Proxy Lab

廖丁鸥

Lab目标

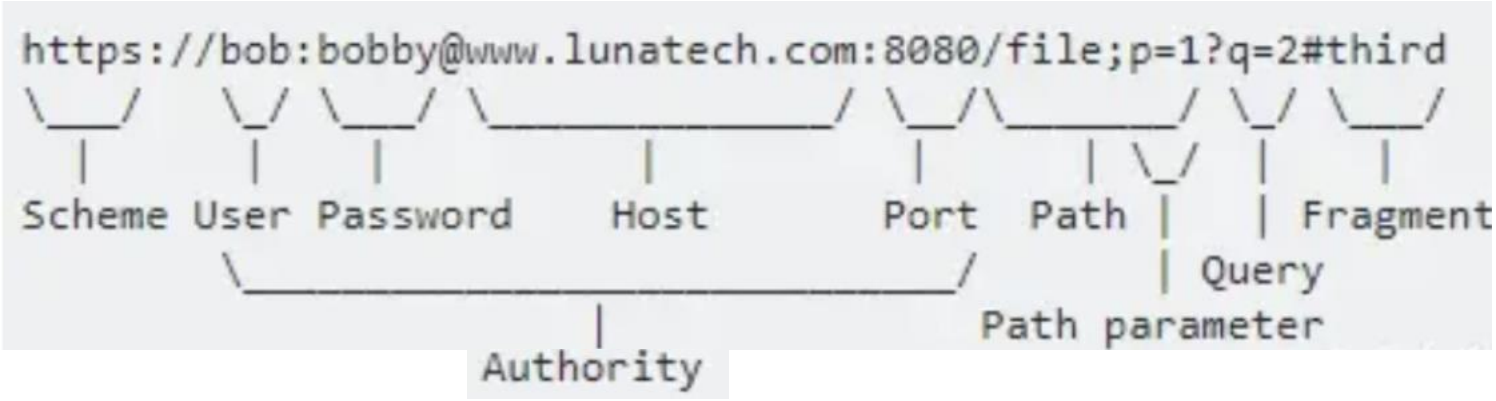
- 编写一个用于缓存Web对象的简单HTTP代理。
- Part1中需要建立代理来接受传入连接、读取和解析请求、将请求转发到Web服务器、读取服务器的响应并将这些响应转发到相应的客户端。需要学习基本的HTTP操作，以及如何使用套接字编写通过网络连接进行通信的程序；
- Part2中需要学习如何处理并发性这一重要系统概念并升级代理以处理多个并发连接；
- Part3中需要使用最近访问的 Web 内容的简单主内存缓存向代理添加缓存。

Part1

- **实现顺序Web代理**
- 启动时代理应侦听端口的传入连接，该端口的编号在命令行中指定。建立连接后代理应读取来自客户端的完整请求并解析该请求。它应确定客户端是否发送了有效的HTTP请求，如确定，它应建立自己与相应Web服务器的连接然后请求客户端指定的对象。最后，代理应该读取服务器的响应并将其转发给客户端。

Part1

- **HTTP/1.0 GET请求**
- 从类似于GET http://www.cmu.edu/hub/index.html HTTP/1.1 读取主机名、端口、路径等信息，然后与服务器建立连接，并接受写回客户端。
- HTTP请求示意图



Part1

- **HTTP/1.0 GET请求**
- 实现思路：
- 逻辑就是先读取，再解析出关键信息，最后拼装成新的请求；
- 读取用到的是Rio_readinitb函数和Rio_readlineb函数；
- 解析需要设置新函数，可参考书中parse_uri函数；
- 拼装放到请求标头部分。

Part1

- **请求标头**
- 在proxy.c文件的开头已经写好了一个客户端发送的特殊的请求头。
- 需要构造新的发送到终端服务器请求。

Part1

- **端口号**
- 在parse_uri函数中实现。
- **doit函数功能实现：**
- 1.读取客户端的请求行，判断其是否是GET请求，若不是，调用clienterror向客户端打印错误信息。
- 2.parse_uri调用解析uri，提取出主机名，端口，路径信息。
- 3.代理作为客户端，连接目标服务器。
- 4.调用build_request函数构造新的请求报文new_request。
- 5.将请求报文build_request发送给目标服务器。
- 6.接受目标服务器的数据，并将其直接发送给源客户端

