Socket 应用编程实验

武庆华
wuqinghua@ict.ac.cn

提纲

- Socket 简介
- Socket 应用实现
- 附件文件列表

Socket API

- BSD Socket API
  - 不是为每个应用程序定义接口，而是提供最基本的通信功能
  - 对上层提供统一的调用接口，支持丰富的上层应用开发

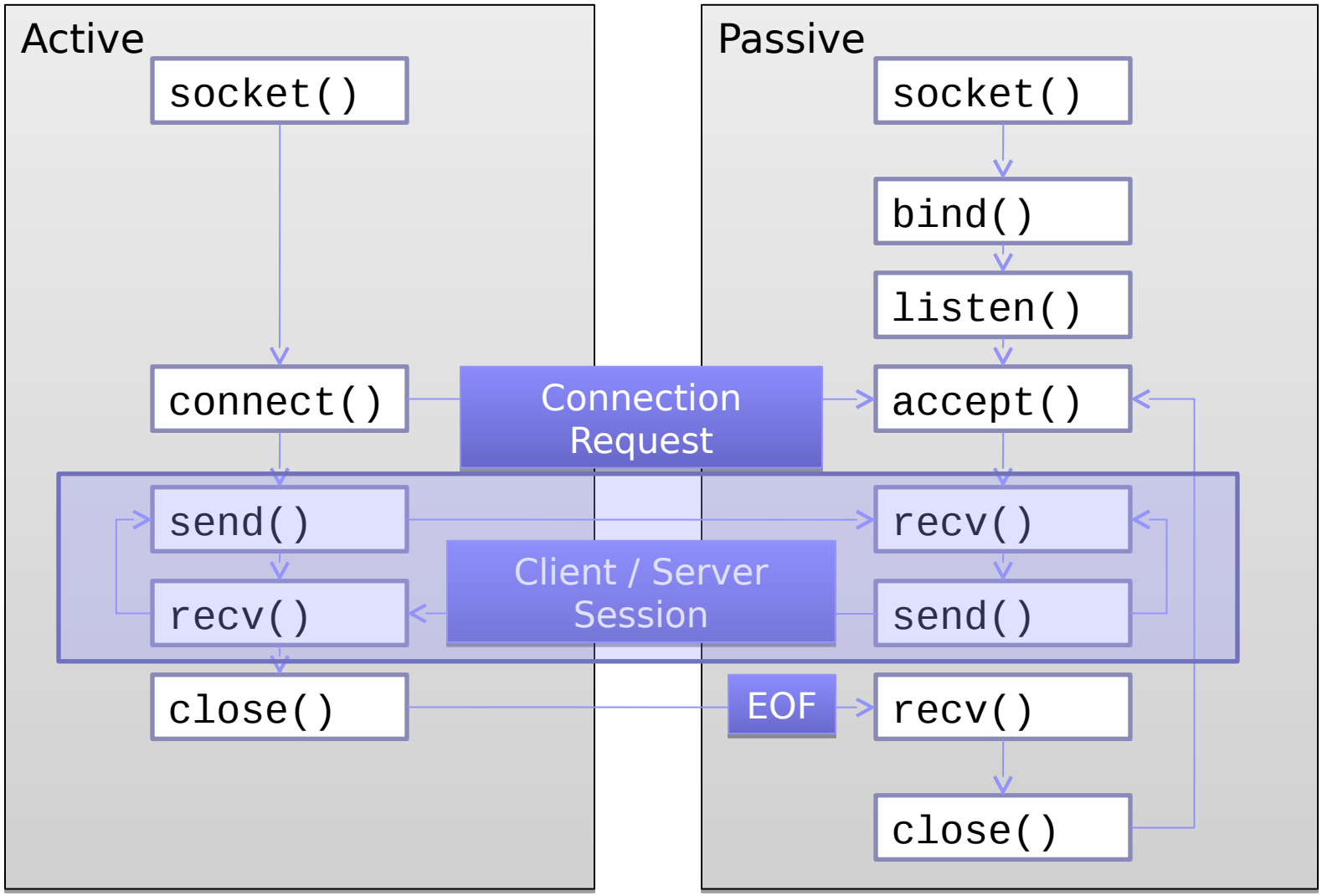
BSD Socket API

socket(domain, type, proto);
close(sockfd);
bind(sockfd, addr, addrlen);

