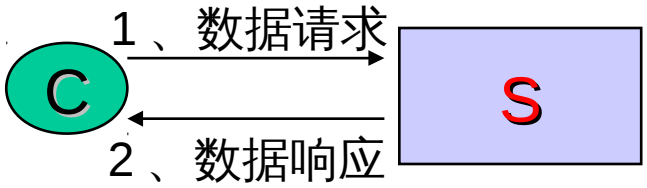
并发服务器（一）

2012-10

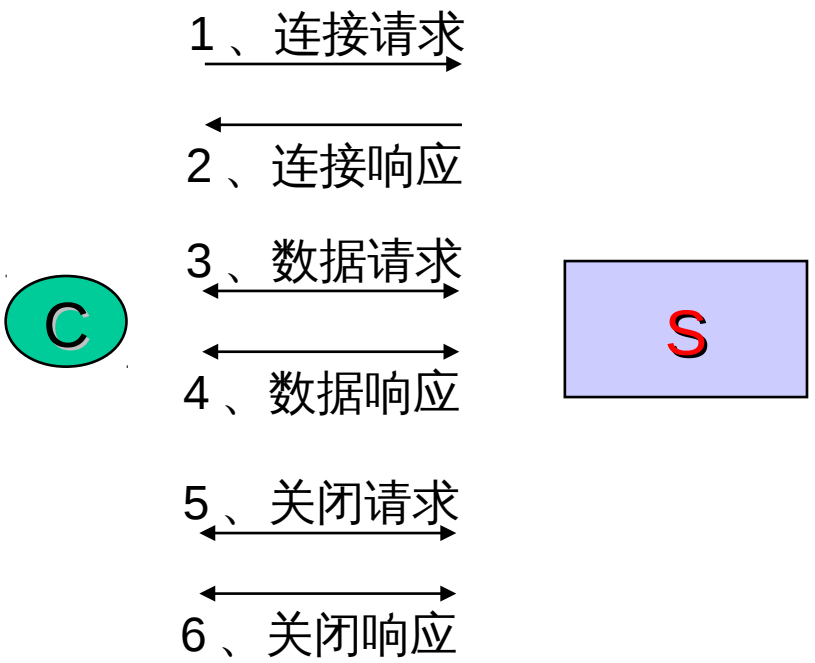
课程内容

- 服务器模型
- 多进程服务器模型

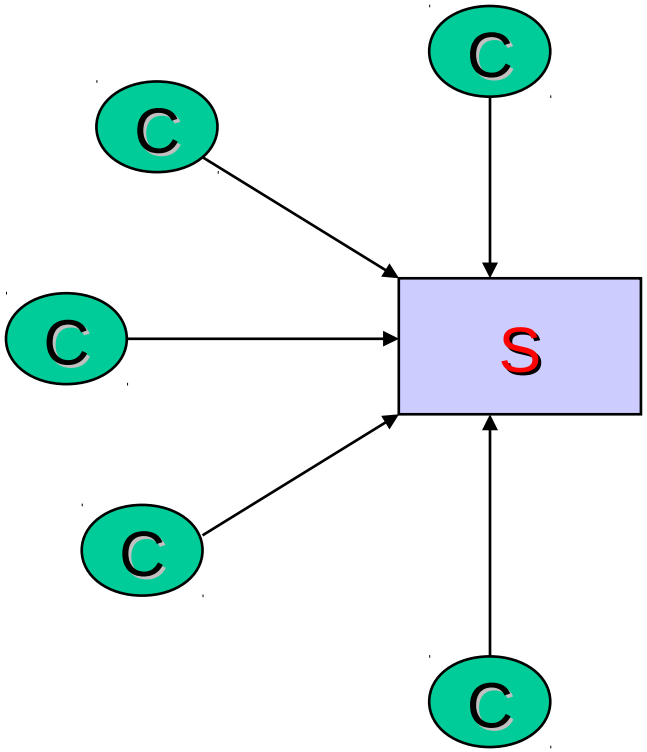
简单服务器模型



无连接



实际的客户服务器模型



服务器模型

- 现有服务器的模型主要有两种：
  - 循环（重复）服务器模型：每次只能处理一个客户的请求，但上一个客户的请求完成后，才能处理下一个客户的请求；
  - 并发服务器模型：服务器在同一时刻可以响应多个客户的请求；