Part2

- **处理多个并发请求**
- 一旦您有了一个工作的顺序代理，您应该修改它以同时处理多个请求。实现并发服务器的最简单的方法是生成一个新的线程来处理每个新的连接请求。也可尝试其他的设计，如教科书第12.5.5节中描述的预线程服务器。
- 采用多线程方式实现处理多个并发请求，主线程只负责监听窗口，当有一个客户端连接的时候，创建一个分线程去为这个客户端提供服务。但是这种方法的缺点是我们为每一个新客户端创建一个新线程导致不小的代价。

Part2

可以使用如图所示的生产者-消费者模型降低这种开销。

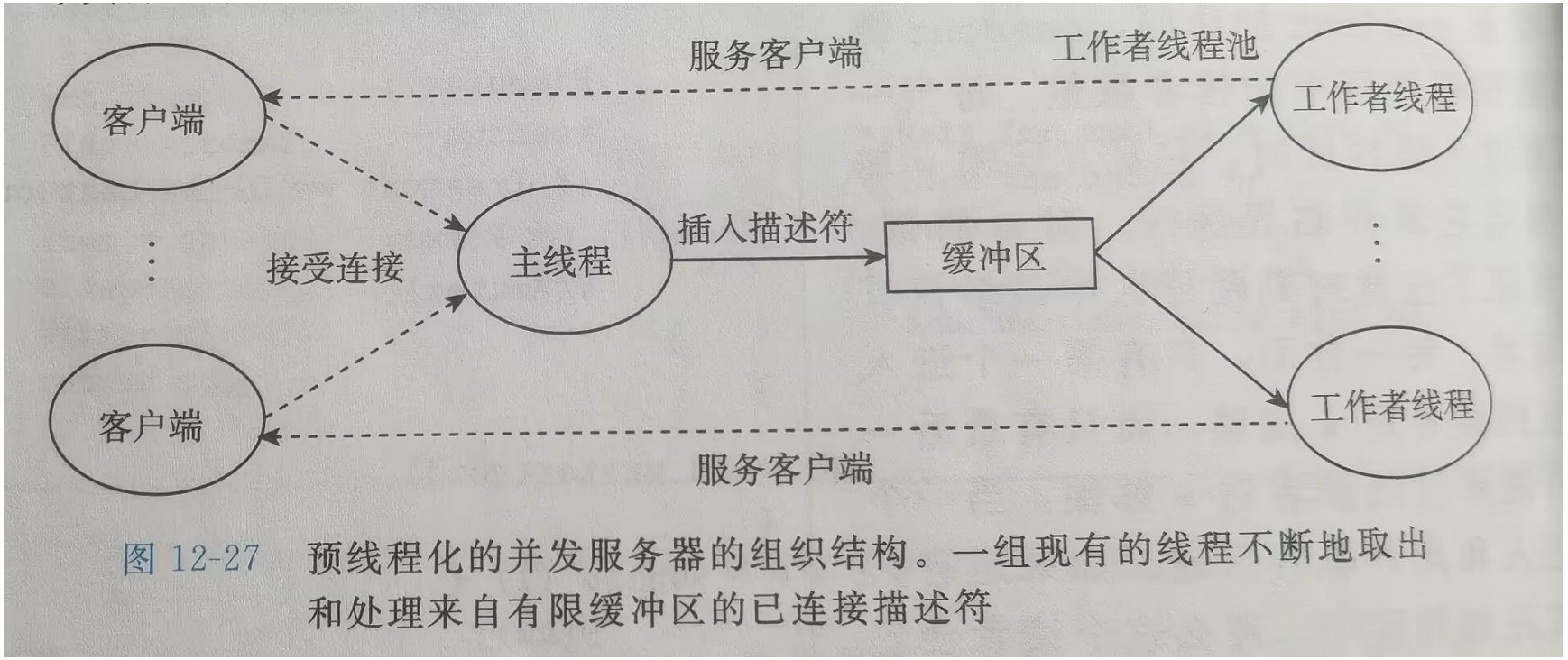


图 12-27 预线程化的并发服务器的组织结构。一组现有的线程不断地取出和处理来自有限缓冲区的已连接描述符

Part2

- **实现思路：**
- 可参考书中709页代码

code/conc/echoservert-pre.c

```
#include "csapp.h"
#include "sbuf.h"
#define NTHREADS  4
#define SBUFSIZE  16

void echo_cnt(int connfd);
void *thread(void *vargp);

sbuf_t sbuf; /* Shared buffer of connected descriptors */

int main(int argc, char **argv)
{
    int i, listenfd, connfd;
    socklen_t clientlen;
    struct sockaddr_storage clientaddr;
    pthread_t tid;

    if (argc != 2) {
        fprintf(stderr, "usage: %s <port>\n", argv[0]);
        exit(0);
    }
    listenfd = Open_listenfd(argv[1]);

    sbuf_init(&sbuf, SBUFSIZE);
    for (i = 0; i < NTHREADS; i++)  /* Create worker threads */
        Pthread_create(&tid, NULL, thread, NULL);

    while (1) {
        clientlen = sizeof(struct sockaddr_storage);
        connfd = Accept(listenfd, (SA *) &clientaddr, &clientlen);
        sbuf_insert(&sbuf, connfd); /* Insert connfd in buffer */
    }
}

void *thread(void *vargp)
{
    Pthread_detach(pthread_self());
    while (1) {
