Lesson 19

1.如果想要服务器可以处理多个请求，一种简单的思路是采用 while。

```
while(1)
{
    fd = accept();
    do_service(fd);
}
```

这种叫做迭代服务器，不具有并发能力。

2.采用多进程编写并发服务器的大概思路：

```
while(1)
{
    peerfd = accept();
    if(fork() == 0)
    {
        close(listenfd);
        do_service(peerfd);
        exit(EXIT_SUCCESS);
    }
    close(peerfd); //这里防止资源耗尽
}
```

一些要点：

a) 父进程要关闭 peerfd，这主要是因为 close 根据引用计数关闭 fd，如果父进程不这样做，那么所有通过 accept 创建的 fd 都不会被真正释放。

b) 父进程不可以直接采用 waitpid 来回收子进程，这样做会使得 server 变为一个迭代服务器，而不具备并发能力。必须采用信号这种异步的处理手段。

c) 使用信号处理必须注意，使用 while 而不是 if，尽可能多处理僵尸进程，这是防止信号的阻塞和丢失问题。waitpid 要使用 WNOHANG 选项。

d) 子进程要关闭 listenfd。

e) 子进程执行 do_service 之后务必执行 exit(EXIT_SUCCESS).

3.采用多线程编写服务器程序的要点：

```
while(1)
{
    peerfd = accept();
    创建一个线程
}
```

要点：

a) 往线程中传 fd，最好使用动态内存分配。在线程中

务必释放内存，防止内存泄露

b) 线程务必使用 detach 函数，将自己设置为分离状态，这样不必再去调用 join 函数，等待线程结束。

4.使用 select 编写并发服务器的模型：

初始化参数

allset(作为 rset 的备份)/rset/clients/maxi(clients 中的最大下标)/maxfd(最大 fd，作为 select 的第一个参数)/nready

while(1)

{

1.初始化 rset. rset = allset

2.执行 select 等待

3.检查 listenfd 是否可读，如果可读

a) accept

b) 将 fd 加入 clients 以及 allset

c) 更新 maxi 和 maxfd

4.轮询 clients 中的客户 fd，查看他们是否在 rset 里面。

当对方 close 时，本地需要：

a) close 套接字

b) 更新 clients[i]，

c) 将 fd 从 allset 中移除

}

5.使用 poll 编写并发服务器的框架：

创建 struct pollfd 数组，存放用来监听的 fd

加入 listenfd

while(1)

{

1.poll 系统调用

2.检查 listenfd，如果可读，把 fd 加入到数组中

3.检查其他 fd，注意 fd 关闭的问题。

}

6.使用 epoll 编写并发服务器的框架：

1.创建 epoll 句柄

2.把 listenfd 加入 epoll

3.创建一个数组，用于接收 epoll_wait 的返回结果

while(1)

{

a) 执行 epoll_wait

b) 判断数组中的每个 fd，如果是 listenfd，那么需要 accept，如果是普通 fd，那么需要进行 echo 服务。

}

7.write 是把数据从用户空间拷贝至内核空间，而 read 是把数据从内核空间拷贝至用户空间。

8.select、poll、epoll 之间的区别：

a) select 文件描述符的大小受到限制，而且 FD_SETSIZE 受内核参数的限制，如果需要更改，需要重新编译内核。

b) poll 没有文件描述符大小的限制。

c) select 和 poll 共同的缺点是：内部的数组不停的在内核空间和用户空间中相互拷贝。而 epoll 采用共享内存，避免了这一开销。

d) select 和 poll 内部都是采用的“轮询”的机制，随着 fd 的增多，select 和 poll 的效率随之下降，而 epoll 只关心已经准备好的 fd，不存在这个缺点。