【OJ】TCP 多进程并发服务器与多进程客户端

©2020 张翔 | zhangx@uestc.edu.cn

1. 实验目的

- 深入理解多进程（Multiprocess）的相关基本概念，深入理解父子进程之间的关系与差异，熟练掌握基于 fork()的多进程编程模式；
- 充分理解僵尸进程的产生原理，能基于 sigaction()或 signal()、waitpid()或 wait()等函数进行程序设计，规避僵尸进程产生；
- 理解 Linux 文件系统的组织方式，掌握文件描述符的基本概念，深入理解当父进程使用 fork 之后，子进程对于父进程 fork 前创建的文件描述符的继承关系；
- 在实验“TCP 单进程循环服务器与单进程客户端”的基础上，进一步实践巩固：

  (1) Socket 网络编程系列核心系统调用；

  (2) 服务器对于客户端正常结束的识别处理；

  (3) 客户端基于命令行指令的退出实现方式；

  (4) 服务器基于 SIGITN 信号的退出实现方式（慢速系统调用退出问题）；

  (5) SIGPIPE 信号产生原因与识别处理方式；
- 同时，深入理解并熟练掌握 TCP 多进程并发服务器与多进程客户端的 Socket 编程模式，可在 Linux 原生环境下进行应用实践，包括：

  (1) 多进程并发服务器与多进程客户端的 Socket 编程核心系统调用模式；

  (2) 多进程应用程序如何有效规避僵尸进程的产生；

  (3) 简单应用层 PDU 的设计、构建、解析（数据）；

  (4) 文件的读写应用。

2. 实验要求

2.1. 整体概述

撰写面向 Echo 服务的 TCP 多进程并发服务器与多进程客户端代码。在 TCP 单进程 Socket 编程在线测评的基础上，本次测评整体新增要求概述如下：

(1) **fork 与多进程并发：**

- 服务器每 accept()一个建立连接的请求，则**必须**通过 fork()派生一个子进程与客户端进行业务对接。

(2) **客户端最大并发数：**

- 客户端**必须**增加一命令行参数，用于指定客户端并发的最大数量，通过客户端的多进程并发练习帮助构建并发连接测试。

(3) **PDU 构建解析与字节序：**

- 在读写边界定义实践的基础上定义了简单应用层 PDU；
- 由客户端进程序号 PIN（Process INdex，非 PID，自定义数值，从 0 开始按父子进程生成时序编号，且父进程总是为 0）、数据长度 LEN、数据 DATA 三部分构成；
- 客户端、服务器收发过程中**必须**严格遵循 PDU 定义，并做好字节序转换。

(4) **僵尸进程规避：**

- 服务器正常情况下子进程必然先于父进程结束，具备僵尸进程产生条件，**必须**通过安装 SIGCHLD 信号处理器进行处理或进行合理屏蔽；
- 客户端子进程与父进程并发工作，结束顺序存在不确定性，同样具有僵尸进程产生条件，**必须**通过安装 SIGCHLD 信号处理器进行处理或进行合理屏蔽。

(5) **文件读写应用：**

- 客户端、服务器在各自父子进程开始之时，均须各自创建 res（result）文本，用于记录收发数据与提示信息，以供测评使用。

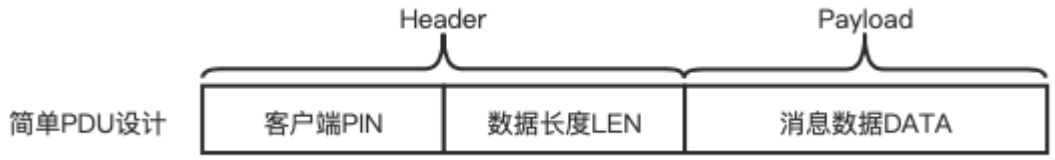
- 客户端所有进程**必须**以"stu_cli_res_*PIN*.txt"方式命名 res 文本，父进程的进程序号 PIN 始终定义为 0，子进程序号则根据创建顺序递增。
- 服务器主进程**必须**以"stu_srv_res_p.txt"命名 res 文本；服务器子进程首先**必须**以"stu_srv_res_*PID*.txt"命名 res 文本，待通过客户端数据获取客户端进程序号 PIN 只后，则**必须**于业务处理结束之时（进程终结之前），将"stu_srv_res_*PID*.txt"重命名为"stu_srv_res_ *PIN*.txt"。

