Системное программное обеспечение локальных компьютерных сетей

Модель сетевого взаимодействия клиент-сервер

Денис Пынькин

2011 – 2012

e-mail: denis.pynkin@bsuir.by

http://goo.gl/32cTB

СЧАСТЬЕ ДЛЯ ВСЕХ, ДАРОМ, И ПУСТЬ НИКТО НЕ УЙДЕТ ОБИЖЕННЫЙ!

(c)Стругацкие, Пикник на обочине

Алгоритмы и задачи проектирования серверного программного обеспечения

Концептуальный алгоритм

По сути каждый сервер функционирует по следующему алгоритму:
в нем создается сокет и выполняется привязка сокета к порту. Затем сервер входит в бесконечный цикл, в котором он принимает очередной запрос, поступающий от клиента, обрабатывает этот запрос, формирует ответ и отправляет его клиенту.

Время обработки запроса сервером – это общее количество времени, которое требуется серверу для обработки одного отдельно взятого запроса
Время отклика – задержка между временем отправки клиентом запроса и временем получения ответа от сервера.

Время отклика всегда больше времени обработки запроса, однако при применении очереди запросов, подлежащих обработке, отклик может занимать значительно больше времени нежели время обработки запроса.

Классификация серверов

По внутренней архитектуре

- Последовательный
- Параллельный

Классификация серверов

По внутренней архитектуре

- Последовательный
- Параллельный

По типу используемого сервиса

- С установлением соединения
- Без установления соединения

Классификация серверов

По внутренней архитектуре

- Последовательный
- Параллельный

По типу используемого сервиса

- С установлением соединения
- Без установления соединения

По состоянию

- С сохранением состояния (stateful)
- Без сохранения состояния (stateless)

Варианты параллелизма

Многопоточное решение применяется, если затраты на создание и переключение между потоками невелики и при этом требуется совместное использование или обмен данными между соединениями.

Варианты параллелизма

Многопоточное решение применяется, если затраты на создание и переключение между потоками невелики и при этом требуется совместное использование или обмен данными между соединениями.
Многопроцессная модель применяется для достижения максимального распараллеливания. Если используются процессы, то появляется возможность использовать для обработки внешние программы.

Варианты параллелизма

Многопоточное решение применяется, если затраты на создание и переключение между потоками невелики и при этом требуется совместное использование или обмен данными между соединениями.
Многопроцессная модель применяется для достижения максимального распараллеливания. Если используются процессы, то появляется возможность использовать для обработки внешние программы.
При использовании *асинхронного ввода/вывода* обработка запросов ведется только в одном потоке, поэтому сервер имеет практически такую же производительность, что и последовательный, даже на компьютере с несколькими процессорами. Удобно применять, если сервер должен иметь доступ к данным разных соединений или на обработку каждого запроса не требуется много времени.

Серверы с установлением логического соединения

Преимущества:

- простота программирования

Недостатки:

- для каждого логического соединения требуется создавать отдельный сокет
- требуется трехэтапное квитирование при установке и разрыве соединения, что невыгодно для использования передачи небольших объемов данных в небольшой сети
- простаивающее соединение, по которому не проходят пакеты, напрасно используют ресурсы

Серверы без установления логического соединения

Преимущества:

- нет накладных расходов на установление и разрыв соединения
- можно реализовывать широковещательные или групповые рассылки
- не требуются ресурсы на поддержание соединения

Недостатки:

- самостоятельная реализация механизмов управления передачей: квитирование, тайм-ауты, оптимизация трафика, контроль надежности

stateful & stateless серверы

Информация о состоянии – это обновляемая сервером информация о ходе взаимодействия с клиентом.
Информация о состоянии применяется для эффективной оптимизации сервера.
Если на сервере сохраняются какие-либо данные о запросах клиента, то:

- можно значительно сократить объем передаваемой информации и ускорить работу сервера
- информация о состоянии также может сохраняться для использования даже в случае перезагрузки сервера.

stateful & stateless серверы

Отрицательная сторона: информация о состоянии, хранящаяся на сервере, может стать ошибочной, если сообщения были потеряны, продублированы или доставлены не в исходном порядке, либо если клиент аварийно перезапустился. Соответственно и ответ сервера, основанный на ошибочной информации может быть ошибочным.

Основные типы сервера

- Последовательный сервер без установления логического соединения
- Последовательный сервер с установлением логического соединения
- Параллельный сервер без установления логического соединения
- Параллельный сервер с установлением логического соединения

Последовательный сервер без установления логического соединения

Используется в службах, требующих незначительного времени для обработки каждого запроса

Последовательный сервер с установлением логического соединения

Используется в службах, требующих незначительного времени для обработки каждого запроса, однако требующих надежного протокола доставки сообщений. За счет больших издержек на установку и завершение соединения среднее время отклика часто значительно выше, чем у предыдущего сервера.

