, it does not provide a unified method for invoking other tools. And it does not provide the process management mechanism, after calling the relevant software did not do any processing, often there will be a large number of invalid processes in the background, and waste of CPU resources.

This article details how to call other tools in Eclipse, and provides a base class to implement related functions. And on the basis of the multithreading mechanism provided by Eclipse, the management of the process is realized. After the process fails, the invalid process is terminated in a timely manner.

## II. DETAILED DESIGN AND IMPLEMENTATION OF OTHER TOOL CALL FUNCTIONS

Other tools are invoked in Eclipse, mostly through the Java class of Process. The Process class will have two output streams at execution time. One is the standard output stream, and the output is the correct execution result. One is the error output stream, and the output is the wrong execution result.

If the two output stream processing, only the standard output stream, if the error or exception occurred during the execution, likely to cause obstruction of output current, and the program will not continue, locked state.

And if the two output stream into one output stream, although can solve the blocking problem of output current, but this error output and correct output merge together, not a clear distinction between error and correct output output.

So here I did the processing, and extended the design and implemented a RealtimeProcess class on the basis of the Process class. Fig.1 is a design class diagram for the RealtimeProcess class.

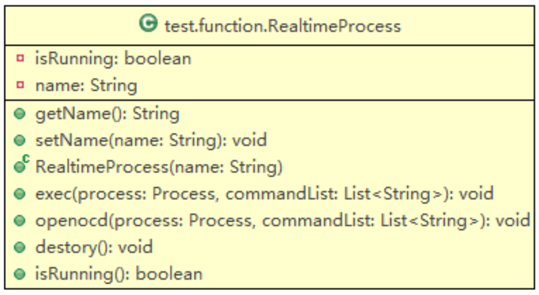
test.function.RealtimeProcess
isRunning: boolean
name: String
getName(): String
setName(name: String): void
RealtimeProcess(name: String)
exec(process: Process, commandList: List<String>): void
openocd(process: Process, commandList: List<String>): void
destory(): void
isRunning(): boolean

Fig. 1. RealtimeProcess design class diagram

The class contains two variables: the isRunning flag, whether the process is running, and the name representing the process's name. There are three kinds of the main methods: (RealtimeProces) is a construction method of the class, is the name of the process parameters; (exec) is a function of the execution of other programs and process parameters for the instruction to be executed list; openocd () and exec () function, its function is similar, but openocd.exe is special, correct execution it will output the error output stream, so the design and implementation of a single function.

The general function flow chart that the RealtimeProcess class calls other programs and gets the execution results, as shown in fig.2.

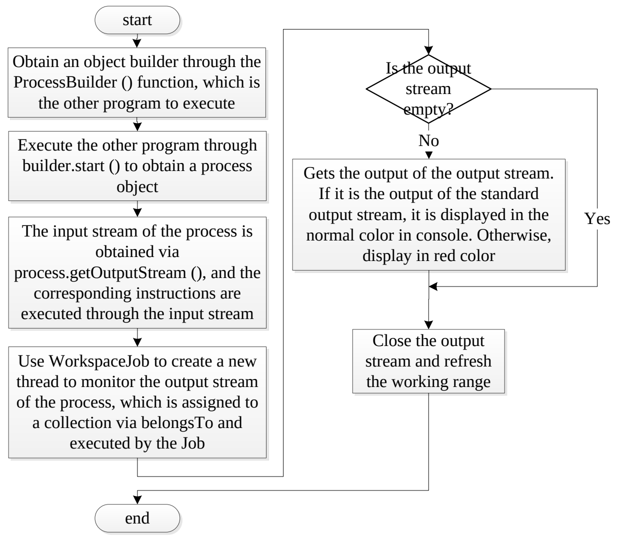
start
Obtain an object builder through the ProcessBuilder () function, which is the other program to execute
Execute the other program through builder.start () to obtain a process object
The input stream of the process is obtained via process.getOutputStream (), and the corresponding instructions are executed through the input stream
Use WorkspaceJob to create a new thread to monitor the output stream of the process, which is assigned to a collection via belongsTo and executed by the Job
Is the output stream empty?
No
Yes
Gets the output of the output stream. If it is the output of the standard output stream, it is displayed in the normal color in console. Otherwise, display in red color
Close the output stream and refresh the working range
end

Fig. 2. A function flow chart that calls other programs and gets output results

The concrete steps are as follows:

(1) create a builder object through the ProcessBuilder () function, which is the absolute path of other executable programs that you want to execute.

