本地套接字编程（socket IPC，稳定性好）

1、基础知识

1）socket API 原本是为网络通讯设计的，但后来在 socket 的框架上发展出一种 IPC 机制，就是 UNIX Domain Socket。虽然网络 socket 也可用于同一台主机的进程间通讯（通过 loopback 地址 127.0.0.1），但是 UNIX Domain Socket 用于 IPC 更有效率：不需要经过网络协议栈，不需要打包拆包、计算校验和、维护序号和应答等，只是将应用层数据从一个进程拷贝到另一个进程。这是因为，IPC 机制本质上是可靠的通讯，而网络协议是为不可靠的通讯设计的。面向消息的 UNIX Domain Socket 是可靠的，消息既不会丢失也不会顺序错乱。
2）UNIX Domain Socket 是全双工的，API 接口语义丰富，相比其它 IPC 机制有明显的优越性，目前已成为使用最广泛的 IPC 机制，比如 X Window 服务器和 GUI 程序之间就是通过 UNIXDomain Socket 通讯的。

2、编程要点

1）使用 UNIX Domain Socket 的过程和网络 socket 十分相似，也要先调用 socket()创建一个 socket 文件描述符，address family 指定为 AF_UNIX，type 可以选择 SOCK_DGRAM 或 SOCK_STREAM，protocol 参数仍然指定为 0 即可。
2）UNIX Domain Socket 与网络 socket 编程最明显的不同在于地址格式不同，用结构体 sockaddr_un 表示，网络编程的 socket 地址是 IP 地址加端口号，而 UNIX Domain Socket 的地址是一个 socket 类型的文件在文件系统中的路径，这个 socket 文件由 bind()调用创建，如果调用 bind()时该文件已存在，则 bind()错误返回。

```
struct sockaddr_un {
__kernel_sa_family_t sun_family;    //地址结构类型
char sun_path[UNIX_PATH_MAX];    //socket 文件名(含路径)
};
```

3）在需要传入 sockaddr_un 结构体长度的函数（bind()和 connect()）中，不应该使用 sizeof(struct sockaddr_un)，而应该求出“地址结构类型长度+ socket 文件名的真实长度”，具体求法如下：

```
len = offsetof(struct sockaddr_un, sun_path) + strlen(name);
```

注：#define offsetof(type, member) ((int)&((type *)0)->MEMBER)
offsetof(struct sockaddr_un, sun_path) 求出在结构体中 sun_path 变量的偏移位置，即 sun_family 变量占用空间大小。

3、具体实现

1）server：

```
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <strings.h>
#include <string.h>
```

```
#include <ctype.h>
#include <arpa/inet.h>
#include <sys/un.h>
#include <stddef.h>

#define SERV_ADDR    "serv.socket"

int main(void)
{
    int lfd, cfd, len, size, i;
    struct sockaddr_un servaddr, cliaddr;
    char buf[4096];

    lfd = socket(AF_UNIX, SOCK_STREAM, 0);

    bzero(&servaddr, sizeof(servaddr));
    servaddr.sun_family = AF_UNIX;
    strcpy(servaddr.sun_path,SERV_ADDR);

    len = offsetof(struct sockaddr_un, sun_path) + strlen(servaddr.sun_path);
    unlink(SERV_ADDR);
    bind(lfd, (struct sockaddr *)&servaddr, len); /*  参 3 不能是 sizeof(servaddr) */

    listen(lfd, 20);

    printf("Accept ...\n");
    while (1) {
        len = sizeof(cliaddr);
        cfd = accept(lfd, (struct sockaddr *)&cliaddr, (socklen_t *)&len);

        len -= offsetof(struct sockaddr_un, sun_path); /*  得到文件名的长度  */
        cliaddr.sun_path[len] = '\0'; /*  确保打印时,没有乱码出现  */

        printf("client bind filename %s\n", cliaddr.sun_path);

        while ((size = read(cfd, buf, sizeof(buf))) > 0) {
            for (i = 0; i < size; i++)
                buf[i] = toupper(buf[i]);
            write(cfd, buf, size);
        }
        close(cfd);
    }
    close(lfd);
    return 0;
```

```
}
```

2）client：

```
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <strings.h>
#include <string.h>
#include <ctype.h>
#include <arpa/inet.h>
#include <sys/un.h>
#include <stddef.h>

#define SERV_ADDR "serv.socket"
#define CLIE_ADDR "clie.socket"

int main(void)
{
	int    cfd, len;
	struct sockaddr_un servaddr, cliaddr;
	char buf[4096];

	cfd = socket(AF_UNIX, SOCK_STREAM, 0);

	bzero(&cliaddr, sizeof(cliaddr));
	cliaddr.sun_family = AF_UNIX;
	strcpy(cliaddr.sun_path,CLIE_ADDR);

	len = offsetof(struct sockaddr_un, sun_path) + strlen(cliaddr.sun_path);
	unlink(CLIE_ADDR);
	bind(cfd, (struct sockaddr *)&cliaddr, len); /*  客户端也需要 bind */


	bzero(&servaddr, sizeof(servaddr)); /*  构造 server  地址  */
	servaddr.sun_family = AF_UNIX;
	strcpy(servaddr.sun_path,SERV_ADDR);

	len = offsetof(struct sockaddr_un, sun_path) + strlen(servaddr.sun_path);
	connect(cfd, (struct sockaddr *)&servaddr, len);

	while (fgets(buf, sizeof(buf), stdin) != NULL) {
		write(cfd, buf, strlen(buf));
		len = read(cfd, buf, sizeof(buf));
		write(STDOUT_FILENO, buf, len);
```

```
        }
        close(cfd);
        return 0;
}
```