北京郵電大學
BEIJING UNIVERSITY OF POSTS AND TELECOMMUNICATIONS

计算机应用编程

实验

熊永平@网络技术研究院

ypxiong@bupt.edu.cn

周一13:30-15:30@**教三**328

2012.10.8

8000000000000

您如果在谷歌上搜索"邓稼先传"，排名第一的不是某某图书馆或者某某研究所，而是一个叫做"弯曲评论"的网站。再继续搜索"IPv9评判"，弯曲评论仍然高居第一。那么看看"美国大峡谷 长江三峡"呢？您还是首先找到弯曲评论。

弯曲评论听起来似乎有点曲笔论道的涵义。在弯曲评论登场的2008年1月之前，在漫长的半年讨论酝酿期中，几位创办人正是这么想的。这

几位 点江山
激扬 曲评
论就

. 越
英广 毛润
之先 弯曲
评论 流.

因此
这里 动与
设备 丁评的
调笑 新
IPv 工程
进行 外学术

界的华人科学家，成为这方面信息最全面的媒体。作为严肃的评论网站，弯曲评论吸引了来自一百三十多个国家的网友前来访问、发表意见并撰写文章。这些评论字字珠玑，众人拾柴火焰高，让弯曲评论更精彩、更深刻。

弯曲评论的编辑和读者们，在辛苦劳作之余，跨越空间，于此小憩，评理评事评科技，论人论道论潮流，可得天道酬勤之乐；而在祖国和民族复兴的进程中，我们的目标是"铁肩担道义，妙手著文章"；为往圣继绝学，为万世开太平，善莫大焉！

弯曲评论

网址：http://www.tektalk.org

电邮：comments@tektalk.org

共同创办人：

高飞，撰稿人，主编，gaofei@tektalk.org （高飞全部文章）

陈怀临，撰稿人，首席科学家，陈怀临空间，huailin@tektalk.org （怀临全部文章）

杰夫，撰稿人，jiefu@tektalk.org （杰夫全部文章）

青成，撰稿人，qingcheng@tektalk.org （青成全部文章）

课程进度

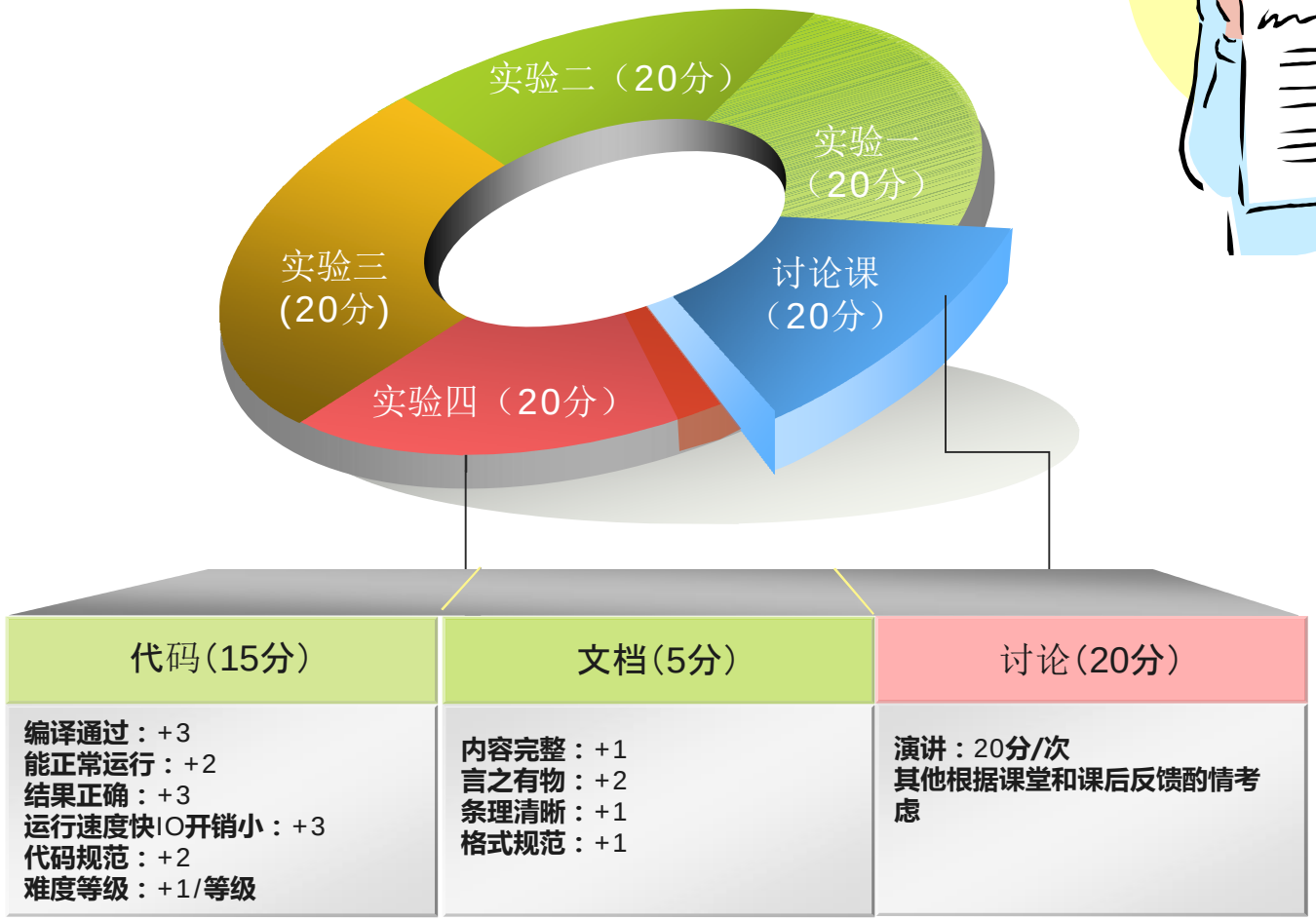
~~开始&作业一(9.3)~~ → ~~实验一讨论(9.17)~~ → 实验二(10.8)

实验二讨论(10.22) → 实验三(11.5) → 实验三讨论(11.19)

实验四（12.3） → 实验四讨论（12.17） → 总结(1.7)


四个实验
多线程、网络编程、索引树……
1. 海量字符串
处理大规模字符串
2. 站点分析器
爬取站点所有页面链接并分析
3. Web服务器
处理高并发的服务器
4. 索引助手
对文件建立索引结构
反馈调整

成绩计算

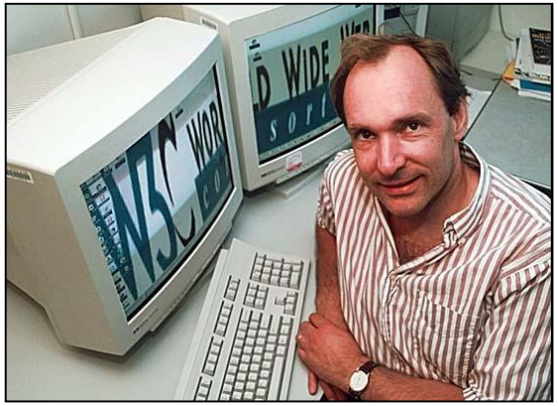

实验二（20分）
实验一
（20分）
实验三
(20分)
讨论课
（20分）
实验四（20分）
代码(15分)
文档(5分)
讨论(20分)
编译通过：+3
能正常运行：+2
结果正确：+3
运行速度快IO开销小：+3
代码规范：+2
难度等级：+1/等级
内容完整：+1
言之有物：+2
条理清晰：+1
格式规范：+1
演讲：20分/次
其他根据课堂和课后反馈酌情考虑

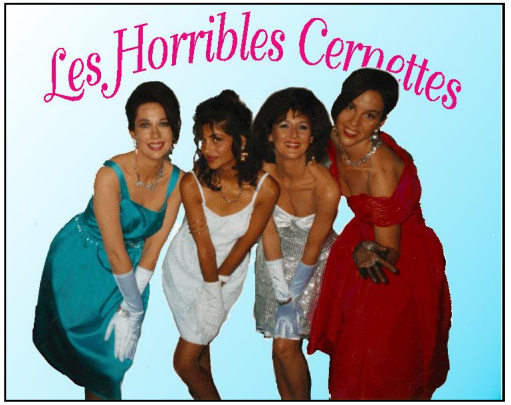
实验二:站点分析器

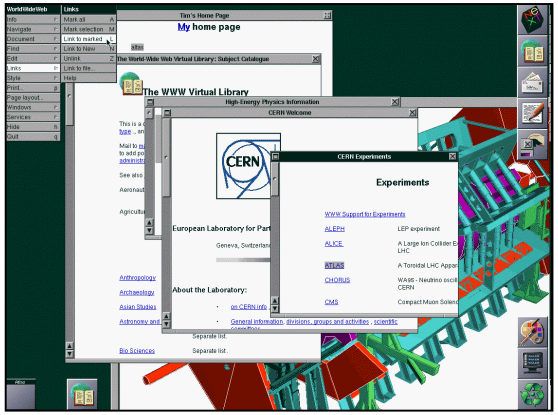
实验目标

- 目标
  - 设计一个站点分析程序，实现
    - 爬取整个网站的页面，去掉无效页面和链接
    - 提取网站页面的链接图
    - 分析页面的链接关系
- 编程技能
  - C语言练习
  - 多线程与线程池
  - 网络编程
  - HTTP协议与爬虫
  - 网站链接图分析
    - PageRank