Datagram (Connectionless)
sendto(sockfd, buf, len, flags, dest_addr, addrlen); recvfrom(sockfd, buf, len, flags, src_addr, addrlen);
Stream (Connection-oriented)

listen(sockfd, backlog);
accept(sockfd, addr, &addrlen);
connect(sockfd, addr, addrlen);
send(sockfd, buf, len, flags);
recv(sockfd, &buf, len, flags);

Socket 调用流程


Active
socket()
connect()
send()
recv()
close()
Passive
socket()
bind()
listen()
accept()
recv()
send()
recv()
close()
Connection Request
Client / Server Session
EOF

一、建立 socket 句柄

- 数据收发两端都需要建立 socket 句柄
  - `int socket(int domain, int type, int protocol);`
- Domain: `AF_INET`
- Type
  - `SOCK_STREAM` TCP
  - `SOCK_DGRAM` UDP
- Protocol: `0`
- 返回值 : socket 文件句柄

```
int sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

二、将 socket 句柄与监听地址绑定

- 只需要被动建立连接一方进行绑定（bind）
  - `int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);`
- sockfd: Socket 句柄
- addr: 需要绑定的地址和端口
- addrlen: 地址和端口数据结构的长度

```
struct sockaddr_in server;
server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons(8888);

bind(sockfd, (struct sockaddr *)&server, sizeof(server));
```

三、进行监听

- 只在被动建立连接一方进行监听，等待新的连接请求
  - `int listen(int sockfd, int backlog)`
- sockfd: 之前建立的 socket 句柄
- backlog: 可以理解为最大并发连接数

```
listen(sockfd, 128);
```

四、接受连接请求

- 被动建立连接一方需要显式的接受连接请求
  - `int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);`
- sockfd: 之前建立的 socket 句柄
- addr: 用于存储对端网络地址的数据结构
- addrlen: 指定 addr 大小
- 返回值为该连接对应的文件句柄，以后收发数据都使用该句柄

```
int csock = accept(sockfd, (struct sockaddr_in *)
          &caddr, &clen);
```

五、发送建立连接请求

- 连接的另一方需要主动建立连接
  - `int connect(int sockfd, const struct sockaddr * addr, socklen_t addrlen);`
- sockfd: 之前建立的 socket 句柄
- addr: 对端的网络地址（包括 IP 地址和端口号）
- addrlen: addr 的大小（不同协议的 addr 大小不同）
- 返回值指示连接是否建立成功

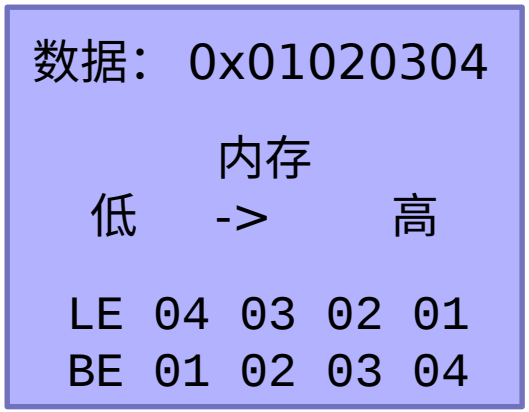
```
struct sockaddr_in dst;
dst.sin_family = AF_INET;
dst.sin_addr.s_addr = inet_addr("10.0.0.1");
dst.sin_port = htons(8888);

connect(sock, (struct sockaddr *)&dst, sizeof(dst));
```

六、数据传输

- 数据发送方
  - `int send(int sockfd, const void *buf, size_t len, int flags);`
- 数据接收方
  - `int recv(int sockfd, void *buf, size_t len, int flags);`
- 连接任意一端都可以发送或者接收数据
- 对于一个阻塞式 (blocking) socket
  - send 直到所有数据被拷贝到协议栈缓存中才返回
  - recv 直到接收到数据 (不一定等于 len) 或产生错误才返回

网络字节序与本地字节序

- 网络协议指定网络传输数据时，使用网络字节序（即大端字节序），地址低位存储值的高位
- Intel 的 x86 平台使用的是小端字节序，地址低位存储值的低位



为了保证数据在不同主机之间传输时能够被正确解释，主机在发送和接收数据时，需要进行网络字节序与主机字节序之间的相互转换：

```
htons()/htonl()
ntohs()/ntohl()
```

只有在收发整数数据时才需要转换字节序，收发字符串时不需要关心

七、处理并发服务请求

如果一台服务器需要同时处理多个服务请求

- 使用多进程 / 多线程技术