■ 客户端父子进程还**必须**根据各自序号 PIN，在业务逻辑开始时对应读取测试数据文件 td*PIN*.txt。

- 如父进程对应读取测试数据 td0.txt，而 PIN 值为 3 的子进程，则对应读取测试数据 td3.txt。

2.2. PDU 设计

- **Header：客户端 PIN（int） + 数据长度 LEN（int，仅 Payload 部分）**
- 构建 PDU 时：
  - 可采用单一缓存（长度大于 LEN+8）；
  - 先行写入 DATA，根据 DATA 获取 LEN，且 LEN **必须**转换为网络字节序后方可写入缓存；
  - 随后记录客户端进程序号 PIN，**必须**转换为网络字节序后方可写入缓存；
  - PDU 构建好之后，即可一次性发送。



2.3. 客户端程序

(1) **必须**通过命令行参数指定服务器端地址（IP、Port），同时增加一个参数用于指定客户端多进程**最大并发数（含父进程）**：

```
# CMD、IP、Port、最大并发数之间采用空格分隔
./tcp_echo_cli.o 127.0.0.1 30000 3 // 3 即最大并发数（含父进程）
```

(2) 父进程**必须**合理调用 fork()，按需生成子进程，不多也不少。

(3) 父进程与所有子进程**必须**在启动时打开**"stu_cli_res_*PIN*.txt"**（父进程的进程序号 PIN 始终定义为 0，子进程序号则根据创建顺序递增）。

- 同时应该以如下形式打印提示信息至 stdout：

```
[cli](8688) stu_cli_res_2.txt is created!
// 8688 为当前进程 PID，“stu_cli_res_2.txt”中 2 为当前进程序号 PIN
```

- 同时应该以如下形式打印提示信息到 res 文件：

```
[cli](8688) child process 2 is created! //2 为当前进程序号 PIN
```

(4) 父进程和所有子进程各自调用 socket()分配 connfd，随后分别连接服务器。成功连接服务器后，**必须**将服务器端地址信息以如下形式**打印输出至对应的 "stu_cli_res_*PIN*.txt"**：

```
[cli](8688) server[127.0.0.1:30000] is connected!
```

(5) 随后父进程和所有子进程应该各自执行业务处理函数 echo_rqt()，**调用时传入自己的进程序号 PIN**，故 echo_rqt()通常可做如下申明：

```
int echo_rqt(int sockfd, int pin);
```

(6) echo_rqt()中，父进程和所有子进程**必须**各自根据自己的 PIN 来读取对应的测试数据文件 **td*PIN*.txt**。如父进程对应读取测试数据为 **td0.txt**，而 PIN 为 3 的子进程则对应读取测试数据 **td3.txt**。其中：

- 每次读入 td 文件的**一行数据**，正确构建 PDU 之后（参见 2.1）即可发送，随后接收解析服务器回传数据并打印输出，其中：
  - **必须**按照如下方式采用 strncmp()函数进行指令"exit"匹配检测，确认匹配后，**必须**立刻释放当前进程相关资源并退出进程：

```
if(strncmp(buf, "exit", 4) == 0)
    return 0;
```

  - **必须**将读入数据尾部的"\n"更换为"\0"（若一行数据只包含"\n"，则向服务器发送"\0"），以便于服务器按照 string 方式处理数据。
  - 接收服务器镜像数据，且**必须**以如下形式打印输出至对应的 **"stu_**

cli_res_*PIN*.txt"（即每行原始数据前增加"[echo_rep](*PID*) "作为头部）：

```
[echo_rep](8868) Hi there, a wonderful day :)
[echo_rep](8868) A blank line will be followed... - -
[echo_rep](8868)
[echo_rep](8868) Weired? Not at all actually!@#$%^*
```