预备知识—HTML与Web



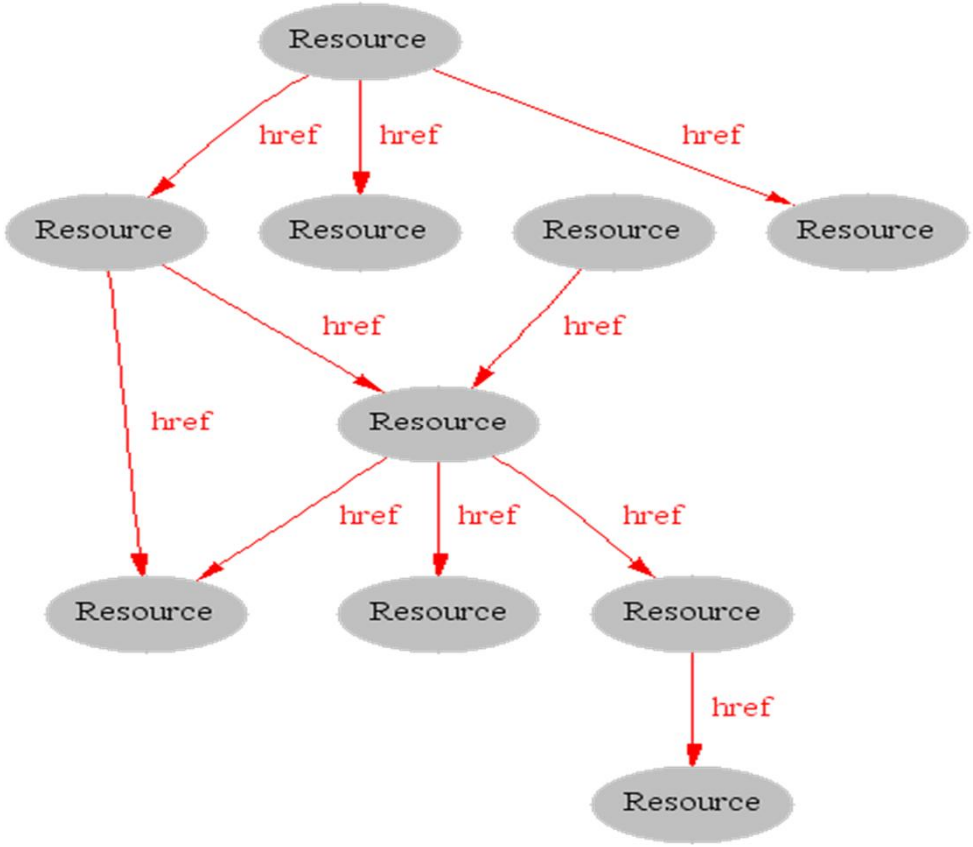




Tim Berners-Lee was knighted by Queen Elizabeth for his invention of the World Wide Web. He is shown here, along with the first picture posted on the Web and a screen shot from an early version of his Web browser.

Web是...

```
<!-- HEADER -->
<div id="header">

    <h1><a href="index.html" onclick="return link(th

    <a class="header-button toggle">Menu</a>

    <!-- NAVIGATION -->
    <div id="nav">

        <ul>
            <li><a href="index.html" onclick="return
            <li><a href="about.html" onclick="return
            <li><a href="portfolio.html" onclick="re
            <li><a href="gallery.html" onclick="retu
            <li><a href="contact.html" onclick="retu
            <li><a href="styles.html" onclick="retur
        </ul>

    </div> <!-- /nav -->
```



预备知识—Web Graph

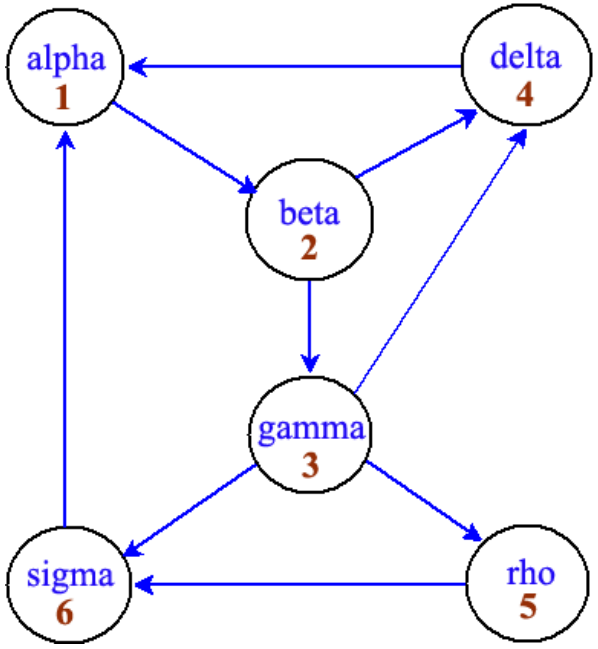
- A graph G = (V,E) is composed of:
  - V: set of vertices
  - E: set of edges connecting the vertices in V
  - An edge e = (u,v) is a pair of vertices
- Web Graph
  - 一个有向图
  - 所有的URL组成的集合作为顶点集
  - An arc $u \rightarrow v$ is identified for each hyperlink from a URL *u* towards a URL *v*.
- **有向图 D 的顶点 v 的出度(out-degree)是指 D 中从 v 邻接的顶点的个数，或以 v 为起点的弧的条数，记作 od(v)**
- **有向图 D 的顶点 v 的入度(in-degree)是指 D 中邻接到 v 的顶点的个数，或以 v 为终点的弧的条数，记作 id(v)**

邻接矩阵

- **定义邻接矩阵**

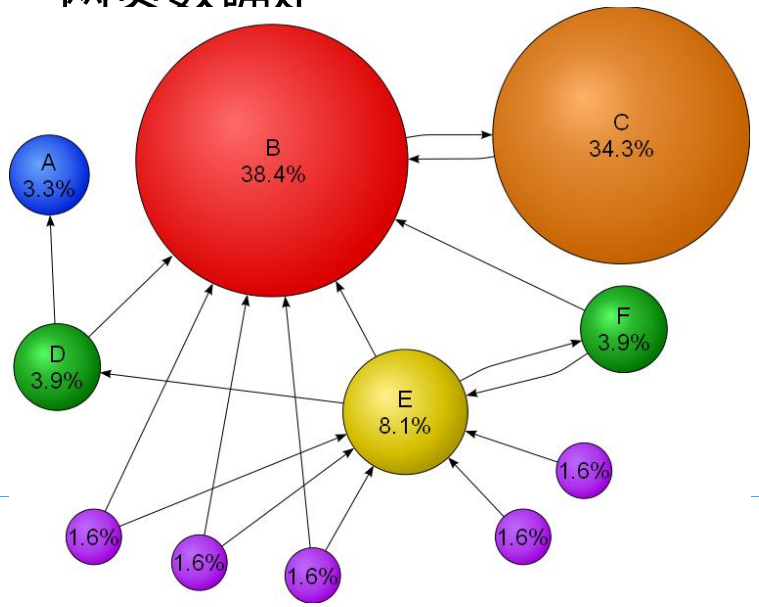
$$G = (g_{ij}), \quad \text{其中 } g_{ij} = \begin{cases} 1, & \text{如果存在从 } j \text{ 到 } i \text{ 的弧} \\ 0, & otherwise \end{cases}$$

$$G = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$



网页权重计算

- 导入链接（入度）
  - 若 B 网页上有连接到 A 网页的链接 ，称 B 为 A 的导入链接
  - 导入链接越多，网页越重要？
- PageRank方法
  - 基于「从许多优质的网页链接过来的网页，必定还是优质网页」的回归关系
  - 网页重要性由链入网页数、链入网页的权重、链入网页的链出网页数确定



链向网页E的导入链接数远大于链向网页C的链接，但是网页C的重要性却大于网页E。这是因为因为网页C被网页B所链接，而网页B有很高的重要性。

PageRank算法

- 基于父页面权重
  - 每一张网页P有一个特定的Rank值r(P)