UDP 重复模型

UDP 循环服务器的实现非常简单 :UDP 服务器每次从套接字上读取一个客户端的请求 , 处理 , 然后将结果返回给客户机 .

```
socket(...);
  bind(...);
  while(1)
   {
       recvfrom(...);
       process(...);
       sendto(...);
  }
```

TCP 重复服务器模型

```
socket(...);
 bind(...);
 listen(...);
 while(1)
 {
        accept(...);
        while(1)
        {
               read(...);
               process(...);
               write(...);
        }
        close(...);
 }
```

TCP 服务器接受一个客户端的连接 , 然后处理 , 完成了这个客户的所有请求后 , 断开连接 .

并发服务模型

在 linux 中提供了三种方式支持并发模型

- 多进程：
- 多线程
- I/O 多路复用

UDP 并发服务器

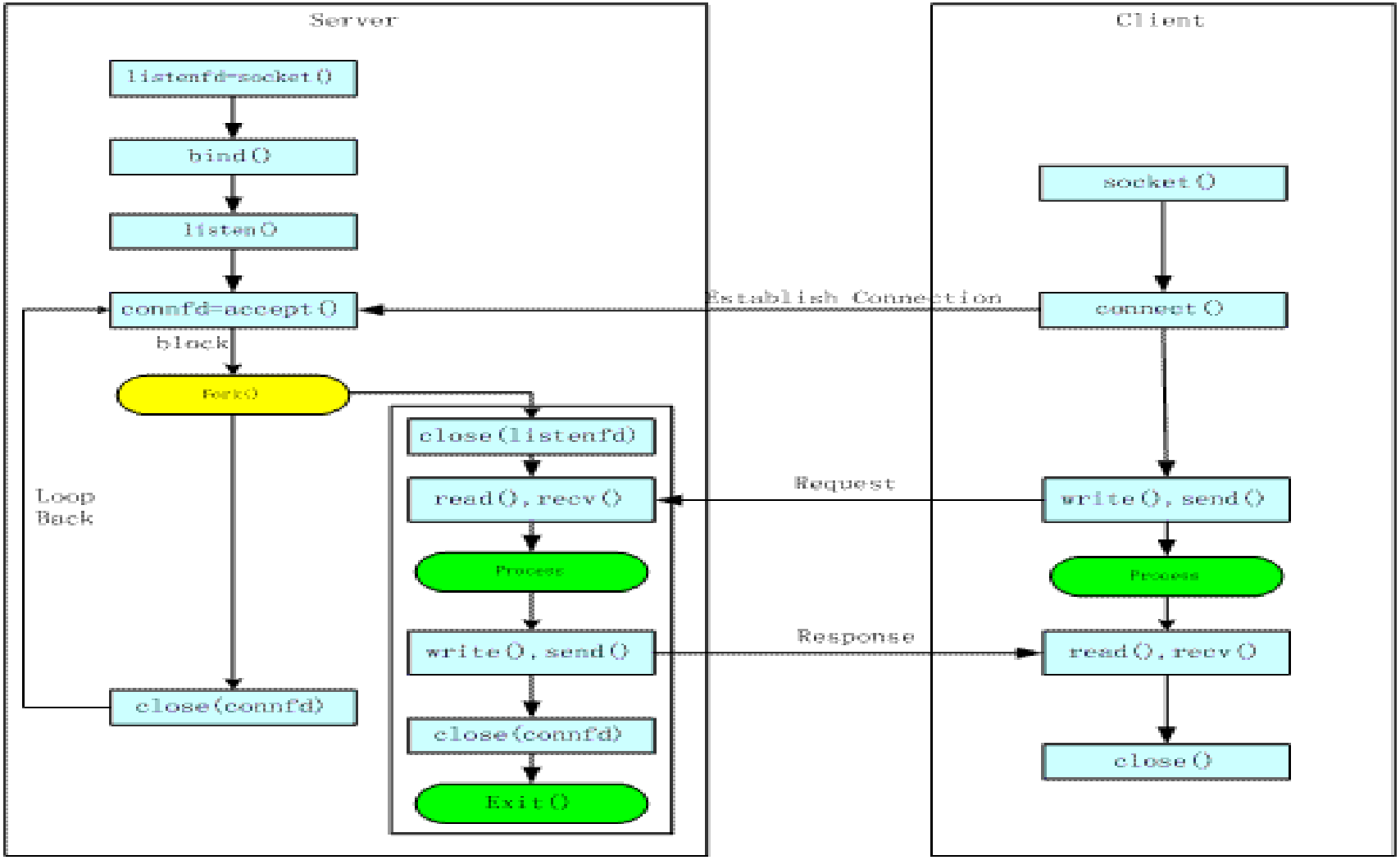
人们把并发的概念用于 UDP 就得到了并发 UDP 服务器模型 . 并发 UDP 服务器模型其实是简单的 . 和并发的 TCP 服务器模型一样是创建一个子进程来处理的 算法和并发的 TCP 模型一样 .

