Research & Implementation of uCLinux-based Embedded Browser

WANG Minting, LIU Fagui
School of Computer Science and Engineering, South China University of Technology, Guangzhou 510640, China
wangminting@163.com, fgliu@scut.edu.cn

Abstract

Embedded browser is an important component to support Ubiquitous Computing in Information Appliances. How to develop an uCLinux-based embedded browser remains to be a big challenge and quite demanding. This paper uses Konqueror/embedded as a prototype. Based on the interactivity between the bottom network connection layers, it proposes a theory to turn the multi-process mechanism into multi-threads one and implements successfully on uCLinux, which provides us a methodology of how to transplant a concurrent system from normal Linux to uCLinux.

Keywords: Konqueror, uCLinux, transplant, multi-process, embedded browser

1. Introduction

With daily increasing cross-integration in consumer electronics, computer and communication industries, embedded browsers, the bridge to outside world, is becoming more and more important. Unlike web browser on PC, the embedded browser is system-specific which makes its development even harder. The current market of embedded browser is monopolized by a handful of large foreign companies, with many small embedded browsers still in the seedtime period. Meanwhile the domestic product has made their own way but still immature at some aspects such as not universal support to various protocols/standards and unable to achieve full web browsing function [1]. Therefore, in the respect of technology proprietary, it is better to develop a domestic mature, stable, full-function embedded browser. Due to involvement of massive technologies, it is risky and uncompetitive to start over the whole research from the very beginning [2]. This paper is based on the open source projects.

The recent market research shows that many consumer electronics products (such as smart phones) do not have to possess a very strong real-time response but a lot of function. That is why it will simplify the development process if they develop their embedded products on the open source operating system Linux/uCLinux [3]. The uCLinux is a variation of Linux, mainly operating on the CPU unequipped with MMU. It reserves a lot of Linux features, for example: most of the system schedule function, low energy consumption, and support for low-end 32-bit embedded machine [3]. The transplantablity, compatibility, popularity and open-source features of uCLinux makes it a perfect develop platform.

According to the MiniGUI White Paper survey, there is only a few uCLinux-based GUI. Among them MiniGUI and Qt/Embedded are outstanding (it can support uCLinux after amending the bottom driver). Only handful open source embedded browsers can run on MiniGUI in unstable and incompetent fashion, On the other hand, the famous Konqueror/Embedded developed from Qt/Embedded can fulfill the gap. Konqueror/Embedded is Linux-based with a natural advantage to transplant to uCLinux, for instance, it can correspond with GNU, support all the browser-related standards and protocols, possess excellent speed, display and user-friendly interface, and what's more, it is stable. That is why this paper chooses Konqueror/Embedded as a developing prototype.

2. The comparison between uCLinux and Linux

The key difference between uCLinux and Linux is uCLinux does not support MMU, which will has certain influences on system memory and multi-process management [4]. In the standard Linux, it creates processes by calling the system function fork(). A child process is an exact copy of the father process, with the support of MMU, father process and child process have their own address space, and each goes its own way from the function fork (). While in the uCLinux, without the support of MMU, all processes sharing a public address space, a child process created by function fork() has its indicators still pointing to the father's data, so uCLinux abolish the function fork() [4]. The standard Linux system can also call vfork() to create child processes. After calling vfork(), the father process will lend its own memory space to the child process, and enters the sleep mode until the child process calls exec() or exit() to quit, then the kernel will wake up the father process and return its memory space. Although in the vfork(), the father and the child process occupy the same memory space, there is no data replication from the address space, and will not have the problem of data conflict, so uCLinux can use vfork() to create new process [4]. As a result, application that rely on function fork() to improve the throughput and response time of the system must be amended for running on uCLinux [4].