$$r(P) = \sum_{P \in \mathcal{B}_P} \frac{r(P)}{|P|}$$

$\mathcal{B}_P$ = {all pages pointing to P}

$|P|$ = number of out links from P

将一个网页的所有链入网页的PageRank值除以该链入网页的链出网页数得到的结果进行累加，即得到该网页的PageRank值

对所有网页集合中的任一页i，权重r_j

$$r_j = \sum_i w_{ij} r_i \qquad i,j = 1,\ldots.n$$

e.g., $r_2 = 1 r_1 + 0 r_2 + 0.5 r_3 + 0.5 r_4$

$\underline{r}$ = n x 1 是n个页面的权重向量

L = n x n 链接权重矩阵

=> n个页面的权重计算公式为：

$$\underline{r} = W^T \underline{r}$$

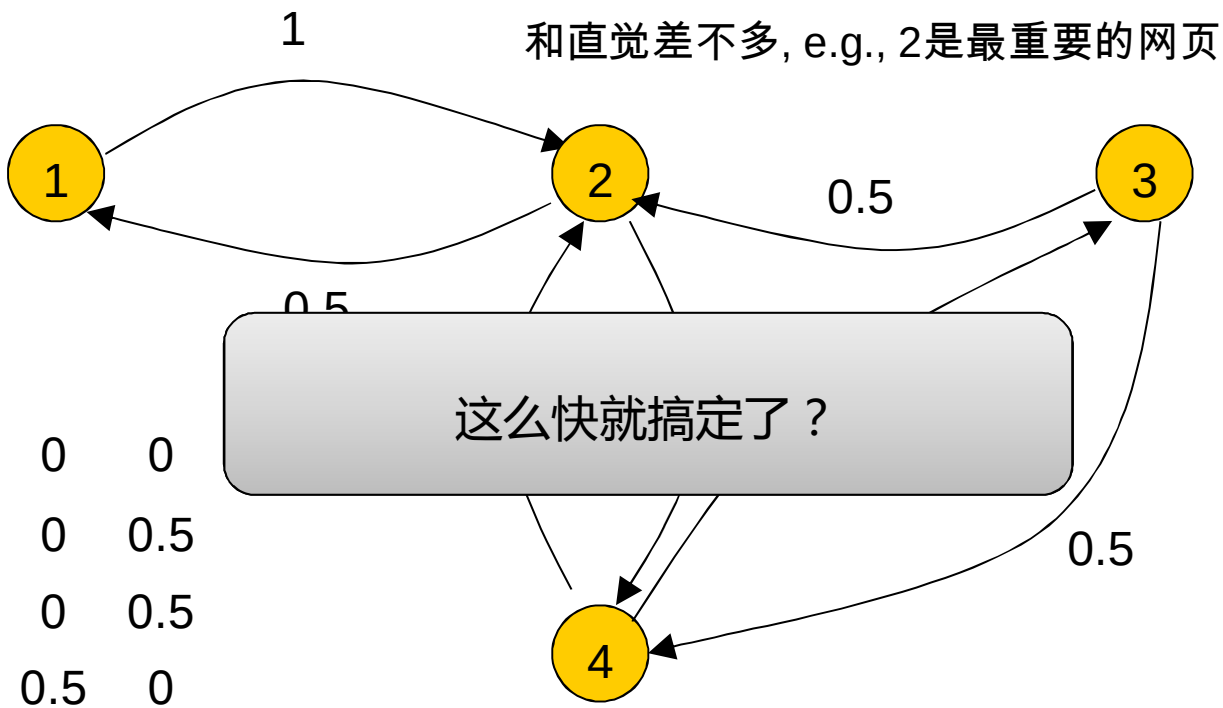
可以看出：r 是 W的对应于特征值为 1 的特征向量

简单计算举例

W的特征向量为

r = [0.2 0.4 0.133 0.2667]

和直觉差不多, e.g., 2是最重要的网页

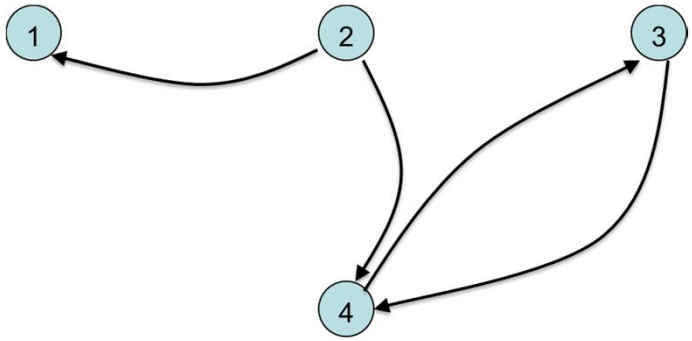


W

0	1	0	0
0.5	0	0	0.5
0	0.5	0	0.5
0	0.5	0.5	0

PageRank的挑战

- 挑战1：矩阵特性
  - 矩阵M一定存在特征值为1的特征向量吗？
  - 必须保证矩阵是不可约的（强连通、非周期）
  - 如右图所示，由于1没有链出页面，3和4形成一个周期，所以该矩阵是可约的，不存在特征值1对应的特征向量



- 挑战2：计算问题
  - 特征向量计算规模是$O(n^3)$
  - 互联网的页面数以百亿计
  - 本实验中是100万个，大规模矩阵的向量计算用普通方法不可行

PageRank应对1—Random Walk模型

- PR的随机游走模型解释
  - 设想有一个永不休止浏览网页的人，随机选择一个链出的链接继续访问。我们问，在稳态情况下（足够长时间后），他会正在看哪一篇网页呢？
  - 等价于：稳态情况下，每个网页v会有一个被访问的概率，p(v)，它可以作为网页的重要程度的度量，依赖于上一个时刻到达“链向”v的网页的概率，以及那些网页中超链的个数。
- 修正模型
  - 让浏览者每次以一定的概率（1-α）沿着超链走，以概率（α）选择一个新的起始节点

$$r = \left[(1-\alpha)W^T + \frac{\alpha}{n}(1_N)\right] r$$

一般取0.15

各元素均为1的N阶矩阵

  - α选在0.1和0.2之间，被称为阻尼系数
  - $G=(1-\alpha)W^T+\alpha/N(1_N)$被称为Google Matrix

PageRank应对1’

- 问题转化为求解方程组r=G·r，且r满足 $\sum_{i=1}^{n} r_i = 1$
  - p有稳态解，需要G满足两个条件
    - 强连通
    - 非周期
  - 这两个条件都可以通过增加的阻尼系数，即以概率 β 重新随机选择一个新的起始节点继续进行游走，来得到实现。
- 根据Perron-Frobenius定理

$$\boldsymbol{x = A x}\text{，}\boldsymbol{x}\text{ 满足：}\sum_{i=1}^{n} x_i = 1$$

  - 该方程组解存在且唯一
    - 而且是A的最大特征值1所对应的特征向量

PageRank应对2-Power Iteration

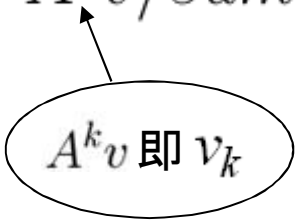
当矩阵 A 的阶很大，无法直接计算其特征值和特征向量时，需要使用该方法

- 幂迭代方法具体计算步骤

1) 输入矩阵 A 和迭代初始向量 v，以及精度 $\varepsilon > 0$（例如0.0001，向量各元素对应差值绝对值），令 $k = 0$；

2) 计算：$v_{k+1} = Av_k$；

3) $x = A^k v / sum(A^k v)$，则计算 PageRank 值并停机。否则转第二步。



PageRank算法只有两步：构造矩阵A和迭代求解

PageRank应对2’

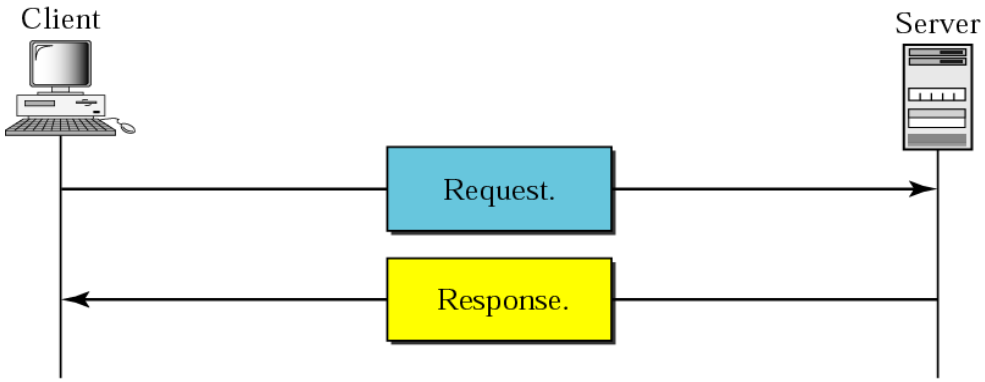
- 幂迭代算法
  - 确定合适的初始向量$\underline{r}^{(1)}$，例如all entries = indegree(node)/|E|
  - 每次迭代是一次矩阵向量乘法复杂度>$O(n^2)$，但W是稀疏矩阵，所以整个迭代速度非常快

  - rate of convergence depends on the “spectral gap”
    -> how quickly does error(k) = $(\lambda_2/\lambda_1)^k$ go to 0 as a function of k ?
    -> if $|\lambda_2|$ is close to 1 (= λ_1) then convergence is slow
  - 3亿个页面的Web Graph
    -> 50 iterations to convergence (Brin and Page, 1998)
- 幂迭代算法的马尔科夫链模型
  - W is a stochastic matrix (rows sum to 1) by definition，can interpret W as defining the transition probabilities in a Markov chain
  - w_{ij} = probability of transitioning from node i to node j
  - Markov chain interpretation:

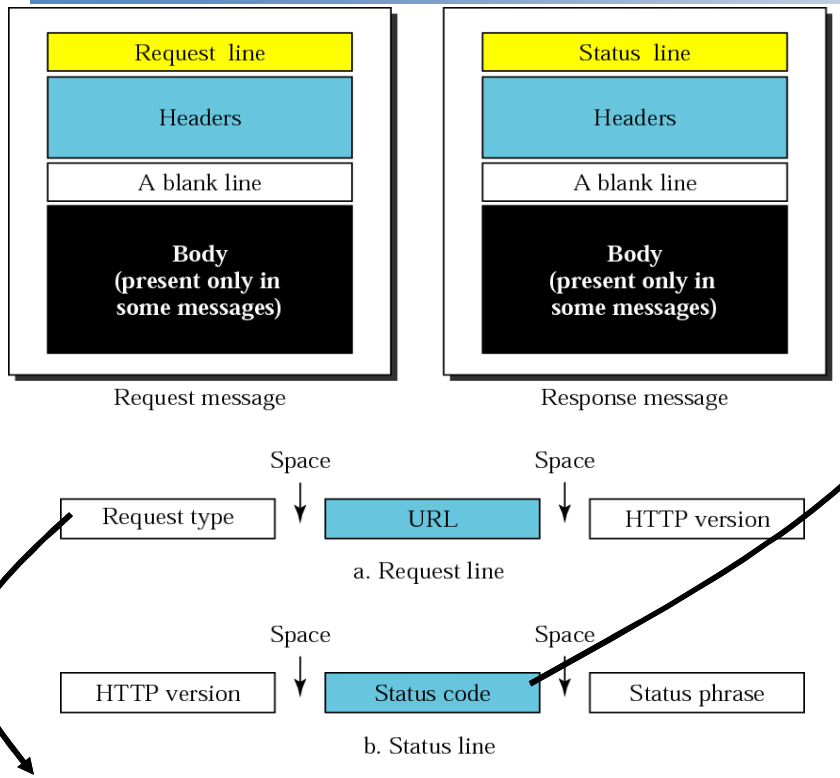
    $$\underline{r} = W^T \underline{r}$$
  - -> these are the solutions of the steady-state probabilities for a Markov chain
  - page importance ⇔ steady-state Markov probabilities ⇔ eigenvector