```
int request = accept(sock, addr, len);
pthread_create(new_thread, NULL, handle_request, request);
```

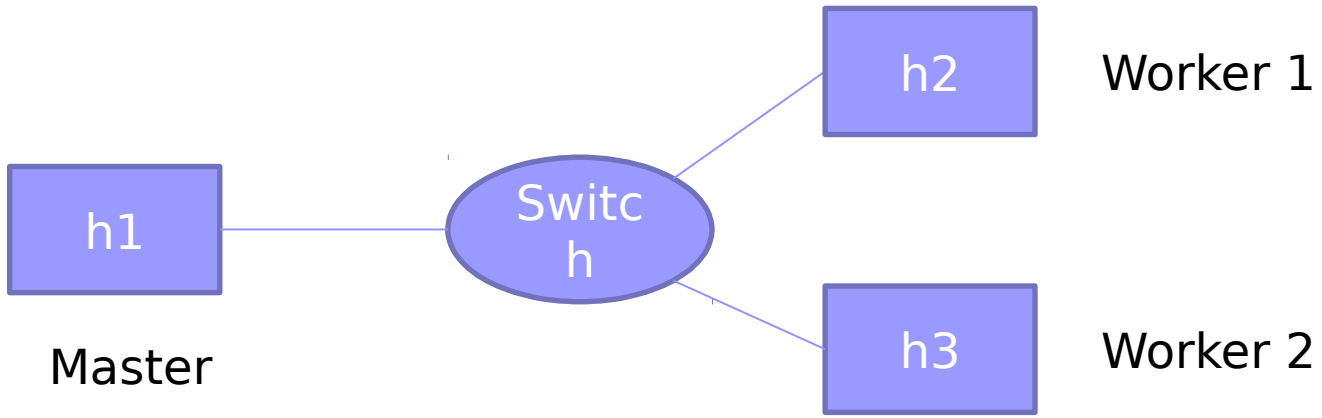
  - 需要维护多个线程，增加系统开销

- 或 使用 I/O 多路复用技术