We need to be aware that even with vfork(), the father process and the child process still share the same address space. The child process must call exec() or exit() to quit as soon as possible, and make sure that the data of the father process will not be rewritten before calling and that the father process's stack environment will not be destroyed. Otherwise, it can easily result to system collapse [4]. Therefore, in this paper we rewrite the Konqueror/Embedded which rely for fork() using the multi-threads structure.

3. The bottom network connection of Konqueror/Embedded

Konqueror/Embedded is made up of bottom network connection, graphical user interface and HTML rendering engine named KHTML [5-6]. The bottom network connection module is closest to the operating system, and more depend on it than others, when changing the operating system, we have to amend this module. Below we will analyze I/O-Slaves mechanism which is the foundation for the bottom communication protocol, focus on how the processes be created, exited and communicated.

3.1. I/O-Slaves mechanism

I/O-Slaves in Konqueror/Embedded use the method of KDE to access data, according to the different protocols, such as HTTP, FTP, different slaves will be created to analyze the protocols, each slave bound to a slave-process, which can access files or directories from the internet automatically.

Open a website is an asynchronous work, as most of the websites contain more than one objects (for example, pictures), if using a synchronous way, it may wait a long time to open the website, and the website may be interrupted when transmitting data. Konqueror/embedded divides a website into several jobs, each job responsible for an URL. Under the control of the main-process, jobs will work independently using the I/O-Slaves mechanism. The implement flow is shown in Figure 1.

Jobs are placed in the queue. When the cycle of event has to deal with a job, Schedule will create or distribute a slave for it, as shown in Figure 2. Slave is a designing concept, represent for the part independently accessing internet documents. Slave includes two one-to-one objects. One is running in the main application side, which inherited from the Class KIO::Slaveinterface, under the control of Schedule and bound with an appropriate job. The other is running in the Slave-end process side, which inherited from the Class KIO::Slavebase, responsible for network file accessing.

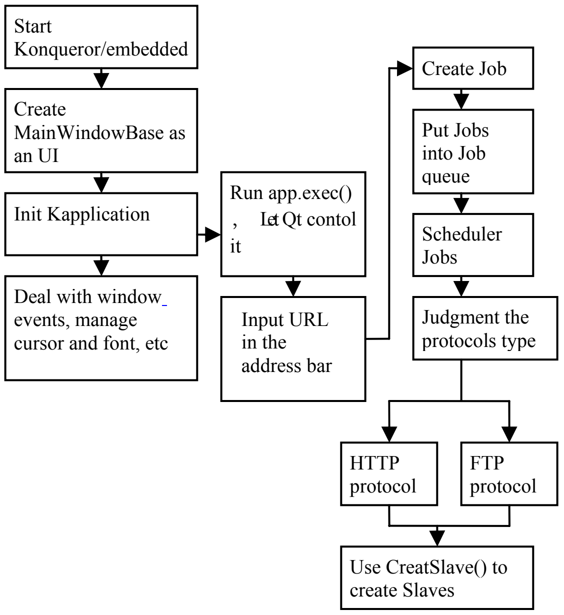
Start Konqueror/embedded
Create MainWindowBase as an UI
Init Kapplication
Deal with window events, manage cursor and font, etc
Run app.exec() , Let Qt contol it
Input URL in the address bar
Create Job
Put Jobs into Job queue
Scheduler Jobs
Judgment the protocols type
HTTP protocol
FTP protocol
Use CreatSlave() to create Slaves

Figure 1. I/O-Slave flows

The two Slaves exchange command and data through pipes. Slave work under the control of job. When job finishes, slave-process will not quit immediately, but wait until the whole task is finished. This will be convenient for the next job to bind to, and save costs from creating processes. For a certain network protocol there can be at most 3 slave-processes at the same time. If the jobs of a protocol are more than 3, the remainders will wait in the queue.