预备知识—HTTP

- HTTP layered over bidirectional byte stream
  - TCP port 80
  - RFC 1945
- 功能
  - 比之前的FTP简单有效
  - 客户可以从Web服务器上下载几乎所有类型的文件，包括HTML文件，图像/视频/音频等多媒体文件，Java Applet等对象，甚至应用程序等。
- C/S模式



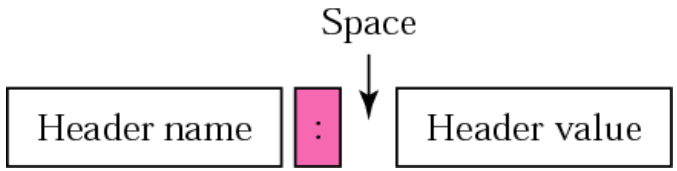
HTTP请求和响应



Method	*Action*
GET	Requests a document from the server
HEAD	Requests information about a document but not the document itself
POST	Sends some information from the client to the server
PUT	Sends a document from the server to the client
TRACE	Echoes the incoming request
CONNECT	Reserved
OPTION	Enquires about available options

Code	*Phrase*	*Description*
	Informational	
100	Continue	The initial part of the request has been received and the client may continue with its request.
101	Switching	The server is complying with a client request to switch protocols defined in the upgrade header.
	Success	
200	OK	The request is successful.
201	Created	A new URL is created.
202	Accepted	The request is accepted, but it is not immediately acted upon.
204	No content	There is no content in the body.
	Redirection	
301	Multiple choices	The requested URL refers to more than one resource.
302	Moved permanently	The requested URL is no longer used by the server.
304	Moved temporarily	The requested URL has moved temporarily.
	Client Error	
400	Bad request	There is a syntax error in the request.
401	Unauthorized	The request lacks proper authorization.
403	Forbidden	Service is denied.
404	Not found	The document is not found.
405	Method not allowed	The method is not supported in this URL.
406	Not acceptable	The format requested is not acceptable.
	Server Error	
500	Internal server error	There is an error, such as a crash, at the server site.
501	Not implemented	The action requested cannot be performed.
503	Service unavailable	The service is temporarily unavailable, but may be requested in the future.

HTTP Header格式



Request headers

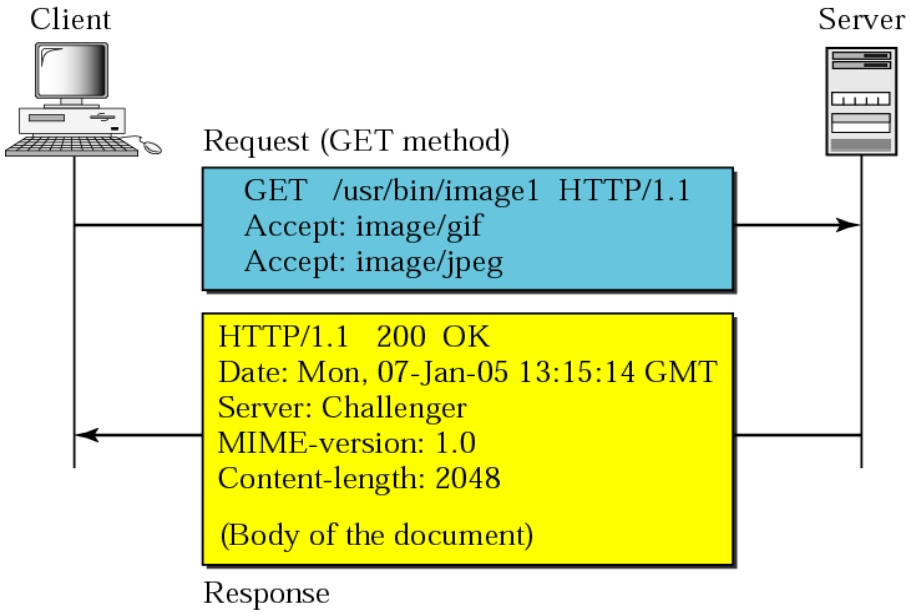
Header	*Description*
Accept	Shows the media format the client can accept
Accept-charset	Shows the character set the client can handle
Accept-encoding	Shows the encoding scheme the client can handle
Accept-language	Shows the language the client can accept
Authorization	Shows what permissions the client has
From	Shows the e-mail address of the user
Host	Shows the host and port number of the client
If-modified-since	Send the document if newer than specified date
If-match	Send the document only if it matches given tag
If-non-match	Send the document only if it does not match given tag
If-range	Send only the portion of the document that is missing
If-unmodified-since	Send the document if not changed since specified date
Referrer	Specifies the URL of the linked document
User-agent	Identifies the client program

Response headers

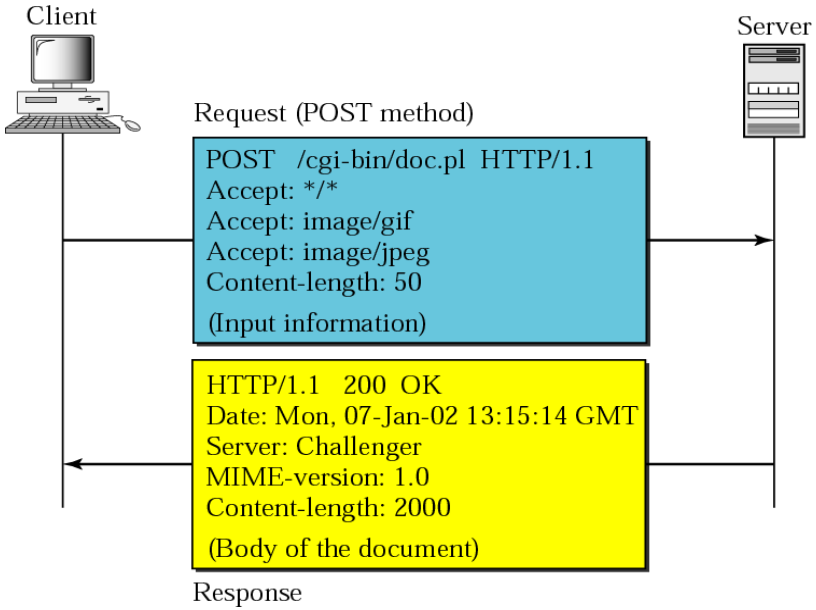
Header	*Description*
Accept-range	Shows if server accepts the range requested by client
Age	Shows the age of the document
Public	Shows the supported list of methods
Retry-after	Specifies the date after which the server is available
Server	Shows the server name and version number

HTTP Example

使用**GET**方法获取图像**/usr/bin/image1**，客户可以接收**GIF** 或**JPEG**格式。



客户机使用**POST**方法发送数据给服务器，服务器使用**perl**脚本处理数据。



预备知识—多线程

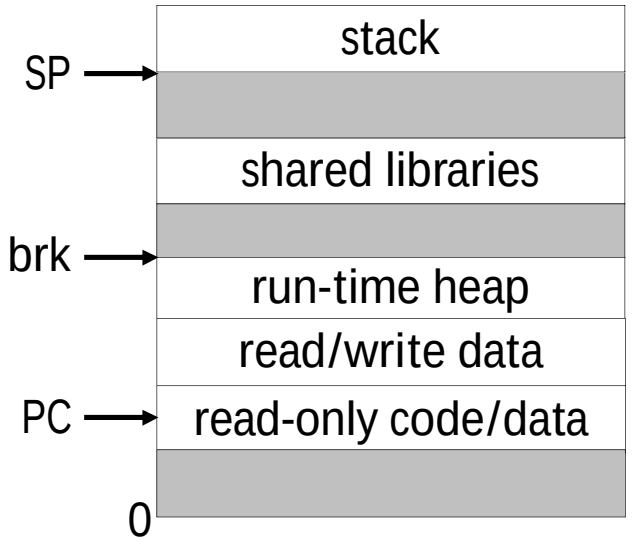
- Process = process context + code, data, and stack

