Сокеты-2

Работа сервера

- Стандартная схема работы плоха тем, что одновременно обслуживается только один клиент! Это приводит к задержкам в работе сети. Во-вторых, здесь используются ***блокирующие*** сокеты, т.е. работа приложения блокируется на время выполнения команд send, recv, accept…

Работа сервера

```
while(1) {
    sock = accept(listener, NULL, NULL);
    …проверка…
    switch (fork()) {
        case -1: ошибка
        case 0: обработка соединения sock
        default: close(sock);
    }
}
```

Работа сервера

- Вместо создания копии процесса можно создавать новый поток. (см. «Операционные системы»)
- Возможная проблема – общение между клиентами.

Неблокирующие сокеты

```
#include <unistd.h>
#include <fcntl.h>

sockfd = socket(AF_INET, SOCK_STREAM, 0);
fcntl(sockfd, F_SETFL, O_NONBLOCK);
```

Такая операция превращает сокет в неблокирующий.

Работа сервера

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

int select(int n, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, struct timeval *timeout);
```

N – максимальное значение дескриптора в множестве + 1
struct timeval { int tv_sec; // секунды
int tv_usec; // микросекунды };

FD_ZERO(int fd, fd_set *set); - очищает множество **set**
FD_SET(int fd, fd_set *set); - добавляет дескриптор **fd** в множество **set**
FD_CLR(int fd, fd_set *set); - удаляет дескриптор **fd** из множества **set**
FD_ISSET(int fd); - проверяет, содержится ли дескриптор **fd** в множестве **set**

Select

Если хотя бы один сокет готов к выполнению заданной операции, **select** возвращает ненулевое значение, а все дескрипторы, которые привели к "срабатыванию" функции, записываются в соответствующие множества.

Работа Select

```
if(select(mx+1, &readset, NULL, NULL, &timeout) <= 0)
    … ошибка …

// Определяем тип события и выполняем соответствующие действия
if(FD_ISSET(listener, &readset))
{
        // Поступил новый запрос на соединение, используем accept
        int sock = accept(listener, NULL, NULL);
          ……..
}

 for(… по всем клиентским сокетам …)
    if(FD_ISSET(сокет, &readset))
          // Поступили данные от клиента, читаем их и если надо
    отправляем
```

Полезные функции

#include <netdb.h>

struct hostent *gethostbyname(const char *name); - Для преобразования доменного имени в IP-адрес

gethostbyaddr - наоборот

#include <unistd.h>

int gethostname(char *hostname, size_t size); - получить имя локального хоста

ICMP

```
struct iphdr* ip;
struct iphdr* ip_reply;
struct icmphdr* icmp;
struct sockaddr_in connection;
char* packet;
char* buffer;

ip = malloc(sizeof(struct iphdr));
ip_reply = malloc(sizeof(struct iphdr));
icmp = malloc(sizeof(struct icmphdr));
packet = malloc(sizeof(struct iphdr) + sizeof(struct icmphdr));
buffer = malloc(sizeof(struct iphdr) + sizeof(struct icmphdr));
ip = (struct iphdr*) packet;
icmp = (struct icmphdr*) (packet + sizeof(struct iphdr));
```

```
ip->ihl = 5;
ip->version = 4;
ip->tos = 0;
ip->tot_len = sizeof(struct iphdr) + sizeof(struct
icmphdr);
ip->id = htons(random());
ip->ttl = 255;
ip->protocol = IPPROTO_ICMP;
ip->saddr = inet_addr(src_addr);
ip->daddr = inet_addr(dst_addr);
```

```
sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);

    /*
    *  IP_HDRINCL must be set on the socket so that
    *  the kernel does not attempt to automatically add
    *  a default ip header to the packet
    */
setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, &optval,
    sizeof(int));
```

```
icmp->type          = ICMP_ECHO;
icmp->code           = 0;
icmp->un.echo.id     = 0;
icmp->un.echo.sequence   = 0;
icmp->checksum       = 0;

icmp-> checksum      = in_cksum((unsigned short *)icmp,
 sizeof(struct icmphdr));
ip->check            = in_cksum((unsigned short *)ip,
 sizeof(struct iphdr));

sendto(sockfd, packet, ip->tot_len, 0,
 (struct sockaddr *)&connection, sizeof(struct sockaddr));
```

```
unsigned short in_cksum(unsigned short *addr, int len)
{
    register int sum = 0; u_short answer = 0; register u_short *w = addr; register int nleft = len;

    while (nleft > 1)
    {
      sum += *w++;
      nleft -= 2;
    }
    if (nleft == 1)
    {
      *(u_char *) (&answer) = *(u_char *) w;
      sum += answer;
    }
    sum = (sum >> 16) + (sum & 0xffff);       /* add hi 16 to low 16 */
    sum += (sum >> 16);               /* add carry */
    answer = ~sum;              /* truncate to 16 bits */
    return (answer);
}
```