(7) 命令行字符最大长度**必须**限制在 100 以内

```
#define MAX_CMD_STR 100
```

重要提示：buf 长度在 MAX_CMD_STR+1 的基础上还需考虑 PDU 头部数据。

(8) 客户端成功释放用于连接服务器的 connfd 后（进程结束之前），应该以如下形式打印提示信息至"stu_cli_res_*PIN*.txt"：

```
[cli](8868) connfd is closed!
[cli](8868) child process is going to exit!//或 parent process
```

(9) 父进程与所有子进程**必须**在退出前关闭"stu_cli_res_*PIN*.txt"。

- 同时应该以如下形式打印提示信息至 stdout

```
[cli](8868) stu_cli_res_2.txt is closed! //2 为当前进程 PIN
```

2.4. 服务器程序

(1) **必须**通过命令行参数指定服务器所要绑定的 IP 与 Port：

```
#IP 127.0.0.1; PORT 30000; CMD、IP、Port 间采用空格分隔
./tcp_echo_srv.o 127.0.0.1 30000
```

(2) 识别处理 SIGPIPE 信号，避免服务器程序因管道破裂（broken pipe）而退出运行，**必须**申明信号处理函数为：

```
void sig_pipe(int signo);
```

(3) 识别处理 SIGINT 信号，**必须**使用 sigaction()安装信息处理器，以支持设置慢调用退出，同时**必须**申明信号处理函数为：

```
void sig_int(int signo);
```

(4) 识别处理 SIGCHLD 信号，避免因为子进程先于父进程结束而产生僵尸进程，**必须**申明信号处理函数为**系统宏 SIG_IGN** 或以下函数：

```
void sig_chld(int signo){
    pid_t pid_chld;
```

```
    while ((pid_chld = waitpid(-1, &stat, WNOHANG)) > 0);
}
```

同时，配置 sigaction 结构体时，sa_flags **必须**设置为 SA_RESTART，以确保受 SIGCHLD 触发影响的慢调用函数立刻重启，而非返回异常。

(5) 在以上三类信号处理函数中：

a. 应该通过设置全局变量 sig_type 接收传入的 signo，用于后续系统中断处理逻辑；

b. 应该在处理函数中，通过 getpid()获取进程 ID，以便标识输出记录，有助于观测程序运行过程以及开展调试：

```
[srv](20369) SIGPIPE is coming!
//20369 为进程 PID 值（后续规范一致，[srv]后紧跟(PID)，不再重复叙述）
```

c. 面向 SIGCHLD，若安装信号处理函数 sig_chld()，则应该在信号处理函数 sig_chld()中，尝试在获取终结的子进程 ID 后，按照以下格式进行提示：

```
[srv](20369) server child(20372) terminated.
//20369 为父进程（信号处理函数执行进程）PID， 20372 为子进程 PID
```

(6) 父进程**必须**在启动时打开**"stu_srv_res_p.txt"**

- 同时应该以如下形式打印提示信息至 stdout：

```
[srv](20369) stu_srv_res_p.txt is opened!
```

(7) 父进程调用 bind()前，**必须**将命令行参数指定的服务器 IP 与 Port 信息以如下形式打印输出至**"stu_srv_res_p.txt"**：

```
[srv](20369) server[127.0.0.1:30000] is initializing!
```

(8) 父进程**必须**在调用 accept()获取到新的已建立连接的客户端请求时，将客户端端地址信息以如下形式打印输出至**"stu_srv_res_p.txt"**：

```
[srv](20369) client[127.0.0.1:42796] is accepted!
```

(9) accept()返回用于处理客户端业务的 socket 描述符 connfd 后，**必须**合理调用 fork()派生子进程，用于执行与客户端收发（读写）互动的业务处理：

