Y86 Simulation Server

多进程的多线程Y86模拟服务器

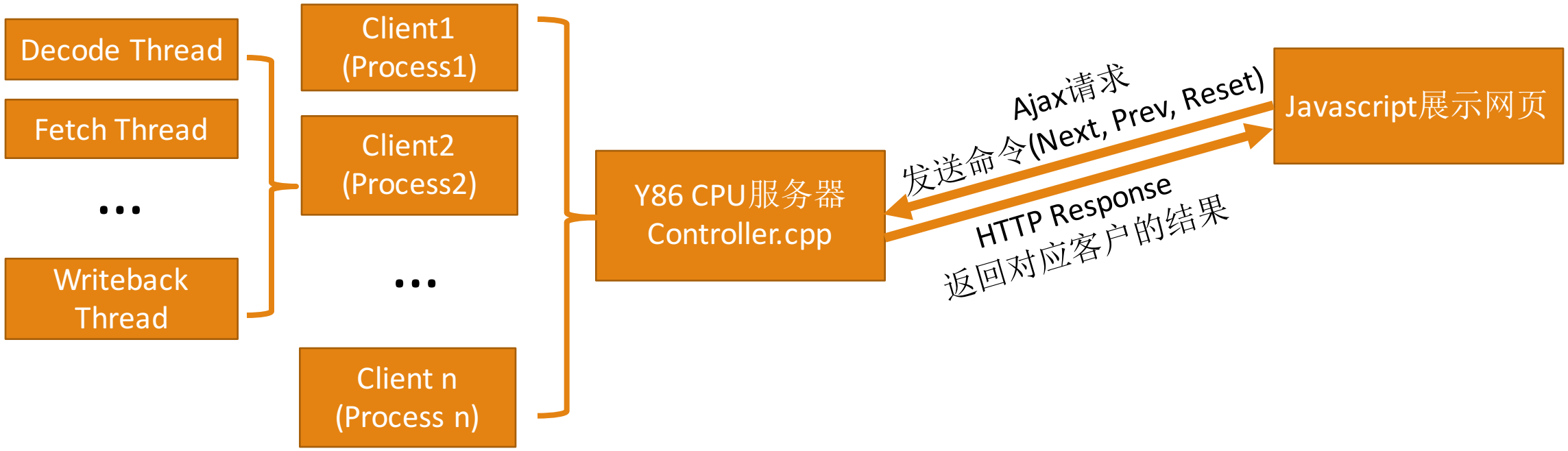
卢力韬 王鹏

实现功能

- 用纯C++实现了一个网络服务器
- 展示每个时钟周期各个阶段寄存器值（网页上）
- 使用多线程进行Y86 CPU的模拟
- 使用多进程来同时模拟不同客户的Y86程序

实现功能

总体架构


Decode Thread
Fetch Thread
...
Writeback Thread
Client1 (Process1)
Client2 (Process2)
...
Client n (Process n)
Y86 CPU服务器 Controller.cpp
Ajax请求
发送命令(Next, Prev, Reset)
HTTP Response
返回对应客户的结果
Javascript展示网页

后端实现

多线程模拟每个CPU时钟周期

- 为提高效率，每个时钟周期，并不重新创建线程。
- 收到next请求后：
  - 同步唤醒5个阶段的线程。
  - 主线程进入休眠
  - 在五个周期的线程执行完毕后唤醒主线程
- 线程实现：`#include <thread>`
- 互斥锁实现：`#include <mutex>`

Code（主线程）

```
4   if (strcmp(uri, "/next") == 0) {
5       if (!ended) {
6           before_wake_thread();
7           f_lock.unlock();
8           d_lock.unlock();
9           e_lock.unlock();
10          m_lock.unlock();
11          w_lock.unlock();
12          fin_lock.lock(); /*wait threads to finish*/
13          PIPE[clock_cnt].stat = PIPE[clock_cnt].W.stat;
14      }
15  }
```

Code（子线程）

```
18  void proc_f() {
19      while (1) {
20          f_lock.lock();
21
22          PIPE[clock_cnt].ConveyF(*last_pipe_p);
23
24          mtx.lock();
25          fin_cnt++;
26          if (fin_cnt == ECHONUM)
27              fin_lock.unlock(); //this wake the webserver thread.
28          mtx.unlock();
29      }
30  }
```

接收JavaScript Ajax传来的HTTP请求

- 使用`uri`区分不同的指令：
  - 如要下一步，前端向`/next`发送`GET`请求
  - 如要上一步，前端向`/prev`发送`GET`请求
  - 等等

接收JavaScript Ajax传来的HTTP请求

- 使用Socket编程与Rio读写包
- 以及CSAPP中提供的工具函数
- `Open_listenfd(port) -> accept(…) -> rio_readn(…) -> rio_writen(…)`
- 根据`Accept()`得到的`hostname`，唤醒或创建对应的进程
- 得到结果后，以`json`格式传回前端`(HTTP Response)`