Параллельный сервер без установления логического соединения

Редко применяемый тип сервера.
Во многих случаях затраты на создание потоков или процессов не оправдывают повышения эффективности, достигнутого за счет параллелизма.

Параллельный сервер с установлением логического соединения

Наиболее распространенный тип сервера, поскольку сочетает надежный протокол (подходит и для глобальных сетей) с возможностью одновременной работы с несколькими клиентами.

Алгоритмы серверов

Последовательный сервер с установлением логического соединения

1. Создать сокет и привязать его к общепринятому адресу службы
2. Перевести сокет в пассивный режим
3. Принять из сокета запрос на установление соединения и получить новый сокет для установления соединения
4. Считывать в цикле запросы от клиента, формировать ответы и отправлять их клиенту, в соответствии с прикладным протоколом
5. После завершения обмена данными с клиентом закрыть соединение и перейти к 3 пункту

Последовательный сервер без установления логического соединения

1. Создать сокет и привязать его к общепринятому адресу службы
2. Считывать в цикле запросы от клиента, формировать ответы и отправлять их клиенту, в соответствии с прикладным протоколом

Параллельный сервер без установления логического соединения

Ведущий поток:

1. Создать сокет и привязать его к общепринятому адресу службы
2. В цикле считывать запросы с помощью `recvfrom` и создавать новые ведомые потоки (процессы) для формирования ответа

Параллельный сервер без установления логического соединения

Ведущий поток:

1. Создать сокет и привязать его к общепринятому адресу службы
2. В цикле считывать запросы с помощью `recvfrom` и создавать новые ведомые потоки (процессы) для формирования ответа

Ведомый поток:

1. Работа потока начинается с получения конкретного запроса от ведущего потока, а также доступа к сокету
2. Сформировать ответ согласно прикладному протоколу и отправить его клиенту с использованием функции `sendto`
3. Завершить работу потока

Параллельный сервер с установлением логического соединения

Ведущий поток:

1. Создать сокет и привязать его к общепринятому адресу службы
2. Перевести сокет в пассивный режим
3. Вызвать в цикле функцию `accept` для получения очередного запроса от клиента и создавть новый ведомый поток или процесс для формирования ответа

Параллельный сервер с установлением логического соединения

Ведущий поток:

1. Создать сокет и привязать его к общепринятому адресу службы
2. Перевести сокет в пассивный режим
3. Вызвать в цикле функцию `accept` для получения очередного запроса от клиента и создавть новый ведомый поток или процесс для формирования ответа

Ведомый поток:

1. Работа начинается с получения доступа к соединению, полученному от ведущего потока
2. Выполнять в цикле работу с клиентом через соединение
3. Закрыть соединение и завершить работу.

Сервер с асинхронным вводом/выводом

1. Создать сокет и привязать его к общепринятому адресу службы. Добавить сокет к списку сокетов, через которые может осуществляться ввод/вывод
2. Использовать функцию `select` для получения информации о готовности существующих сокетов к вводу/выводу
3. Если готов первоначальный сокет, то использовать функцию `accept` для получения очередного запроса на установление соединения и добавить новый сокет к списку сокетов, через которые может осуществляться ввод/вывод
4. Если готов сокет, отличный от первоначального, использовать функцию `recv` для получения очередного запроса, сформировать ответ и передать ответ клиенту с использованием функции `send`
5. Перейти к пункту 2

Примеры

Пример создания серверного сокета

```
1  int servsock( char *host, char * service, char * proto,
2  struct sockaddr_in *sin) {
3    int sd;
4    if ( (sd = mksock( host, service, proto, (struct sockaddr_in *) sin)) == -1)
5      return -1;
6    if( bind( sd, (struct sockaddr *) sin, sizeof( *sin)) < 0) {
7      perror( "Ошибка при привязке сокета");
8      return -1;
9    }
10
11   if( strcmp( proto, "tcp") == 0) {
12     if ( listen( sd, qlen) == -1) {
13       perror( "Ошибка при переводе сокета в пассивный режим");
14   return -1;
15     }
16   }
17   return sd;
18 }
```