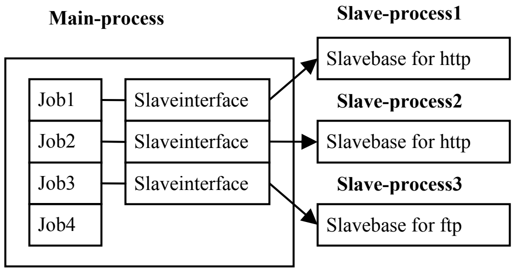

Figure 2. Jobs and Slaves in I/O-Slaves

3.2 The creation, exit and communication of processes

From the above analysis, we can see that slave-processes are created and terminated by the main-process. The main-process sends orders to and receives data from slave-processes; all the processes are independence and concurrent. At the bottom network connection, there are three kinds of communication methods between processes: pipe, socket, and signal; and how they have been used will be discussed below.

Although slave-processes are created by the main-process, the main-process does not create them itself. Instead, there is a middle-process with a Launcher object which will do the job to release the main-process's load. The main-process only has to send messages that are necessary for creating a slave to the middle-process through socket in order to continue. The main-process and slave-process can not communicate with each other until the middle-process has completed the creation and returns the slave-process ID. The details are shown in Figure 3 (only the major procedures are described here).

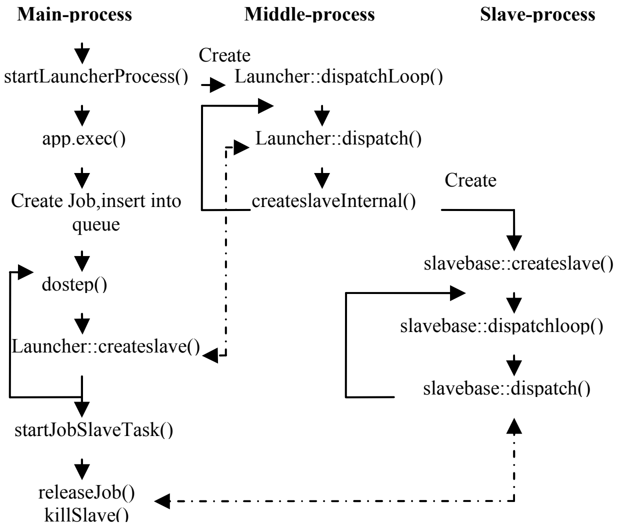

Figure 3. Processes communicate

In figure 3, the real lines represent the flow, the dotted lines represent communications between processes. The main-process uses fork() to create the middle-process at the very beginning of the process and configure the socket ports at both ends at the same time. Thus when the main-process receives an URL where translation is needed, it will divide the task into several jobs and enter the doStep() repeatedly until the task is finished. In the doStep(), scheduler will bind a slave to a job, if slave is absent, a new slave will be created. As the main-process and the slave-process are communicate with pipes, the main-process will send the file descriptors located at one end of the pipes needed by the slave to the middle-process, then middle-process will bind them together as a communication node. When the slave-process is created, the main-process will bind the other end of the pipes as a communication node too. Then the main-process will call startJobSlaveTask() to configure the job and start to communicate with the slave-process. The main-process will send command to slave-process. The slave-process will follow it and analyze the protocol, then return data.

The pipes used at both ends of the slave are shown in Figure 4:

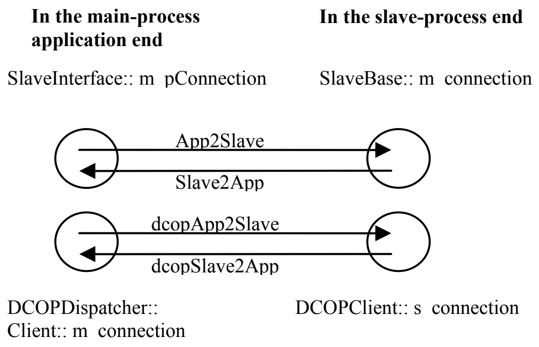
In the main-process application end
In the slave-process end
SlaveInterface:: m_pConnection
SlaveBase:: m_connection
App2Slave
Slave2App
dcopApp2Slave
dcopSlave2App
DCOPDispatcher:: Client:: m_connection
DCOPClient:: s_connection

