Windows Service for Remote Job Server of Möbius™

Kaishen Wang

Introduction

Möbius™ is a software tool for modeling the behavior of complex systems. Although it was originally developed for studying the reliability, availability, and performance of computer and network systems, its use has expanded rapidly. It is now used for a broad range of discrete-event systems, from biochemical reactions within genes to the effects of malicious attackers on secure computer systems, in addition to the original applications.[1]. A remote job server [2] has been implemented to enable parallel job exectuion on different machines. While Unix-like operating systems like Linux and OS X support remote login through SSH, the Windows operating system doesn't. As a result, a Windows service is needed to start the remote job server on a Windows operation system.

This paper gives a detailed explanation of the necessity and the function of this service and the step by step implementation of it. The *Windows Service* section explains the function and importance of the Windows service [3]. The *Möbius Remote Job Server* section explains the framework of the RJS and how it works. The *First Steps* section describes basic preparation made to build the service. *Building the Möbius RJS Service* section descibes how the RJS service should behave and how is it implemented.The *Future Work* section mentions the possible work that can be done in the future. The *Conclusion* section summarizes the current progress that has been made. The *Appendix* explains the role and implementation of several important files.

Windows Service

A Windows service is a program that acts like a normal Windows process. However, it is started by the operating system and runs in a separate session where the service will not be interfered by user or other processes. Similar to a daemon in Linux and Mac OS X, a Windows service does not need human interaction while running.

A service application conforms to the interface rules of the Service Control Manager (SCM). It can be started automatically at system boot, by a user through the Services control panel applet, or by an application that uses the service functions. Services can execute even when no user is logged on to the system [4].

To manually install a service to or delete one from the operating system, command line statements such as “sc create” or “sc delete” are needed. These commands can be saved in bash scripts for convenience.

Möbius Remote Job Server

The Remote Job Server provides users of Möbius with the ability to schedule and manage processes on remote machines. It also allows users to indicate the appropriate priority of a job to the scheduler and to halt remote jobs and remove them from the scheduler, enabling solver execution to be cleanly halted and resumed later without losing existing progress [2].

The Remote Job Server operates in one of the three modes: manager, worker, and forward. The manager node will schedule and distribute jobs to worker nodes and forward nodes. It will also monitor the progress of worker and forward nodes. The forward node will forward the job it receives to its corresponding worker nodes. And the worker node will run the real simulation and calculation.

In Linux and Mac, the Remote Job Server can be started using a secure shell connection. However, this is not the case in Windows as Windows NT systems does not support SSH. However, it is tedious to start the Remote Job Server on remote machines every time manually. A Windows service will perfectly handle this. The remote machines with this service installed will automatically start the Remote Job Server when they start.

First Steps

During the preparation stage of creating the service, a sample C++ service which was derived from a code sample website [5] was built. To keep things simple while learning how to build Windows services, the sample service starts a small example server using MobiusSocket, which reads 8 characters from a connected client, sends the same characters back to the client in reverse order, and closes the connection when done.

Just like other Windows services, the sample service has the following 3 important functions:

- `ServiceMain`: The main body to initialize the service and start it.
- `ServiceCtrlHandler`: The function that handles the stop command.
- `ServiceWorkThread`: The function executes the small example server, which listens and handles connections from multiple client, performing the 8 character reversal for each one.

MobiusSocket class are used to abstract the Windows socket library calls.To test the functionality, a small sample client using MobiusSocket was created to interact with the sample server service.

Several bash scripts were created in order to add, start, stop, and delete the service. These bash scripts were modified later to cope with the real service.

Building the Möbius RJS Service

The *Möbius* RJS Service should be started when the machine starts. Just like other services, the RJS service can be stopped or restarted through the Service Control Manager. Although the RJS supports three modes, manager, worker, and forward, the RJS started by the service can only be manager mode. It is possible for a manager node on a machine to launch a worker node or a forward node although it is quite inefficient and resource consuming to do so. To implement forward mode and worker node, additional functions should be added to `JobServer` library to support. However, due to the limit of time and effort available at this stage, work on the other two modes have been left for the future.

Compared to the sample service, the real RJS Service also runs a MobiusSocket. Rather than doing the reverse string job, the server will do the real job of manager node or forward node.

To implement the RJS Service, several changes are made to the sample one. First, the library “libJobServer” was linked to the project, together with several system libraries like `openSSL` and `Xerces`. The service will read an ini configuration file that contains several properties. The RJS server uses the `ServiceIsValid()` function to check the properties it has, and then use `InitConnection()` function and `Live()` function to initialize the server and create new Reader and Writer threads to work.

To shut down the service elegantly, all the corrosponding Forward and Worker nodes that belong to the machine should be closed before the service ends. Otherwise, the work finished earlier maybe lost. As the RJS server will run a blocking receive function, there is no other method to notify the server to shut down except sending a packet to it.

In order to complete this, when the service needs to be shut down, it will build a new RJS client and connect to the local server. Then a packet that contains the `DieCommand` will be sent from localhost to notify the server to halt. In the end, the server will handle all the nodes belong to it and shut down itself.

The service will also log necessary information in each step to a log file `RjsService.log` in `C://Windows/SysWOW64`.