(2) by executing the other program through builder.start (), a Process process object process is obtained, and process is the process after opening the other program.

(3) obtain the input stream of the process via process.getOutputStream (), and execute the corresponding instructions through the input stream. The specific code is as follows:

New BufferedWriter (New OutputStreamWriter (process.getOutputStream ())); initializes a BufferedWriter object br through this line of code.

Then, by executing the function br.write (CMD), the parameter is the program instruction to be executed, and the instruction can be sent to other executable programs through the input.

(4) to create a new thread using WorkspaceJob output to monitor the process process flow, through the belongsTo (the Job) belonging to a set, convenient for the unified management of the process, through the schedule () to add the thread to the waiting thread, waiting for the execution of the Job operating system.

The multi threading option here is the Job mechanism of the Eclipse itself.

In contrast to the two implementations of multithreading, Runnable and Thread, the Eclipse mechanism provided by Job is more easily controlled by programs and more compatible with Eclipse. At the same time, it can display the progress bar in the right lower UI interface of Eclipse real time, and it is more practical and friendly.

Inheriting the Job base class, the Job mechanism of Eclipse contains three extensions:

1.UIJob

UIJob is the UI thread of Eclipse. In the course of its execution, UI is not refreshed, so it is not suitable to perform long time-consuming operations in this thread, otherwise it will cause the interface card to die easily.

2.WorkbenchJob

WorkbenchJob is also a UI thread. Is an extension of UIJob. Unlike UIJob, it can run jobs only when Eclipse is executing.

3.WorkspaceJob

WorkspaceJob is a non UI thread. You can use it when you modify a resource file. Except for UI, the operation can also use it.

Comparing the three Job, it takes a lot of time during the execution of this function, so it cannot be executed on the UI thread. So WorkspaceJob was chosen.

(5) when the monitor thread starts executing, it will listen to the output stream of other executable programs. A while loop is used here, and the termination condition of the loop is that both the standard output stream and the error output stream are null.

When the standard output stream has a result, it is output to the custom console in white; when the error output stream has a result, it is output to the custom console in red color. This allows a clear distinction between correct information and false information.

(6) when the standard output stream and the error output stream are all empty, turn off the two output streams and refresh the work area at which the project is located.

The reason for refreshing the working area of the project is that sometimes, when invoking other executable programs, a file result is generated, for example, when the FPGA project is compiled, there will be a bit file. If you do not perform a refresh operation, the bit file will not be displayed directly in the Eclipse project directory, users need to manually refresh, so in order to user friendliness, here on each call to end other executable program will automatically refresh the project workspace.

## III. Detailed Design and Implementation of Process Management Module

Due to the integrated development environment often need to call other executable programs, from the windows operating system point of view, these are called other executable programs and Eclipse.exe belong to different processes in the background in a non graphical way of running. If you do not provide a unified termination management for these processes, problems can occur, such as Eclipse shutdown or abort, while other executable programs that are called will still run in the background. For example, when calling Vavido software, if you do not kill the previous process, there may be repeated establishment of the process, the use of a large number of CPU resources.

Therefore, the integrated development environment needs to provide a unified process management mechanism. Management here refers mainly to the termination of the void process. Here, the operation of the kill process encapsulates a function. This is primarily called when the Eclipse is closed, and other calls are needed to terminate the invalid process. The following details describe how to automatically call the process termination function when the Eclipse is closed.

In the plug-in development project you built, you inherited AbstractUIPlugin to implement a Activator class that manages the life cycle of the plug-in.

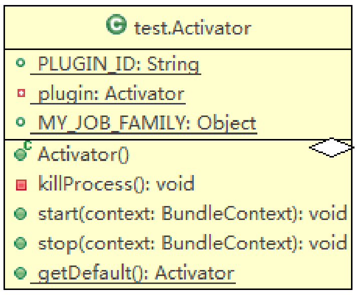

Fig. 3. Class Activator design class diagram.

The variables included: PLUGIN_ID ID plug-in, plug-in can uniquely identify through this ID; MY_JOB_FAMILY is used for all the Job management plugin created, all created by the plugin Job belong to this collection.