HTTP Response

- 用Rio读写包向打开的Socket文件写
- HTTP Response的格式：

```
HTTP/1.0 200 OK
Server: Tiny Web Server
Connection: close
Content-length: 699
Content-type: application/json

{"W" : { "begin:" : 0x0, "stall" : 0, "stat" : 1, "icode" : 0, "valE" : 0, "valM" : 0, "dstE" : 0, "dstM" : 0 }
"M" : { "begin:" : 0x0, "bubble" : 0, "stat" : 1, "icode" : 0, "Cnd" : 1, "valE" : 0, "valA" : 0, "dstE" : 15, "dstM"
"E" : { "begin:" : 0x0, "bubble" : 0, "stat" : 1, "icode" : 3, "ifun" : 0, "valC" : 256, "valA" : 0, "valB" : 0, "dst
"D" : { "begin:" : 0x6, "stall" : 0, "bubble" : 0, "stat" : 1, "icode" : 3, "ifun" : 0, "rA" : 15, "rB" : 5, "valC"
"F" : { "stall" : 0, "predPC" : 12 }
"RegisterFiles" : { "eax" : 0, "ecx" : 0, "edx" : 0, "ebx" : 0, "esp" : 0, "ebp" : 0, "esi" : 0, "edi" : 0 }
"stat" : 1
}
```

多进程实现同时模拟不同客户的程序

- 若不进行并发，服务器端无法同时模拟不同客户的程序。
- **但是：**
- HTTP是无状态协议
- 每次Accept一个连接就创建新的进程？
- 这样无法知道这个客户的Clock Count
- 前端传回客户的Clock Count？
- 每次重新模拟，过于低效

解决方法

- 每次Accept一个连接，唤醒对应的子进程，父进程等待子进程处理完毕
- 处理完每个时钟周期，子进程进入休眠
- 怎样将子进程的模拟结果传到父进程？
- **Unix Pipe**

Introduction to Unix Pipe

```
96   int pipefd[2];
97   pipe(pipefd)
98   if (Fork() == 0) { // Child
99       rio_t rio;
100      Rio_readinitb(&rio, pipefd[0]);
101      /* Read Somthing from parent process */
102  }
103  // Parent
104  Rio_writen(pipefd[1], "message", 8);
105  /* send a string "message" to child process*/
106
```

打开两个文件（读和写），将fd存入pipefd数组中

主进程不发送消息，子进程就会在此处阻塞

Code (Server)

```
107  while (1) {
108      建立Socket, 获取hostname等
109      if (!name2pid.count(hostname)) {
110          /*若没有进程负责hostname,需要新建一个进程*/
111          create_new(hostname, connfd);
112      }
113      解析HTTP Request, 获取uri等
114      uri[strlen(uri)] = '\n';
115      proxy(result, hostname, uri);
116      /*"代理"函数转发uri到对应的子进程中,
117        然后从子进程中取回模拟的结果存入result*/
118      serve_json(connfd, result);
119      /*将result以HTTP Response的格式返回给Client*/
120      Close(connfd);
121  }
```

Code (create_new)

```
void create_new(const char *hostname, int connfd) {
    pid_t pid;
    int pipefd1[2], pipefd2[2];
    pipe(pipefd1);pipe(pipefd2);
    if ((pid=Fork()) == 0) { //Child
        Close(connfd);
        创建线程等工作
        while (1) {
            从pipefd1[0]中读取指令
            执行多线程模拟
            将模拟结果写到pipefd2[1]
        }
    }
    // Parent
    name2pid[hostname] = pid;
    name2wfd[hostname] = pipefd1[1];
    name2rfd[hostname] = pipefd2[0];
}
```

Code (proxy)

```
void proxy(char *result, const char *hostname, const char *input) {
    Rio_writen(name2wfd[hostname], input, strlen(input));
    rio_t rio;
    Rio_readinitb(&rio, name2rfd[hostname]);
    Rio_readlineb(&rio, result, MAXLINE);
}
```

Unix Pipe 优势

- 若子进程没有完成模拟，子进程不会输出东西，父进程在读入处就阻塞了，不用复杂的进程Signal控制。
- 同理，子进程的读入处也会阻塞，直到父进程(Proxy 函数)传递指令给子进程
- **为何调用两次pipe()?**
  - Avoid Race!
  - （防止父进程读到自己写给子进程的东西）
  - （子进程同理）

前端实现

- HTML + CSS 完成外观设计
- Javacript 完成内容的展示和变化，以及与服务器的交互
- 跨域问题：返回 jsonp

杂项

C++与C链接？

- 不能直接链接，gcc, g++在编译后.symtab中的格式不同
- e.g. double func()
- gcc: _fun
- g++: _fun_double

解决方法：

```
#ifdef __cplusplus
extern "C" {
#endif
函数声明
#ifdef __cplusplus
}
#endif
```

杂项

- 网络编程调试：
  - telnet

DEMO

Q&A