除非服务器在处理客户端的请求所用的时间比较长以外 , 人们实际上很少用这种模型 .

TCP 多进程并发服务器模型

```
socket(...);
bind(...);
listen(...);
 while(1)
{
        accept(...);
        if(fork(..)==0)
        {
              Close(listenfd)
              while(1)
               { read(...); process(...); write(...); } close(...); exit(...); }
               close(...); }
       Else if (fork()>0)
            {
                close(accept);
                continue;
            }
}
```

TCP 多进程并发模型


Server
listenfd=socket()
bind()
listen()
connfd=accept()
block
fork()
close(listenfd)
read(),recv()
Process
write(),send()
close(connfd)
Exit()
Loop
Back
close(connfd)
Client
socket()
connect()
Establish Connection
write(),send()
Request
Process
read(),recv()
Response
close()

多进程相关函数

```
#include <sys/types.h>
#include <unistd.h>

Pid_t  fork();
```

特点：一次调用，两次返回

子进程执行方式

- 与父进程执行完成相同的指令；
- 根据 fork 的返回值分别执行在同一段程序中执行不同的指令；
- 采用 exec 函数族执行覆盖数据段，代码段于堆栈段，执行一段新的代码；

Exec 函数族

分为四类：

l: 可变参数；

v: 使用数组作为参数；

p ：不需显示指定可执行文件的路径，根据 PATH 变量搜索文件；

e ：指定程序执行的环境变量

exit() 与 _exit()

- exit 推出程序时，关闭所有的文件描述符，同时清除标准输入出中的缓存，执行 at_exit() 注册的清除函数；
- _exit(), 关闭所有文件描述符，不清除标准输入出中的缓存，不执行 at_exit() 注册的清除函数；

Exec 函数族使用案例

```
char *exec_argv[4];
   exec_argv[0] = "telnet“;
   exec_argv[1] = ip;
   exec_argv[2] = port;
   exec_argv[3] = NULL;
   if (execv("/bin/telnet", exec_argv) == -1)
   {

      DoDisconnect();

      CheckError(nResult, etTelnetConnect, "Connect");

   }
```

僵尸程序

- 一个进程执行完后，与这个进程执行情况的信息会存储在进程表中，如果他的父进程不对这部分信息进行处理，这个部分信息会永远存储在进程表中；
- 我们将状态信息永远不会得到处理的子进程成为僵尸程序；

- 解决方法：
  - 在父进程中采用 wait();
  - 杀死父进程，使得 init 成为子进程的父进程；

进程同步

- pid_t wait(int *stat_loc);
- pid_t waitpid(pid_t pid,int *stat_loc,int options);

试验题目 1

- 自己编写程序实现远程控制系统中使用到函数 popen 功能；

思路

使用管道 pipe(int f_des[2]) 函数（参数 f_des[0] 用于读取管道，f_des[1] 用于向管道写入数据），通过管道实现父子进程间通讯；

步骤：

1. 创建管道；
2. 创建子进程；
3. 在父进程中：关闭 f_des[1], 使用 wait 操作与等待子进程，然后将管道中的数据读出打印显示；
4. 在子进程 : 关闭 f_des[0], 将管道 f_des[1] 与标准输出进行重定向（dup2(f_des[1],STDOUT_FILENO)），然后调用 execvp() 函数执行程序中接收到的命令；

试验题目 2

- 修改远程控制服务器代码，使得服务器同时能够向多个用户提供服务。