- 子进程**必须**在启动时打开**"stu_srv_res_*PID*.txt"**，可以用如下方式进行命名指定：

```
char fn_res[20];
```

```
sprintf(fn_res, "stu_srv_res_%d.txt", pid);
fp_res = fopen(fn_res, "wb");
```

同时应该以如下形式打印提示信息至 stdout：

```
[srv](20372) stu_srv_res_20372.txt is opened!
```

同时应该以如下形式打印提示信息到 res 文件：

```
[cli](20372) child process is created!
```

- 子进程应该各自执行业务处理函数 echo_rep()，且**必须**返回从客户端数据中获取的客户端进程序号 PIN，故 echo_rep()通常做如下申明：

```
int echo_rep(int sockfd)
// 必须返回客户端进程序号 PIN，以便进程退出前修改 res 文本名称
```

- 子进程**必须**在收到客户端 echo_req 数据之后，以如下形式打印输出至 `res` 文件，并将客户端原始数据镜像回送：

```
[echo_rqt](20372) Hi there, a wonderful day :)
[echo_rqt](20372) A blank line will be followed... - -
[echo_rqt](20372)
[echo_rqt](20372) Weired? Not at all actually!@#$%^*
```

客户端原始输入数据为：

```
Hi there, a wonderful day :)
A blank line will be followed...

Weired? - - Not at all actually!@#$%^*
```

- 子进程**必须**能够识别处理客户端的正常退出（参见实验 1）
- 子进程**必须**在退出前更名**"stu_srv_res_*PID*.txt"**为**"stu_srv_res_*PIN*.txt"**
  - 同时应该以如下形式打印提示信息至 res 文件：

```
[srv](20372) res file rename done!
```

- 子进程**必须**在退出前关闭 connfd
  - 同时应该以如下形式打印提示信息至 res 文件：

```
[srv](20372) connfd is closed!
```

- 子进程**必须**在退出前关闭**"stu_srv_res_*PID*.txt"**
  - 在关闭文件之前，应该以如下形式打印提示信息至 res 文件：

```
[srv](20372) child process is going to exit!
```

○ 同时应该以如下形式打印提示信息至 stdout：

[srv](20372) stu_cli_res_20372.txt is closed

(10) 父进程 accept()挂起等待期间 若收到 SIGINT 信号 父进程**必须**终结运行，即 accept()不能重新启动而是**必须**退出。此时 accept 返回-1 且对应 errno 为 EINTR，服务器识别后须即刻释放 listenfd 等必须释放的资源（参见实验 1），

- 同时应该以如下形式打印提示信息至**"stu_srv_res_p.txt"**。

[srv](20369) listenfd is closed!
[srv](20369) parent process is going to exit!

(11) 父进程**必须**在退出前关闭**"stu_srv_res_p.txt"**，同时应该以如下形式打印提示信息至 stdout：

[srv](20369) stu_srv_res_p.txt is closed!

3. 实验原理

参见课程讲义。

4. 输入输出

4.1. 输入规范

- **参见上述 2. 实验要求**

4.2. 输入范例

./tcp_echo_srv.o 127.0.0.1 30000 # 服务器
./tcp_echo_cli.o 127.0.0.1 30000 3 # 客户端，含父进程最大 3 个并发

- **备注：客户端启动父进程与全部子进程根据进程序号 PIN 对应读取测试数据 td0.txt ~ td*(PIN-1)*.txt，本范例中即 td0.txt, td1.txt 与 td2.txt，测试数据由测试系统自动生成，范例如下（td1.txt）：**