Последовательный сервер без установления логического соединения

```
1  main( void)
2  {
3  char *host = "amok.evm", *service = "2525", *proto = "udp";
4  struct sockaddr_in sin, remote;
5  struct timeval timev;
6  int sd, rlen, readed;
7  char buf[513], *t_now;
8      if ( (sd = servsock( host, service, proto,  &sin, 10)) == -1) {
9        printf( "Ошибка при создании сокета\n");
10       return 1;
11     }
12     while(1){
13       rlen = sizeof( remote);
14       if( (readed = recvfrom( sd, buf, 512, 0, (struct sockaddr *)&remote, &rlen)) !=
          -1) {
15         gettimeofday( &timev, NULL);
16         t_now = ctime( &(timev.tv_sec));
17         sendto(sd, t_now, strlen(t_now), 0, (struct sockaddr *)&remote, sizeof (
          remote));
18       }
19     }
20     return 0;
```

Последовательный сервер с установлением логического соединения

```
1   main( void)
2   {
3   ...
4   char * proto = "tcp";
5   int sd, rsd, rlen, readed;
6   ...
7       while(1) {
8         rlen = sizeof( remote);
9         rsd = accept( sd, (struct sockaddr *)&remote, &rlen);
10        if( (readed = recv(rsd, buf, 512,0)) != -1) {
11          gettimeofday( &timev, NULL);
12          t_now = ctime( &(timev.tv_sec));
13          send(rsd, t_now, strlen(t_now), 0);
14        }
15        close( rsd);
16      }
17      return 0;
18  }
```

Реализация параллельного сервера с установлением логического соединения с помощью процессов

```
1   while(1) {
2     rlen = sizeof( remote);
3     rsd = accept( sd, (struct sockaddr *)&remote, &rlen);
4     switch( fork()) {
5       case -1:
6         exit(ERR);
7       case 0:
8         close( sd);
9         exit ( TCP_Proc ( rsd) );
10      default:
11        close( rsd);
12    }
13  }
```

Реализация параллельного сервера с установлением логического соединения с помощью потоков

```
1   pthread_t th;
2   pthread_attr_t ta;
3
4     while(1) {
5       rlen = sizeof( remote);
6       rsd = accept( sd, (struct sockaddr *)&remote, &rlen);
7       pthread_create (&th,&ta, TCP_Proc, rsd);
8     }
```

Пример реализации сервера с асинхронным вводом/выводом

```
1   fd_set rfds, afds;
2   int nfds;
3     nfds = getdtablesize();
4     FD_ZERO( &afds);
5     FD_SET( sd, &afds);
6     while(1) {
7       memcpy( &rfds, &afds, sizeof(rfds));
8       if ( select( nfds, &rfds, (fd_set *) 0, (fd_set *) 0, (struct timeval *) 0) <0)
9         return 2;
10      if( FD_ISSET( sd, &rfds)) {
11        rlen = sizeof( remote);
12        rsd = accept( sd, (struct sockaddr *)&remote, &rlen);
13        FD_SET( rsd, &afds);
14      }
15      for( rsd=0; rsd < nfds; ++rsd)
16        if( (rsd != sd) && FD_ISSET( rsd, &rfds)) {
17          TCP_Proc( rsd);
18          close( rsd);
19          FD_CLR( rsd, &afds);
20        }
21    }
```

Альтернативное устройство параллельного сервера

- предварительное создание дочерних процессов (preforking)
- предварительное создание потоков (prethreading)
- мультисервисные серверы (суперсервер)

preforked server

- сервер с предварительным созданием дочерних процессов с параллельным вызовом accept
- сервер с предварительным созданием дочерних процессов с блокировкой для защиты accept
- сервер с предварительным созданием дочерних процессов с использованием взаимного исключения для защиты accept
- сервер с предварительным созданием дочерних процессов с последующей передачей дескриптора сокета дочерним процессам

prethreaded server

- сервер с предварительным созданием потоков с использованием взаимного исключения для защиты accept
- сервер с предварительным созданием потоков, главный поток вызывает accept

Пример сервера с предварительным созданием процессов с параллельным вызовом accept

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <errno.h>
4   #include <string.h>
5   #include <netdb.h>
6   #include <time.h>
7   #include <sys/types.h>
8   #include <sys/socket.h>
9
10  #define CHILDS 10
11
12  int mksock( char *host, char * service, char * proto, struct sockaddr_in *sin)
```