Figure 4. The pipes and nodes used in the bottom network connection

In Figure 4, the arrows represent the pipes between main-process and a slave-process; there are a total of 4 pipes. The head of the arrow is the reading-end and the tail is the writing-end, and the circle represents the node which binds the reading-end and the writing-end together. Each end has two such nodes that make them have two communication channels. One is for the slaves of both ends to work synchronically, while the other acts as a desktop communication protocol (DCOP). We need to note that s_connection is a static variable, and does not need to create objects when using it.

After the main process reassures that it has finished the whole task, the signal SIGTEAM will be send to kill all slaves of the task.

4. The improvement of bottem network connection on uCLinux

In uCLinux system, all processes are sharing a public address space [4], This is similar with the thread mechanism that several threads share the same address space of a process. Thread also known as "lightweight process", it can do a job separately as a process. As thread can maximize sharing of resources, making the switch quickly, so it is very suitable for the concurrent system. Born with concurrent features, it is feasible to transform the Konqueror/Embedded into a multi-thread system.

After analyzing the bottom network connection, we work out a proposal: let Konqueror/embedded run on one single process, turn the processes of I/O-Slaves into threads, use multi-thread mode instead of multi-process running on uCLinux, and improving concurrency. This paper focuses on solving the threads communication and the data corresponding problem.

As slave-processes in Konqueror/Embedded are independent, clear-tasked, and no need to share public data from creation to exit, and after it exits, system will recycle its resources immediately. It coincides with separate-type thread and we adopt it.

Although the child process does not take possession of public resources, when turn into thread, static variable conflict must be paid attention to. Static variable is a global variable of a document stored in the global data section. In the case of multi-process, child process will have its own global data section, no overlapping data sharing section. But it is different in the multi-thread case, as a child thread and father thread share the same global data section; it is easier to have data conflict. Normally, we can use pthread_mutex_lock to solve the problem, but it will bring in another complicated problem. For example, if a thread locks a public data and exit abnormally, it will lead to a deadlock. Moreover, only a thread can use the data one time will reduce the concurrent effect, and lock-unlock the pthread_mutex_lock frequently will consume the system resources. Therefore, we do not take the pthread_mutex_lock, but create a public structure array in the global data section. The structure ties the thread ID and its data into array together when need data, the thread will find it in the array with its ID. When thread exits, it will call the function to delete its structure from the array. For example, in the case of s_connection mentioned above, we rewrite the code as follows:

This is the original code:

```
static KIO::Connection *s_connection;
static void setGlobalConnection( KIO::Connection *conn
) { s_connection = conn; }
static KIO::Connection *globalConnection() { return
s_connection; }
```

This is the rewritten code:

```
struct connection_tid
{   //the structure for tiring date and thread id together
    KIO::Connection *s_connection;   // the data
    pthread_t tid;   // thread id
};
static QList<connection_tid> * cplist;   //the structure
array in global data section
//below are the rewrite of the original function
Void     DCOPClient::setGlobalConnection(connection_
tid& cp)
{   // add new structure to the array
      cplist->append(&cp);
}
KIO::Connection *DCOPClient::globalConnection()
{   //find the data will thread id
      pthread_t tid = pthread_self();
      for(QListIterator<connection_tid>it(*cplist);it.curre
nt();++it)
      {
            if(tid == it.current()->tid)
            return it.current()->s_connection;
      }
      return NULL;
}
// because the structure data still exist after the thread exit,
so we add this function
void DCOPClient::delGlobalConnection(pthread_t tid)
{
      for(QListIterator     <connection_pid>     it(*cplist);
it.current(); ++it)
            if(it.current()->tid == tid)
                  cplist->remove(it.current());
}
```