Process context

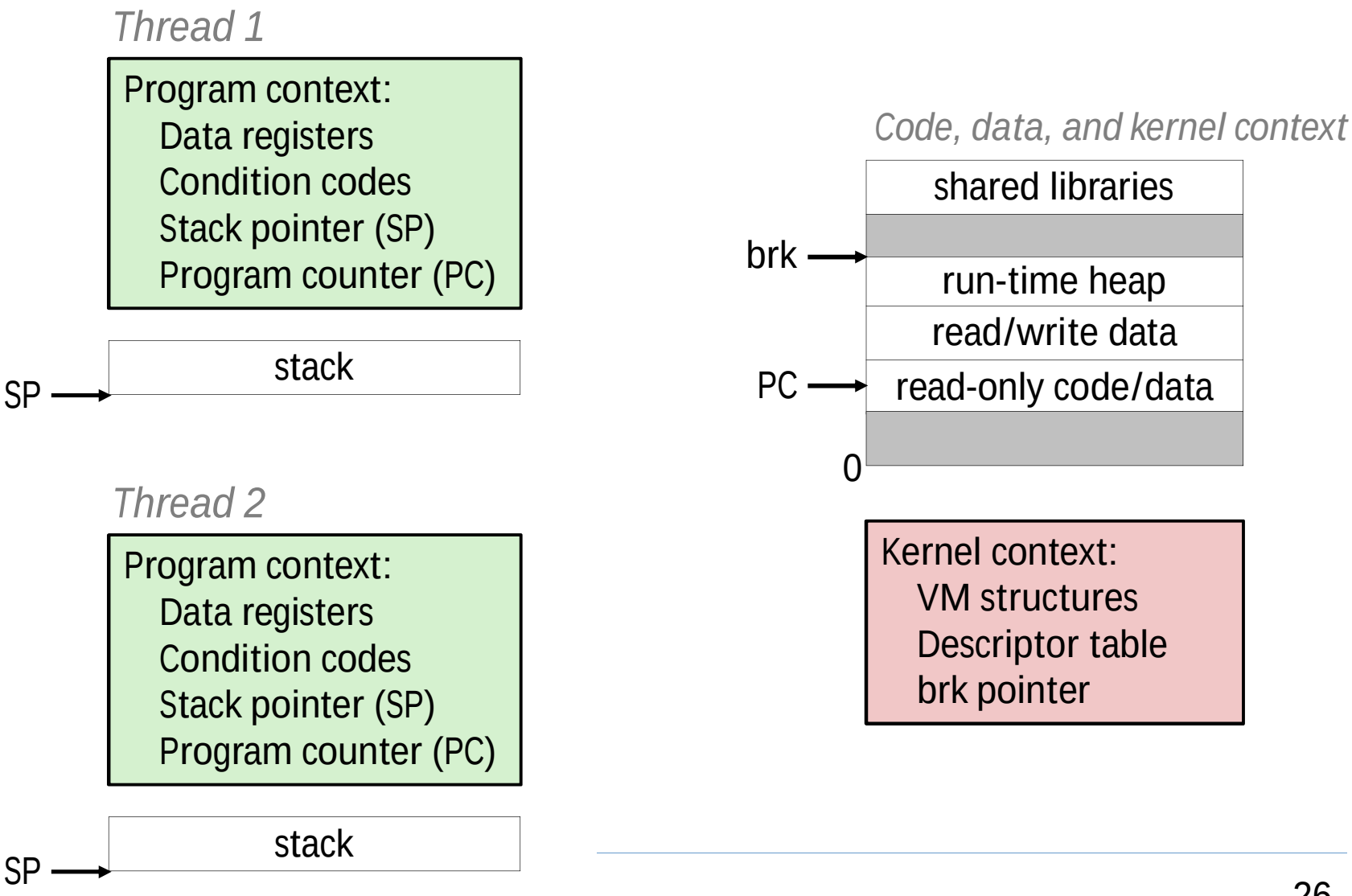
Program context:
- Data registers
- Condition codes
- Stack pointer (SP)
- Program counter (PC)

Kernel context:
- VM structures
- Descriptor table
- brk pointer

Code, data, and stack



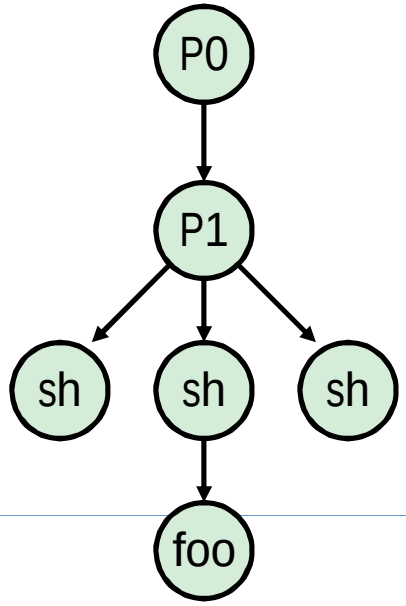
Process with Two Threads



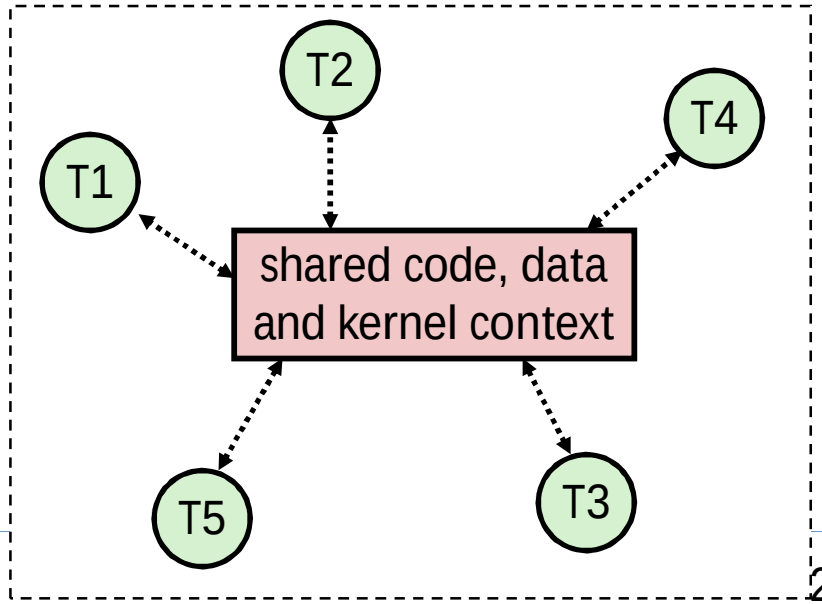
Threads vs. Processes (contd.)

- Processes form a tree hierarchy
- Threads form a pool of peers
  - Each thread can kill any other
  - Each thread can wait for any other thread to terminate
  - Main thread: first thread to run in a process

Process hierarchy



Thread pool

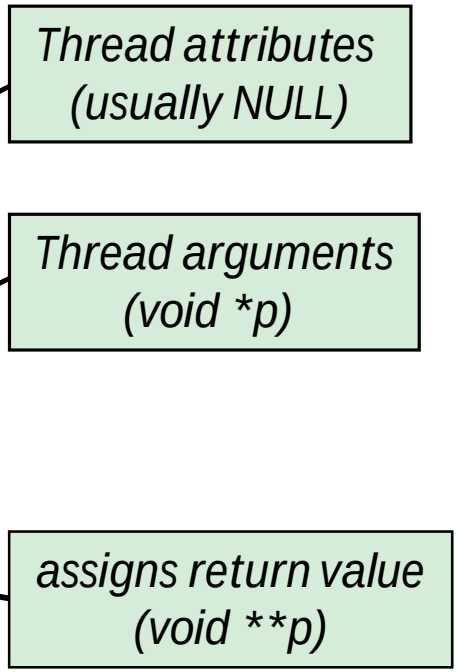


Posix Threads 接口

- *Pthreads:* ~60个控制线程的标准C接口函数
  - Threads run thread routines:
    - **`void *threadroutine(void *vargp)`**
  - Creating and reaping threads
    - **`pthread_create(pthread_t *tid, …, func *f, void *arg)`**
    - **`pthread_join(pthread_t tid, void **thread_return)`**
  - Determining your thread ID
    - **`pthread_self()`**
  - Terminating threads
    - **`pthread_cancel(pthread_t tid)`**
    - **`pthread_exit(void *tread_return)`**
    - **`return`** (in primary thread routine terminates the thread)
    - **`exit()`** (terminates all threads)
  - Synchronizing access to shared variables

The Pthreads “Hello, world" Program

```
/*
 * hello.c - Pthreads "hello, world" program
 */
#include "csapp.h"

void *thread(void *vargp);

int main() {
  pthread_t tid;

  Pthread_create(&tid, NULL, thread, NULL);
  Pthread_join(tid, NULL);
  exit(0);
}

/* thread routine */
void *thread(void *vargp) {
  printf("Hello, world!\n");
  return NULL;
}
```



线程内存模型

- Conceptual model:
  - Multiple threads run within the context of a single process
  - Each thread has its own separate thread context
    - Thread ID, stack, stack pointer, program counter, condition codes, and general purpose registers
  - All threads share the remaining process context
    - Code, data, heap, and shared library segments of the process virtual address space
    - Open files and installed handlers
- Operationally, this model is not strictly enforced:
  - Register values are truly separate and protected, but
  - Any thread can read and write the stack of any other thread
- *Mismatch between the conceptual and operation model is a source of confusion and errors*

内存共享

Global var: 1 instance (`ptr` [data])

Local vars: 1 instance (`i.m, msgs.m`)

Local var: 2 instances (
`myid.p0` [peer thread 0's stack],
`myid.p1` [peer thread 1's stack]
)

```
char **ptr;  /* global */

int main()
{
    int i;
    pthread_t tid;
    char *msgs[2] = {
        "Hello from foo",
        "Hello from bar"
    };
    ptr = msgs;

    for (i = 0; i < 2; i++)
        Pthread_create(&tid,
            NULL,
            thread,
            (void *)i);
    Pthread_exit(NULL);
}
```

```
/* thread routine */
void *thread(void *vargp)
{
    int myid = (int)vargp;
    static int svar = 0;

    printf("[%d]: %s (svar=%d)\n",
        myid, ptr[myid], ++svar);
}
```

Local static var: 1 instance (`svar` [data])

内存同步访问机制

- 多线程编程的主要问题
  - 是对共享数据的保护，即：在多个线程同时访问同一个数据时应保证数据读写的安全。

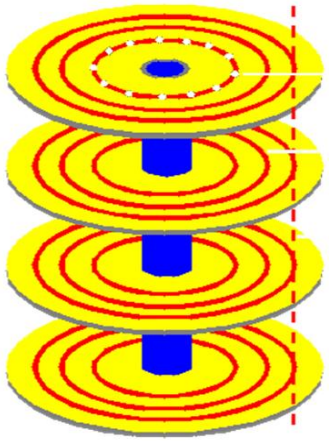
- pthread线程一般通过三种机制来实现对共享数据的保护：
  - 线程互斥锁 (pthread mutex)（本实验使用）

```
pthread_mutex_t mutex1;

        .

        .

pthread_mutex_init(&mutex1, NULL);
```

```
pthread_mutex_lock(&mutex1);

        .

   critical section

        .

pthread_mutex_unlock(&mutex1);
```