```
54  Ошибкаприраспределениисокета
55
56  main( void)
57  {
58
59  char * host = "0.0.0.0";
60  char * service = "2525";
61  char * proto = "tcp";
62  //char * proto = "sctp";
63  struct sockaddr_in sin, remote;
64  struct tm *tp;
65  time_t t;
66  int i, sd, rsd, rlen, readed, pid;
67  char buf[513], t_str[512];
68
69
70      if ( (sd = mksock( host, service, proto, &sin)) == -1)
71      {
72      perror( "Ошибка при создании сокета");
73      return 1;
74      }
75
76      printf( "Адрес сервера %s = %s\n", host, (char *) (inet_ntoa( sin.sin_addr)));
77      printf( "Адрес порта сервера %s = %X\n", service, sin.sin_port);
78
79
80    i = 1;
81    i = setsockopt( sd, SOL_SOCKET, SO_REUSEADDR, &i, sizeof( &i));
82    if( i != 0) perror("Опция сокета (SOL_SOCKET, SO_REUSEADDR))");
83
84    if( bind( sd, (struct sockaddr *) &sin, sizeof( sin)) < 0)
85    {
86      perror( "Ошибка при привязке сокета");
87      return 1;
88    }
```

```
89  ОшибкаприраспределенииСокетаОшибкаприсозданииСокетаАдресСервераАдресПортаСервераОпцияСокетаОш
90    for ( i=0; i<CHILDS; i++)
91    {
92      pid = fork();
93      switch ( pid)
94      {
95        case -1:
96          perror( "Не создается дочерний процесс");
97          break;
98        case 0:
99          pid = i+1;
100         i=CHILDS+1;
101       break;
102       default:
103         pid=0;
104     }
105   }
106
107     if ( listen( sd, 0) == -1)
108     {
109   perror( "Ошибка при переводе сокета в пассивный режим");
110         return 1;
111   }
112
113     while(1)
114     {
115     rlen = sizeof( remote);
116     rsd = accept( sd, (struct sockaddr *)&remote, &rlen);
117     fprintf( stderr, "Process %d accepted connection\n", pid);
118     t = time( NULL);
119     tp = localtime( &t);
120     strftime( t_str, 512, "%a, %d %b %Y %T %z", tp);
121     snprintf( buf, 512, "Server [%d]: %s\n", pid, t_str );
122     send(rsd, buf, strlen(buf), 0);
123     close( rsd);
124     }
125
```

Пример скрипта для теста сервера

```
1  #!/bin/bash
2
3  for((i=0;i<20;i++))
4  do
5      nc localhost 2525
6  done
```

Вывод

```
Server [0]: Wed,  20 Oct 2010 12:39:37 +0300
Server [4]: Wed,  20 Oct 2010 12:39:37 +0300
Server [3]: Wed,  20 Oct 2010 12:39:37 +0300
Server [5]: Wed,  20 Oct 2010 12:39:37 +0300
Server [6]: Wed,  20 Oct 2010 12:39:37 +0300
Server [7]: Wed,  20 Oct 2010 12:39:37 +0300
Server [8]: Wed,  20 Oct 2010 12:39:38 +0300
Server [2]: Wed,  20 Oct 2010 12:39:38 +0300
Server [9]: Wed,  20 Oct 2010 12:39:38 +0300
Server [10]: Wed,  20 Oct 2010 12:39:38 +0300
Server [1]: Wed,  20 Oct 2010 12:39:38 +0300
```

Пример UDP сервера с предварительным созданием процессов

```
1   --- preforked_server.c  2011-10-24 11:55:18.255762085 +0300
2   +++ preforked_server_udp.c  2010-10-20 12:26:02.000000000 +0300
3   @@ -58,7 +58,7 @@
4
5    char * host = "0.0.0.0";
6    char * service = "2525";
7   -char * proto = "tcp";
8   +char * proto = "udp";
9    //char * proto = "sctp";
10   struct sockaddr_in sin, remote;
11   struct tm *tp;
12  @@ -104,23 +104,17 @@
13     }
14    }
15
16  -    if ( listen( sd, 0) == -1)
17  -    {
18  - perror( Ошибка" при переводе сокета в пассивный режим");
19  -        return 1;
20  - }
21
22  +    rlen = sizeof( remote);
23       while(1)
24       {
25  -    rlen = sizeof( remote);
26  -    rsd = accept( sd, (struct sockaddr *)&remote, &rlen);
27  +    recvfrom( sd, buf, 512, 0, (struct sockaddr *)&remote, &rlen);
28       fprintf( stderr, "Process %d accepted connection\n", pid);
29       t = time( NULL);
30       tp = localtime( &t);
```

Пример скрипта для теста UDP сервера

```
1  #!/bin/bash
2
3  (for((i=0;i<20;i++))
4  do
5      echo test
6      sleep 1
7  done) | nc -w 3 -u localhost 2525
```

Вывод