Future Work

Two future improvements need to be made to make this work complete. It is difficult for users to install the service by writting configuration file and command lines on their own. So an installer application is necessary to automate the install for potential users. Another task is to create a small UI application to configure the xml ini file.

Conclusion

In this project, I gained a lot of experience in coding and debugging. As I have never dealt with a project that contains more than 20 files, so handling this project, which contains more than 400 files, was a big challenge for me.

Additionaly, I didn't have any coding experience in the Windows operating system. Using Visual Studio in Windows and getting familar with Windows system functions was also a valuable experience.

Appendix: Detailed File Explanation

All the files are stored at kwang40@blue.perform.illinois.edu/srv/git/grp/mobius/RjsService.git. There are four major parts included in this project. The first one is `JobServer`, which is a real JobServer. It is only stored for reference and backup. The second one is `JobServerService`, which is in charge of starting and stopping the service. The third one is `libJobServer`, which encapsulates the implementation of the job server and provides the interface of its member functions to the other three parts. The fourth one is `RjsClient`, which can set up a client in order to shutdown the service.

`Properties.hpp/ Properties.cpp (libJobServer)`

These two files are in project libJobServer. The Properties class provides an interface for defining various service parameters, like those decribed in the "RJS_config.ini" section below.

`ServiceMain.cpp (JobServerService)`

This is the core part of the service. It reads the information in the configuration file and uses the Properties class to define the operating parameters. Besides creating and starting the server, this file also handles the stop request. These details have been explained in the Building the Möbius RJS Service section.

`InitializeClientNode.hpp/InitializeClientNode.cpp (RjsClient)`

These files are used to initialize the client node that the service builds to shut down the server. It will also read the configuration file to learn about the properties that the server has, and thus to set its own.

`TestNetwork.hpp/TestNetwork.cpp (RjsClient)`

These files are used to shut down the server. It contains functions that make the client node connect to the local RJS and send a `DIE` command packet to the RJS. After receving the `DIE` command, the RJS will wait for all the forward and worker nodes to close and then shut down itself. After that, the service can be stopped.

`RJS_config.ini`

This is a configuration file that contains the value we need to initialize the service. This file includes four sections: “portSection”, “PersistModeSection”, “AdminPasswordSection” and “NodeTypeSection”. Given the section names, the information contained in each section is intuitive. “portSection” contains an integer represents the port number the service is listening on. “PersistModeSection” contains the information about whether the node will sit and wait until a network with the administrative password connects to it and it should join that network. The value can be “Non-persistent mode”, “Persistent mode” or “Manual Persistent mode”. “AdminPasswordSection” contains the password for the node, which maybe used to send commands to other nodes in the topology. And “NodeTypeSection” contains the node type of the machine, which maybe manager, worker, or forward.

The parameters available for use in the RJS_config.ini file are:

- portSection - This section defines information about the connection port.
    - Parameter_1 - An integer represents the port number the service is listening on.
- PersistModeSection - This section contains the information about the mode of the server.
    - Parameter_1 - A string that can only be "Non-persistent mode", "Persistent mode” or "Manual Persistent mode”. As for “Non-persistent mode”, this node should immediately attempt to connect to its parent node. This node will shutdown if it loses a path to the manager node. If this is the manager node, it shuts down when there is no path to some client. As for “Persistent mode”, This node should immediately attempt to connect to it's parent node. If it becomes disconnected from the network it can join network if a network connects to it with the administrative password. This node will shutdown only when told to by a manager node. If this is a manager, it only shuts down when a client with administrative privileges tells it to. As for "Manual Persistent mode”, this node should sit and wait until a network with the administrative password connects to it and then it should join that network. This node will shutdown only when told to by a manager node. If this is a manager, it only shuts down when a client with administrative privileges tells it to.

- AdminPasswordSection- This section contains the password for the node, which maybe used to sthe server.
    - Parameter_1 - A string representing the password.
- NodeTypeSection- This section contains the node type of the machine
    - Parameter_1 - A string that can only be “manager”, “worker”, or “forward”.

addService.bat/startService.bat/stopService.bat/delService.bat

The functionalities of these bash scripts are quite apparent according to their names. In the future, addService.bat is no longer needed because of the existence of MSI installer package. Starting and stopping service can also be completed through Service Control Manager.

References

[1]
"The Möbius Tool :: Home." The Möbius Tool :: Home. Accessed October 28, 2016. https://www.mobius.illinois.edu/.
[2]
K. Keefe, Q. Mitchell, E. Rozier, and W. H. Sanders. (09GUL01) Proceedings of the 6th International Conference on Quantitative Evaluation of SysTems (QEST) 2009, Budapest, Hungary, September 13-16, 2009, pp. 209-210. [IEEE Xplore entry]
[3]
"Introduction to Windows Service Applications." Introduction to Windows Service Applications. Accessed October 28, 2016.
https://msdn.microsoft.com/en-us/library/d56de412(v=vs.110).aspx.
[4]
"Introduction to Windows Service Applications." Introduction to Windows Service Applications. Accessed October 28, 2016.
https://msdn.microsoft.com/en-us/library/d56de412(v=vs.110).aspx.
[5]
Arora, Mohit. "Simple Windows Service in C." - CodeProject. 2013. Accessed October 28, 2016.
http://www.codeproject.com/Articles/499465/Simple-Windows-Service-in-Cplusplus.