        int connfd = sbuf_remove(&sbuf); /* Remove connfd from buffer */
        echo_cnt(connfd);                /* Service client */
        Close(connfd);
    }
}
```

code/conc/echoservert-pr

Part3

- **缓存web对象**
- 向代理添加一个缓存，该代理将最近使用的Web对象存储在内存中。

Part3

- **最大缓存大小**
- 代理的整个缓存应具有以下最大大小：
  MAX_CACHE_SIZE = 1 MiB
  在计算其缓存的大小时，代理必须只计算用于存储实际web对象的字节；应该忽略任何无关的字节，包括元数据。
- 首先设置好推荐最大缓存和对象大小，文件中已经写好了。我们要做的是设置好cache。

Part3

- **最大对象大小**
- 代理应只缓存不超过以下最大大小的web对象：
- MAX_OBJECT_SIZE = 100 KiB
- 为了方便起见，这两个大小限制都是作为proxy.c中的宏提供的。
- 实现正确缓存的最简单方法是为每个活动连接分配一个缓冲区，并在从服务器接收到的数据时积累数据。如果缓冲区的大小曾经超过了最大的对象大小，则可以丢弃该缓冲区。如果在超过最大对象大小之前读取了web服务器的整个响应，则可以缓存该对象。使用此方案，代理将用于web对象的最大数据量如下，其中T是活动连接的最大数量：
- MAX_CACHE_SIZE + T * MAX_OBJECT_SIZ

Part3

- **驱逐政策**
- 代理的缓存应使用接近最近使用最少的（LRU）驱逐策略的驱逐策略。它不一定是严格的LRU，但它应该是相当接近的东西。注意，读取对象和写入对象都视为使用该对象。
- **同步**
- 对缓存的访问必须是线程安全的，并且确保缓存访问不受竞争条件可能是实验室这一部分更有趣的方面。事实上，有一个特殊的要求，即多个线程必须能够同时从缓存中读取。当然，一次只允许一个线程写入缓存，但是阅读器必须不存在这种限制。
- 因此，使用一个大的独占锁来保护对缓存的访问并不是一个可以接受的解决方案。可探索一些选项，如分区缓存、使用Psthewe读-写器锁或使用信号量来实现自己的读-写器解决方案。在任何一种情况下，都不必执行严格的LRU驱逐策略，这将使我们在支持多个读者方面提供一些灵活性。
- 最后要在doit函数里调用我们写好的cache函数

Part3

- **实现思路：**
- 定义缓存结构（可使用双向链表）
- 实现读者写者问题：需要定义相关变量，以及定义reader和writer函数作为读者和写者。
- int reader(int fd, char *url);其内调用get_cacheData检查是否缓存命中，若是，则将所缓存的数据通过fd发送给客户端，否则返回0表示缓存未命中。
- void writer(char **url*, char **content*);缓存未命中后，与之前一样进行代理服务，从目标服务器接收数据后发送到客户端，如果web object的大小符号要求的话，再调用writer将接收的数据进行缓存