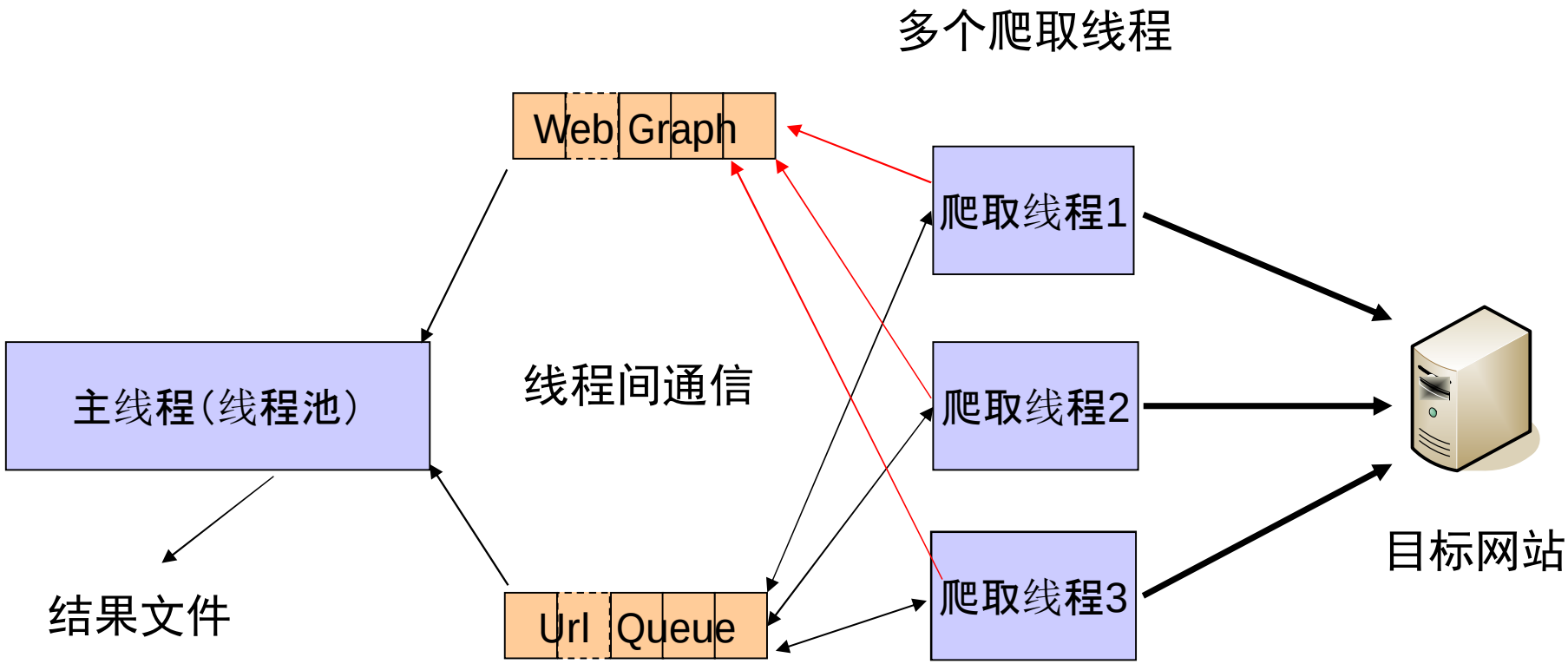
  - 线程条件变量(pthread cond)
  - Posix信号量

多线程可能的问题



- 多线程的性能代价（performance penalty）
  - 内存地址空间不够
  - 互斥
    - 共享的工作数据区数据结构的并发访问，用程序的方式协调这种访问的开销可能会更大（线程互斥，或者进程通信，都很复杂）
  - 磁盘访问代价
    - 并发线程在完成抓取后，进行文档存储时会形成大量交叉的，随机磁盘I/O，外部表现就是磁盘不断的寻道，很慢。

系统架构


多个爬取线程
Web Graph
爬取线程1
主线程(线程池)
线程间通信
爬取线程2
目标网站
结果文件
Url Queue
爬取线程3

爬取模块设计

- 包括
  - crawler_thr.c
    - 爬取站点链接的核心代码
  - network.c
    - 对网络socket的收发进行封装
  - http_client.c
    - 构造HTTP请求
    - 解析HTTP响应格式
    - 获取HTTP服务器返回的内容
  - link_parser.c
    - 识别给定字节流的编码格式（utf8 or gb2312）并进行解码
    - 提取html中的超文本链接

crawler_thr核心流程

- **接口**

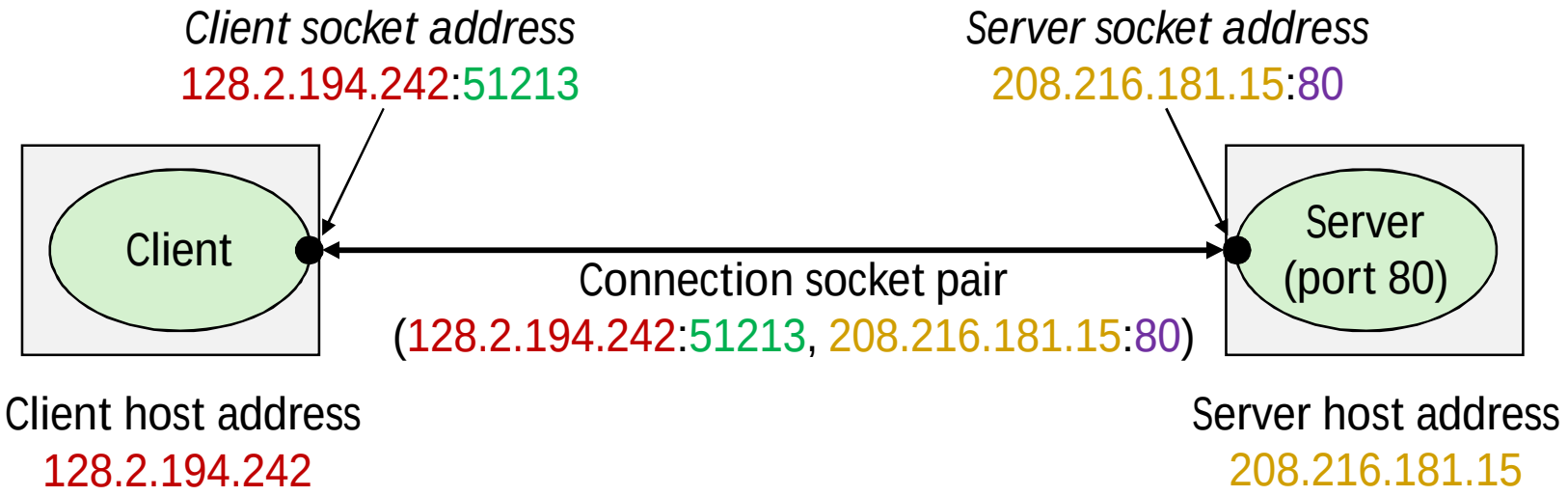
```
crawler_do()
soc := open_tcp(TargetSite)
While URLQUEUE is not empty,
    URL_curr := deque(URLQUEUE)
    PAGE := get_urlcontent(URL_curr)
    COLLECTION URLLIST := extract_link(PAGE)
    For every URL_i in URLLIST,
            insert_edge(WEBGRAPH,<URL_curr,URL_i>)
            If URL_i not in WEBGRAPH.vertex,
                    enque(URL_i, URLQUEUE)
            End
    End
    insert(URL_curr, WEBGRAPH.vertex)
End
Return
```

network模块设计

- 接口
  - open_tcp():建立一个到远程服务器的socket连接
  - nsend（ ）发送字节到网络
  - nrecv（ ）接收网络字节流
  - recv_line()接收一行
  - close（ ）关闭socket

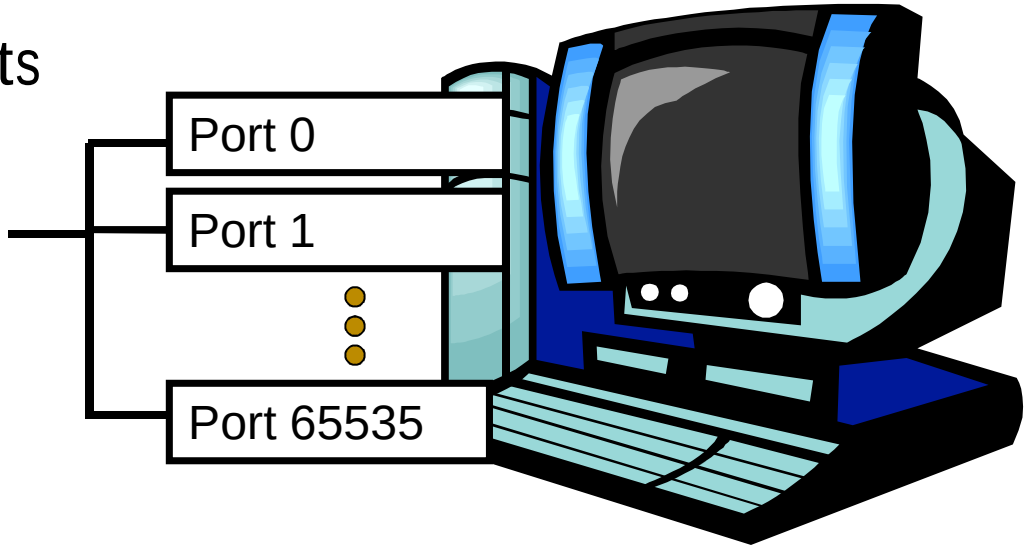
Socket网络编程概念

- What is a socket?
  - To the kernel, a socket is an endpoint of communication
  - To an application, a socket is a file descriptor that lets the application read/write from/to the network
- Clients and servers communicate with each other by reading from and writing to socket descriptors