As we know, processes use pipe, socket, signal to communicate, the first two can be use directly in the thread, but not the third as it is specific for process. For example, we mention that the main-process will send a SIGTEAM signal to kill a slave-process. In the multi-thread case, not only the slave-thread will be killed, the whole Konqueror/Embedded process will exit as well. If we use SIGSTOP, it would just make the slave-thread stop and hang up, never recycle the resources, the same happens when using the function pthread_cancel(). In order to simulate the main-process uses signal to kill slave-processes, we make use of the pipe built by the system, and add a new command named "end". When we need to kill a slave-thread, main-thread just need to send a command through the pipe and the slave-thread will call function pthread_exit() to quit.

5. Testing and analysis

In this paper, the experiment software environment is the uCLinux operating system, and the hardware environment is EM862xL chipset. A large number of tests proving that the web browser can run on this environment stably and smoothly, and the speed and layout are reasonable and acceptable. So that the rewritten Konqueror/embedded is a good support of uCLinux operating system.

After rewriting, the Konqueror/embedded can support most of the original function, which has good performance and stability advantages while comparing to the other open source web browsers running on uCLinux.

As we all know, the Konqueror/embedded system can not run on the uCLinux operating system. In order to get the comparison of the performance quality, we have to choose a general Linux operating system for test. We also use the official website http://www.sina.com as the testing object for its larger data flow and rich display elements. Finally, we got a group of compared test data: the response time of the rewritten Konqueror/embedded has been reduced by 2.3%, and EMS memory use has been reduced by 6.7%. Considering a large network server load at the same time would inference the response time, so we also provide a group of data displaying website on the web server in our lab intranet for reference, which is: the response time has been reduced by 6.9%, and ENS memory use has been reduced by 31.2%. All the data above prove that the response time and the memory occupancy rate have been decreased.

6. Conclusion

This paper develops an uCLinux-based multi-thread embedded browser. It uses the famous open source Konqueror/embedded as a prototype, elaborates on its bottom network connection layer, and brings out a proposal on how to transplant a multi-process system from the normal Linux system to the uCLinux system which also provides a useful and valuable methodology of how to introduce and develop applications on non-MMU system. After transformation, the Konqueror/embedded browser is able to run on uCLinux stably, and it can support HTML, CSS, JavaScript, DOM and multiple picture formats, support English and Chinese, etc. We will take a further study on the Konqueror/Embedded to utilize thread with its lightweight and strong concurrency features more fully

6. References

[1] Li Chi-ying, "Design and Implementation of embedded browser of integrate multifunction machine", master's degree, East China Normal University, 2006.4

[2] Liu Gang, "Research and develop of the embedded browser based on MiniGUI", master's degree, Middle China University of Science and Technology, 2004.2

[3] "MiniGUI White Paper", Beijing Fei Man Software Technology Limited Company

[4] Zhang Chao, "Study on embedded system construction base on ARM and uCLinux", master's degree, 2005

[5] Lu Yun-kun, Yu Jian, Zhao Li, and Zou Cai-rong, "Transplant the embedded browser Konqueror and make it into Chinese edition", Electronic Design , Southeast University radio engineering school, in 2006 the 12th periods

[6] Jiang Wen-jun, Zhang Xiao-lin, and Cui Ying-wei, "Analysis the technical of embedded browser Konqueror/Embedded", Microcontroller and embedded systems, Beijing University of Voyage, in 2005 the 5th periods

[7] QT reference document,
http://www.qiliang.net/qt/index.html , 2002

[8] Alan Grosskurth and Michael W.Godfrey, "A Reference Architecture for Web Browsers", School of Computer Science University of Waterloo Waterloo,ON N2L 3G1 Canada,IEEE 2005

[9] Toshihiko Yamakami, "A Micro-Component Architecture Approach for Next Generation Embedded Browsers", Research and Development Division, ACCESS 2-8-16 Sarugaku-cho,Chiyoda-ku, Tokyo,Japan ,IEEE 2005