```
for (int i = 0; i < nfds; i++)
    FD_SET(fd[i], &rfds);
int ready = select(nfds, &rfds, NULL, NULL, timeout);
for (int i = 0; i < nfds, i++)
    if (FD_ISSET(fd[i], &rfds)) recv(fd[i], buffer, len, 0);
```

  - select 最多支持 1024 个并发连接，且每次都需要遍历每一个连接，用 epoll 可以避免该限制

基于 socket 的分布式字符统计程序



- workers.conf 配置文件中存储所有 worker 的 IP 地址
- 需要统计的字符存放在 war_and_peace.txt 文件中
- Worker 1 和 Worker 2 分别监听端口 12345

Master 分发任务

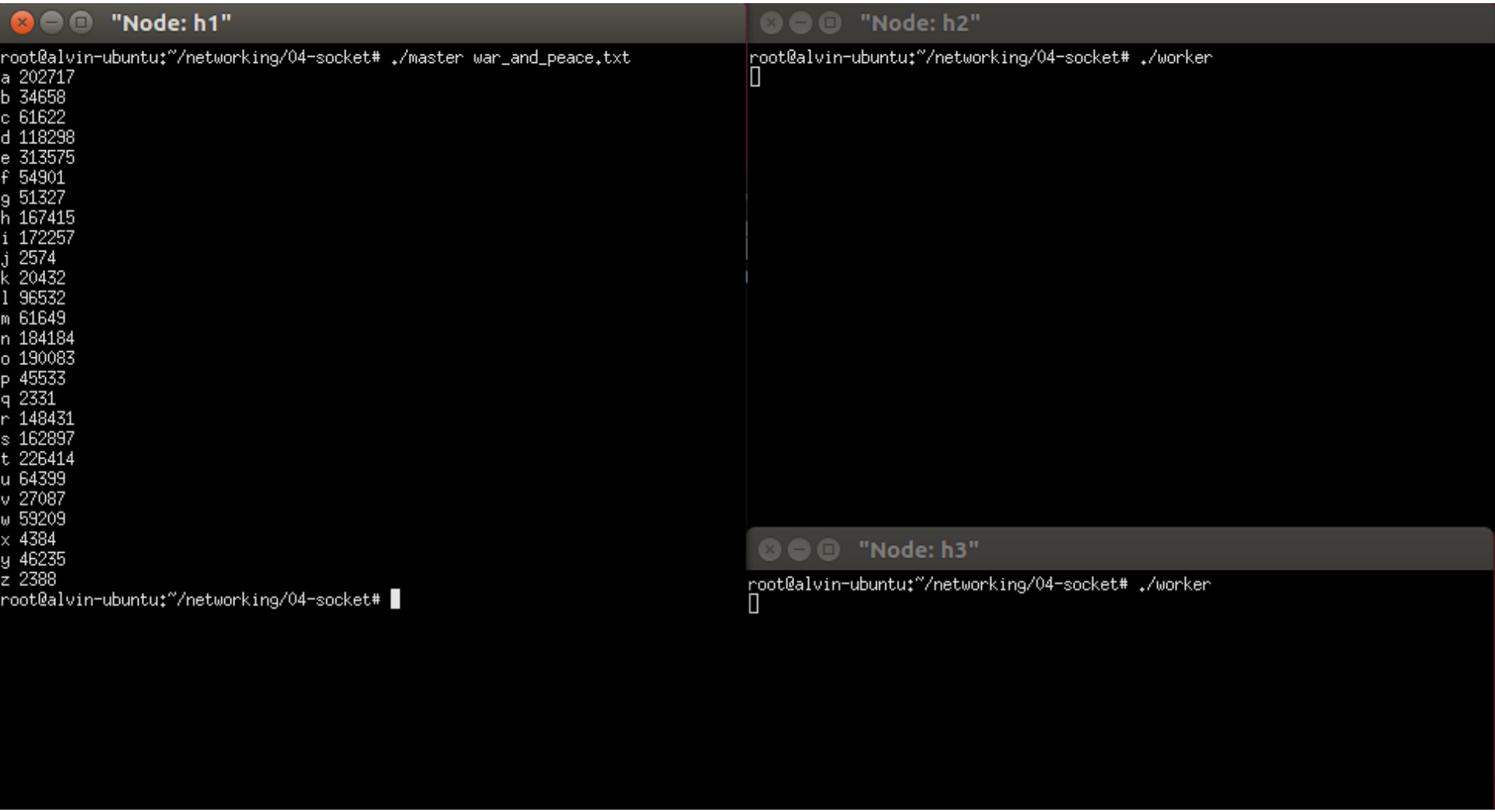
- Master 通过读取 workers.conf 配置文件，获取每个 worker 的 IP 地址，然后分别建立连接
- Master 获取 war_and_peace.txt 文件长度，将统计任务等分到所有的 worker
- Master 给每个 worker 发送消息，包括如下内容：
  - 消息总长度（4 个字节）
  - 文件所在位置（因为 master 和 worker 在同一主机同一目录，所以给出相对位置即可）
  - 需要进行字符统计的起始位置（4 个字节）和终止位置（4 个字节）

  注意：这里的 4 字节整数值应该用网络字节序传输

Worker 计算并返回结果

- 每个 worker 收到消息后，进行解析，根据指定统计区间对文件进行统计
  - 只统计 26 个英文字符的个数，大写字符转换成小写后再统计
- Worker 统计结束后，将每个字符出现的次数以 4 字节整数形式（网络字节序）返回给 Master ，因此传输消息长度为 104 字节
- Master 收到所有 worker 的消息后，进行聚合并输出到屏幕

结果形式


"Node: h1"
root@alvin-ubuntu:~/networking/04-socket# ./master war_and_peace.txt
a 202717
b 34658
c 61622
d 118298
e 313575
f 54901
g 51327
h 167415
i 172257
j 2574
k 20432
l 96532
m 61649
n 184184
o 190083
p 45533
q 2331
r 148431
s 162897
t 226414
u 64399
v 27087
w 59209
x 4384
y 46235
z 2388
root@alvin-ubuntu:~/networking/04-socket#
"Node: h2"
root@alvin-ubuntu:~/networking/04-socket# ./worker
"Node: h3"
root@alvin-ubuntu:~/networking/04-socket# ./worker

附件文件列表

- master-reference　　// master 程序参考
- topo.py　　// Mininet 拓扑文件
- war_and_peace.txt　　// 待统计文件
- worker-reference　　// worker 程序参考
- workers.conf　　// workers IP 地址配置文件
- example:　　// 完整的 socket 信息收发例子
  - client.c
  - Makefile
  - server.c