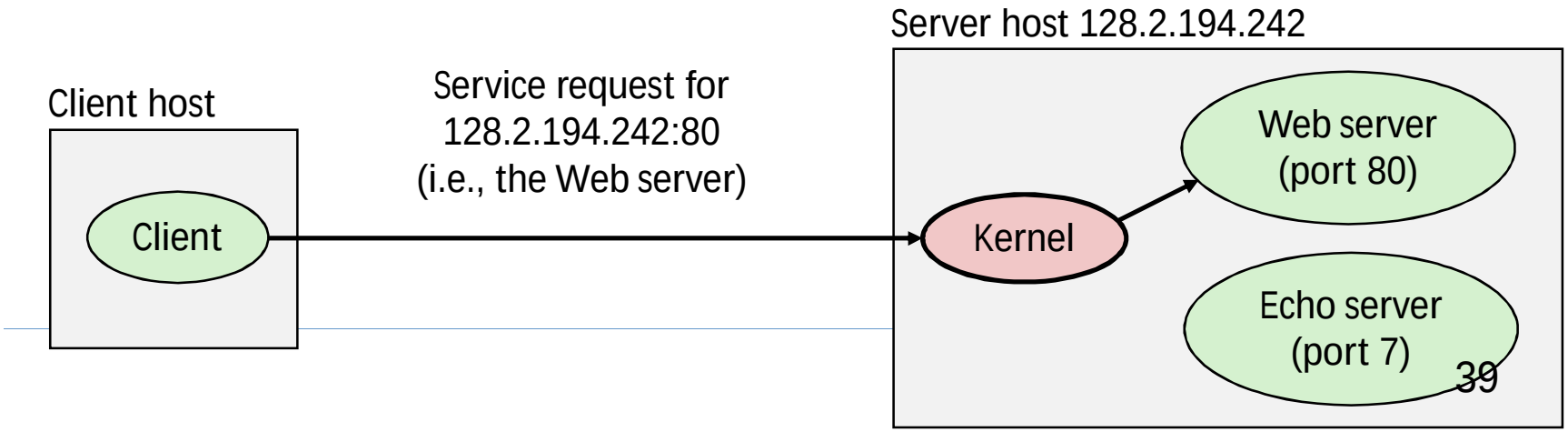


端口概念

- Each host has 65,536 ports
- Using Ports to Identify Services
  - 20,21: FTP
  - 23: Telnet
  - 80: HTTP
  - see RFC 1700 (about 2000 ports are reserved)

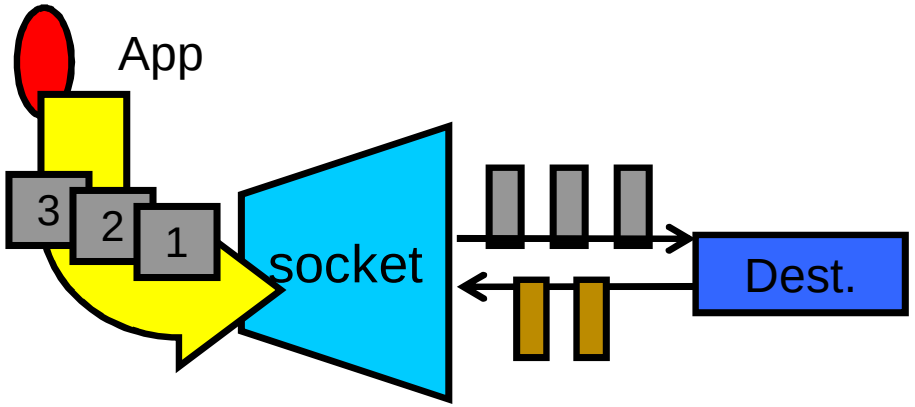


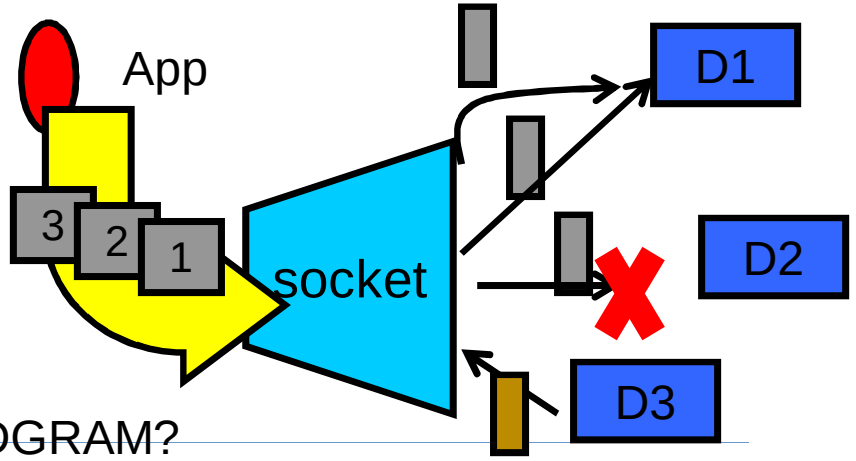
- A socket provides an interface to send data to/from the network through a port



两种基本sockets

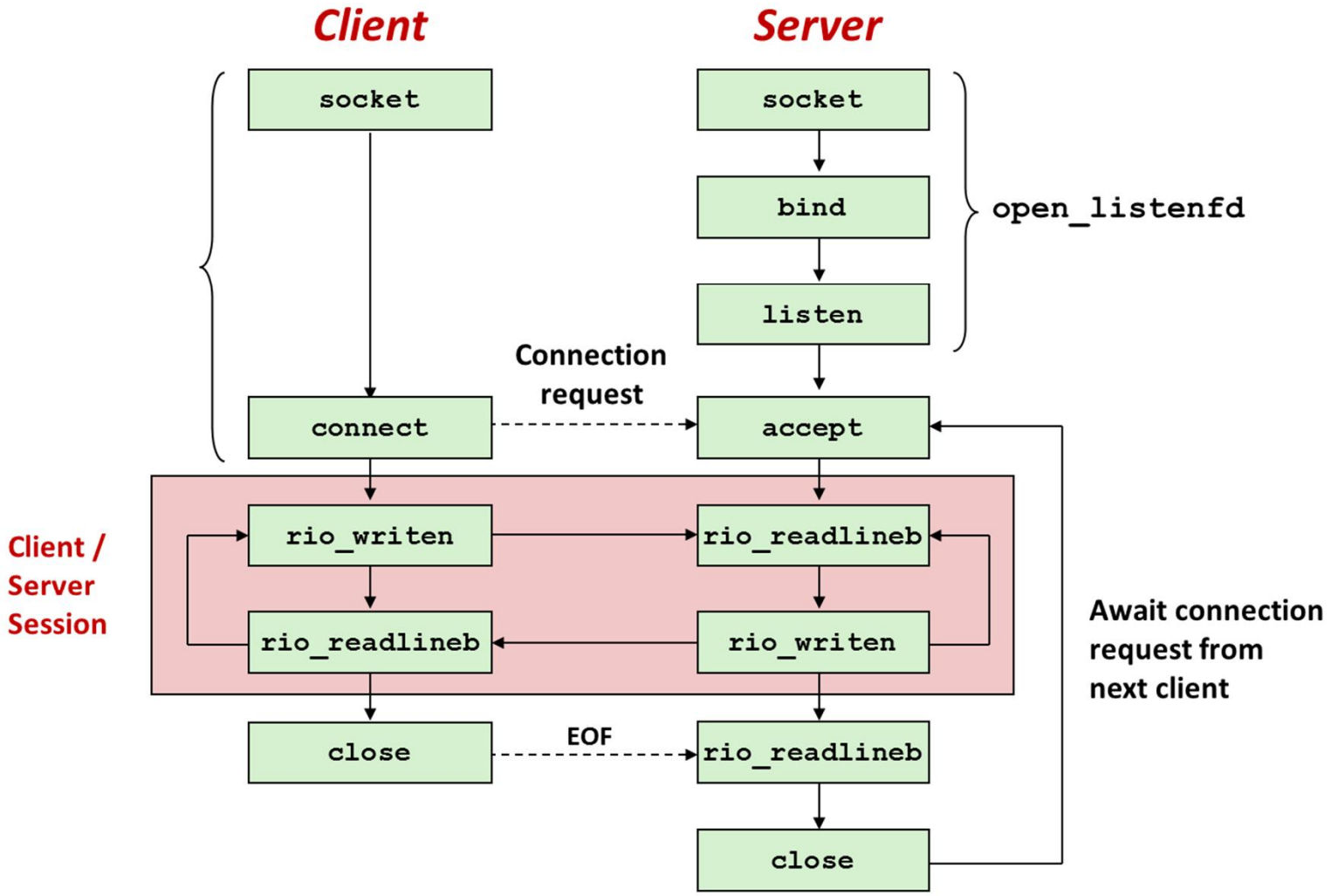
- SOCK_STREAM
  - a.k.a. TCP
  - reliable delivery
  - in-order guaranteed
  - connection-oriented
  - bidirectional
- SOCK_DGRAM
  - a.k.a. UDP
  - unreliable delivery
  - no order guarantees
  - no notion of “connection” – app indicates dest. for each packet
  - can send or receive





Q: why have type SOCK_DGRAM?