The class contains methods: the killProcess () function is the newly created function, used for process termination operations. The start () function is a plug-in initialization function that triggers execution when the plug-in starts calling. The stop () function is the end function of the plug-in that automatically leaves the function at the end of the plug-in lifecycle.

In this class, you override the stop function, which will execute when the plug-in terminates, the Eclipse exception terminates, and the normal exit will trigger the function. In the stop function, the process termination function is invoked, and a unified termination operation is made to the process opened by the integrated development environment. Fig.4 is a functional flow chart that terminates the invalid process.

There are two types of processes that require termination, one called the process produced by other executable programs, and the other is the WorkspaceJob type process created by the Eclipse when the integration development environment is doing some operations. The two processes terminate in different ways.

The process of terminating an invalid process is as follows:

(1) obtain the IJobManager object through Platform.getJobManager (), which is the global job manager of Eclipse.

In the introduction above, each time you create WorkspaceJob, you assign it to a series, and the WorkspaceJob created by this integrated development environment is assigned to the same series. Therefore, IJobManager.cancel (MY_JOB_FAMILY) is used to cancel all job under this series, so that all WorkspaceJob created by this integrated development environment is cancelled.

(2) similar to the other executable programs described above, use the Java function to call the windows of cmd.exe, first execute the Tasklist command, and get all the running process information. This information is saved as a list.

(3) through the information list, if the need to traverse to the line containing the termination process, the split function using java cutting out the PID of the process, the implementation of the "windows cmd.exe taskkill /F by calling the /PID PID command to terminate the process.

(4) terminate the cmd.exe process after the completion of the traversal.

By doing so, you can terminate all invalid processes when needed.

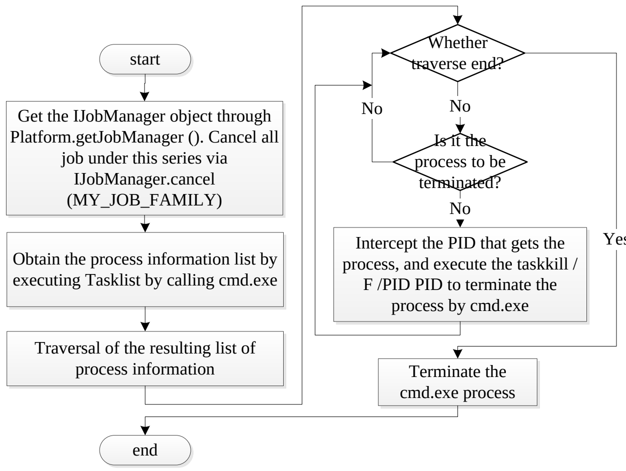
start
Get the IJobManager object through Platform.getJobManager (). Cancel all job under this series via IJobManager.cancel (MY_JOB_FAMILY)
Obtain the process information list by executing Tasklist by calling cmd.exe
Traversal of the resulting list of process information
Whether traverse end?
No
No
Is it the process to be terminated?
No
Intercept the PID that gets the process, and execute the taskkill / F /PID PID to terminate the process by cmd.exe
Yes
Terminate the cmd.exe process
end

Fig. 4. A function flow chart that terminates the invalid process

## Acknowledgment

This work is funded by National natural Science Foundation of China (No.61402131); China Postdoctoral Science Foundation Special Fund (2016T90293); China Postdoctoral Science Foundation Item (No.2014M551245); Heilongjiang Provincial Postdoctoral Science Foundation Item (No.LBH-Z13105); Central University Basic Scientific Research Business Expence Special Fund (No.HIT.NSRIF.201651); Harbin Science and Technology Innovator Special Fund (2015RAQXJ047).

## References

[1] Peng Hao Yang,Rong Liang Wang,Zi Guo Fan. Study on Debugging Method Based on Embedded System Development Platform[J]. Advanced Materials Research,2013,2385(694):.

[2] Hui Qin He. Application Research of JTAG Standard Based on ARM Debugging System[J]. Applied Mechanics and Materials,2015,3749(719):.

[3] Neal Stollon. Multicore Debug[M].Elsevier Inc.:2013.

[4] Catalin Dan Udma. Software Development Tools for Embedded Systems[M].Elsevier Inc.:2013

[5] Wang S, Kang M N. Implementation of Embedded Remote Debugging Software Based on Eclipse Platform[J]. Microprocessors, 2014.