39Z7 6ryKiZ 1 l2zzJM
WBKzp1r O EcE9v
l0E hU0eI TmYeH2V Wetfa s6O01f4

ot6mas
pugzplX aul3w n SAGJ2UD
Ae77H1YZ qnAA7e wi G o vBj

exit

4.3. 输出规范

- **参见上述 2. 实验要求**

4.4. 输出范例

若客户端指定最大并发数为 3，则服务器输出 4 个文件，客户端输出 3 个文件：

- **服务器**

服务器输出文件如下：

stu_srv_res_p.txt
stu_srv_res_0.txt
stu_srv_res_1.txt
stu_srv_res_2.txt

文件 stu_srv_res_p.txt 范例如下：

[srv](9960) server[127.0.0.1:30000] is initializing!
[srv](9960) client[127.0.0.1:40236] is accepted!
[srv](9960) client[127.0.0.1:40238] is accepted!
[srv](9960) client[127.0.0.1:40240] is accepted!
[srv](9960) SIGCHLD is coming!
[srv](9960) SIGCHLD is coming!
[srv](9960) SIGCHLD is coming!
[srv](9960) SIGINT is coming!
[srv](9960) listenfd is closed!
[srv](9960) parent process is going to exit!

文件 stu_srv_res_*PIN*.txt 范例如下（PIN 取 1，即 stu_srv_res_1.txt）：

[srv](9971) child process is created!
[srv](9971) listenfd is closed!
[echo_rqt](9971) 39Z7 6ryKiZ 1 l2zzJM
[echo_rqt](9971) WBKzp1r O EcE9v
[echo_rqt](9971) l0E hU0eI TmYeH2V Wetfa s6O01f4
[echo_rqt](9971) ot6mas
[echo_rqt](9971) pugzplX aul3w n SAGJ2UD

```
[echo_rqt](9971) Ae77H1YZ qnAA7e wi G o vBj
[echo_rqt](9971)
[srv](9971) res file rename done!
[srv](9971) connfd is closed!
[srv](9971) child process is going to exit!
```

- **客户端**

客户端输出文件如下：

```
stu_cli_res_0.txt
stu_cli_res_1.txt
stu_cli_res_2.txt
```

文件 stu_cli_res_*PIN*.txt 范例如下（PIN 取 1，即 stu_cli_res_1.txt）：

```
[cli](9967) child process 1 is created!
[cli](9967) server[127.0.0.1:30000] is connected!
[echo_rep](9967) 39Z7 6ryKiZ 1 l2zzJM
[echo_rep](9967) WBKzp1r O EcE9v
[echo_rep](9967) l0E hU0eI TmYeH2V Wetfa s6O01f4
[echo_rep](9967) ot6mas
[echo_rep](9967) pugzplX aul3w n SAGJ2UD
[echo_rep](9967) Ae77H1YZ qnAA7e wi G o vBj
[echo_rep](9967)
[cli](9967) connfd is closed!
[cli](9967) child process is going to exit!
```

5. 命名打包

- **TCP ECHO 客户端：`tcp_echo_cli.c`**
- **TCP ECHO 服务器：`tcp_echo_srv.c`**
- **请将 `tcp_echo_cli.c` 与 `tcp_echo_srv.c` 直接打包为 `src.tar` 文件（不能带目录）**

```
tar cvf src.tar tcp_echo_cli.c tcp_echo_srv.c
```

6. 重要提示

6.1. TCP 多进程并发服务器与多进程客户端代码框架模版

代码模版仅供参考，并不要求严格按模版框架实现，只要能满足测评要求即可。

代码框架模版：https://docs.qq.com/doc/DREhvZmpMVHRQRXFO

特别提示：文中但凡标注为“必须”的任务，必须严格按照要求执行（考核依据，涉及到输出要求规范时请特别注意）；而其他标注为“应该”的任务，目的是希望帮助同学们理解程序、协议执行过程，帮助进行开发调试，是否执行自行考量。

6.2. 但凡需要向 res 文件写入的地方，强烈建议同时向 stdout 输出，以便观察调试

送大家一个宏，方便同时向 stdout 与指定文件输出（fp 为空，等同于 printf）：

```
#define bprintf(fp, format, ...) \
    if(fp == NULL){printf(format, ##__VA_ARGS__);}    \
    else{printf(format, ##__VA_ARGS__); \
            fprintf(fp,format,##__VA_ARGS__);fflush(fp);}
```