Socket接口框架


Client
Server
socket
socket
bind
open_listenfd
listen
Connection
request
connect
accept
Client /
Server
Session
rio_writen
rio_readlineb
rio_readlineb
rio_writen
Await connection
request from
next client
close
EOF
rio_readlineb
close

open_tcp例子

```
int open_tcp(char *hostname, int port) {
  int clientfd;
  struct hostent *hp;
  struct sockaddr_in serveraddr;
  if ((clientfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    return -1; /* check errno for cause of error */
   /* Fill in the server's IP address and port */
  if ((hp = gethostbyname(hostname)) == NULL)
    return -2; /* check h_errno for cause of error */
  bzero((char *) &serveraddr, sizeof(serveraddr));
  serveraddr.sin_family = AF_INET;
  bcopy((char *)hp->h_addr_list[0],
        (char *)&serveraddr.sin_addr.s_addr, hp->h_length);
  serveraddr.sin_port = htons(port);
  /* Establish a connection with the server */
  if (connect(clientfd, (SA *) &serveraddr,
      sizeof(serveraddr)) < 0)
    return -1;
  return clientfd;
}
```

This function opens a connection from the client to the server at `hostname:port`

Create socket

Create address

Establish connection

http_client模块

- 接口
  - http_do_get(): 用get命令请求远程服务器的网页
  - ~~http_do_post(): 用post方式请求远程服务器网页~~
  - http_response_status(): 远程WEB服务器的http响应代码，如404
  - http_response_body(): 获取http响应的消息体字节流

```
int http_do_get(int socket, char *path )
{
   char req[1024];
    sprintf(req, “GET %s HTTP/1.0\r\n", path);
    sprintf(req, "Host: %s\r\n", host);
    sprintf(req, "User-Agent: " USER_AGENT "\r\n");
    sprintf(req, "Content-Type: application/x-www-form-urlencoded\r\n");
    sprintf(req, "\r\n");
    Send(socket, req);
}
```

link_parser模块设计

- 接口
  - extract_link(char *htmltxt)
- 功能
  - 自动识别http响应的HTML字节流的编码格式
  - 能提取其中的html的链接
    - <a href=....>
- 方法
  - 1、正则表达式
  - 2、String match “<a href=”, use strstr() function
  - 3、利用htmlparser库将整个字节流解析成一棵标签树

主线程核心流程

- 核心流程

  初始化URLQUEUE

  初始化WEBGRAPH

  构造一个线程池，创建爬取线程，

  enque(rooturl, URLQUEUE)

  While (True),

  If URLQUEUE *not empty*

  唤醒置位idle在等待任务的爬取线程

  else if 所有爬取线程都在等待任务且URLQUEUE为空

  爬取完成

  Break

  usleep(100) 等待小会儿

  WEBGRAPH.analyze()调用分析模块输出分析结果

线程池大小？

- 获取一个网页的时间？
  - 局域网的延迟在1-10ms，带宽为10-1000Mbps,Internet的延迟在100-500ms，带宽为0.010-2 Mbps
  - 在一个网络往返时间RTT为200ms的广域网中，服务器处理时间SPT为100ms，那么TCP上的事务时间就大约500ms（2 RTT+SPT）
  - 网页的发送是分成一系列帧进行的，则发送1个网页的时间量级在(13KB/1500B) * 500ms ≈4s
- 北邮校园网
  - RTT为50ms，一个网页传输约1.5s
  - 1M*1.5s≈17天
- 线程数量计算
  - 平均1.5秒抓取时间下，按带宽60%利用计算，可完成1.5*0.6*100Mb/(8*13KB) ≈ 900个网页
  - 约启动900个线程

主线程—线程池设计

- 数据结构
  - Threads包含创建thread的参数，由pthread_create创建

```
typedef struct _threadpool_st {
int num_threads;      //number of threads
pthread_t *threads;  //pointer to threads
pthread_mutex_t waitlock; //等待URL队列任务
int idle; //当url队列为空时置位为1，否则为0
}_threadpool;
```

- 线程同步
  - idle和waitlock用于主线程和抓取线程同步
  - 爬取线程在URLQUEUE为空时，置idle为1，并且trylock(waitlock)是否已锁，如果锁了则lock，进入休眠
  - 主线程不断检查是否URLQUEUE为空，如果是，且p线程idle置位，则锁住p线程的waitlock，如果队列不为空，查看线程idle，如果置位了则unlock该线程的waitlock

线程间通信模块

- 包括
  - urlqueue.c
    - 构造和维护待下载的url队列
    - 确保多线程安全访问
  - webgraph.c
    - 构造、维护和存储整个web graph的顶点和边数据
    - 分析web graph的链接关系
    - 确保多线程安全访问

urlqueue模块设计

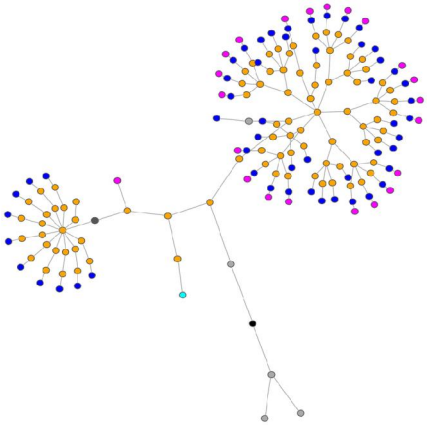
- 接口
  - init_urlq(): 初始化队列
  - enque_urlq(): 入队
  - deque_urlq(): 出队
  - size(): 队列总大小
  - num(): 当前元素个数
  - destroy_urlq(): 销毁队列
- 要点
  - 建议基于环形队列实现，url长度不大于256byte
  - 注意利用mutex在合适地方加锁，确保临界区最短

webgraph模块设计

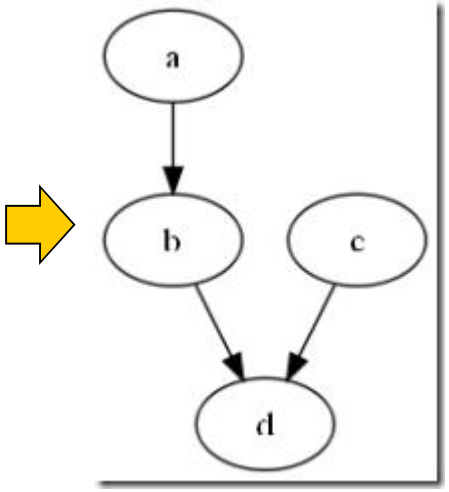
- 接口
  - init_webg(): 初始化web graph
  - insert_vertex(char * url): 加入一个顶点
  - has_vertex(char *url): 判断给定顶点是否在graph中
  - insert_edge(): 插入一条边到graph中
  - vertex_size(): graph的顶点数
  - calc_ind_cdf(): 计算所有顶点的入度的累积分布
  - calc_pagerank(): 计算所有顶点的pagerank
  - find_ind(char *url): 查找给定顶点的入度
  - gen_graphviz(): 生成描述图拓扑的graphviz脚本
  - destroy_webg(): 销毁web graph
- 要点
  - 建议基于环形队列实现，url长度不大于256byte
  - 注意利用mutex在合适地方加锁，确保临界区最短

实验环境

- 校内服务器
  - IP:
  - Web 服务器：apache
- 网站来源
  - 大约10万个网页
  - 部分链接页面不可用，需要剔除
- 网站链接图
  - graphviz是贝尔实验室开发
  - 使用dot脚本语言加布局引擎
  - 语法简单



```
1: digraph
abc{
2: a;
3: b;
4: c;
5: d;
6:
7: a -> b;
8: b -> d;
9: c -> d;
10: }
```



实验说明

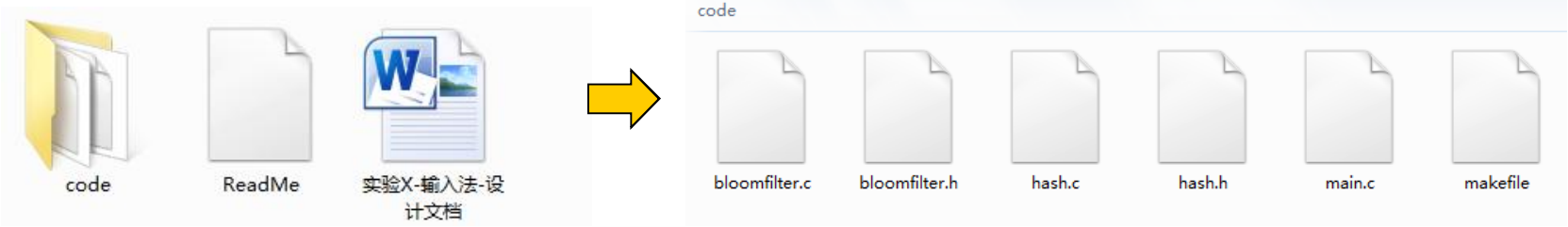
- 运行要求

./siteanalyzer http://192.168.1.2/index.html checkedurls.dat
第一个参数：网站的入口url
第二个参数：待检测的url列表，一行一个
输出以下4个文件：
sitemap.dot: 描述网站拓扑图的graphviz脚本
indcdf.dat: graph的入度cdf，两列，第一列从1到最大的入度，第二列度数对应的CDF（0-1分布）
top10.dat: graph的入度和pagerank的前十的url，共20行2列，1-10行是url 对应入度值
11-20行是url和对应的pagerank值
checkresult.dat: graph中对应输入文件checkedurls.dat的入度和pagerank值，分3列显示，第一列url，第二列入度，第三列pagerank

- 参数设计建议
  - PageRank的α取值0.15，迭代精度 ε 取值0.0001
  - 线程池线程个数和URLQUEUE大小据机器速度和网速调整

代码与文档格式要求！！

- 提交格式
  - Code目录（文件名不能修改）
    - main.c http_client.c link_parser.c network.c crawler_thr.c webgraph.c urlqueue.c
    - Makefile编译规则文件，可执行文件名siteanalyzer
  - Readme
    - 说明程序的编译和使用方法
  - 实验XXXXX设计文档.doc
    - Graphviz**生成的网络拓扑图、入度**CDF**图放入文档**




```
# ./bfchecker urlpoolfullname checkedurlfullname
```


THE END