```
Server [0]: Wed,  20 Oct 2010 12:41:03 +0300
Server [2]: Wed,  20 Oct 2010 12:41:04 +0300
Server [3]: Wed,  20 Oct 2010 12:41:05 +0300
Server [4]: Wed,  20 Oct 2010 12:41:06 +0300
Server [5]: Wed,  20 Oct 2010 12:41:07 +0300
Server [6]: Wed,  20 Oct 2010 12:41:08 +0300
Server [7]: Wed,  20 Oct 2010 12:41:09 +0300
Server [8]: Wed,  20 Oct 2010 12:41:10 +0300
Server [9]: Wed,  20 Oct 2010 12:41:11 +0300
Server [1]: Wed,  20 Oct 2010 12:41:12 +0300
Server [10]: Wed,  20 Oct 2010 12:41:13 +0300
```

Пример сервера с предварительным созданием дочерних процессов с последующей передачей дескриптора сокета дочерним процессам

```
 1  #include <stdio.h>
 2  #include <stdlib.h>
 3  #include <errno.h>
 4  #include <string.h>
 5  #include <netdb.h>
 6  #include <time.h>
 7  #include <sys/types.h>
 8  #include <sys/socket.h>
 9  #include <signal.h>
10
11  #define CHILDS 10
12
13  typedef struct{
14    pid_t cpid; //ID дочернего процесса
15    int cfd; // канал для связи с дочерним процессом
16    int status; // свободензанят/
17  } child;
18
19  child cptr[CHILDS];
```

```
63  дочернегопроцессаканалдлясвязисдочернимпроцессомсвободензанятОшибкапираспределениисокета
64
65  int signal_handler(int sig, siginfo_t *info,void *args)
66  {
67  int i;
68    fprintf(stderr, "Signal catched from %d\n", info->si_pid);
69    for(i=0; i<CHILDS; i++)
70      if( info->si_pid == cptr[i].cpid)
71        cptr[i].status=0;
72    return 1;
73  }
```

```
74  дочернегопроцессаканалдлясвязисдочернимпроцессомсвободензанятОшибкаприраспределениисокета
75  int read_fd(int recvfd)
76  {
77  struct msghdr    msg;
78  struct cmsghdr  *cmsg;
79  union {
80    struct cmsghdr hdr;
81    unsigned char    buf[CMSG_SPACE(sizeof(int))];
82  } cmsgbuf;
83  int fd;
84
85  struct iovec iov;
86
87  char ptr[]={0};
88      memset(&msg, 0, sizeof(msg));
89      msg.msg_control = &cmsgbuf.buf;
90      msg.msg_controllen = sizeof(cmsgbuf.buf);
91
92      msg.msg_name=NULL;
93      msg.msg_namelen = 0;
94
95      iov.iov_base = ptr;
96      iov.iov_len = 1;
97      msg.msg_iov = &iov;
98      msg.msg_iovlen = 1;
99
100     recvmsg(recvfd, &msg, 0);
101     fprintf( stderr, "Received message\n");
102     for (cmsg = CMSG_FIRSTHDR(&msg); cmsg != NULL;
103       cmsg = CMSG_NXTHDR(&msg, cmsg)) {
104     if (cmsg->cmsg_len == CMSG_LEN(sizeof(int)) &&
105       cmsg->cmsg_level == SOL_SOCKET &&
106       cmsg->cmsg_type == SCM_RIGHTS) {
107         fd = *(int *)CMSG_DATA(cmsg);
108         return(fd);
109       }
110    }
```

```
111  дочернегопроцессаканалдлясвязисдочернимпроцессомсвободензанятОшибкаприраспределениисокета
112
113  void send_fd(int connfd, int sendfd)
114  {
115  struct msghdr    msg;
116  struct cmsghdr  *cmsg;
117  struct iovec iov;
118
119  char ptr[]={0};
120
121
122  union {
123    struct cmsghdr hdr;
124    unsigned char    buf[CMSG_SPACE(sizeof(int))];
125    } cmsgbuf;
126
127      memset(&msg, 0, sizeof(msg));
128      msg.msg_control = &cmsgbuf.buf;
129      msg.msg_controllen = sizeof(cmsgbuf.buf);
130
131      msg.msg_name=NULL;
132      msg.msg_namelen = 0;
133
134      iov.iov_base = ptr;
135      iov.iov_len = 1;
136      msg.msg_iov = &iov;
137      msg.msg_iovlen = 1;
138
139      cmsg = CMSG_FIRSTHDR(&msg);
140      cmsg->cmsg_len = CMSG_LEN(sizeof(int));
141      cmsg->cmsg_level = SOL_SOCKET;
142      cmsg->cmsg_type = SCM_RIGHTS;
143      *(int *)CMSG_DATA(cmsg) = sendfd;
144
145      sendmsg(connfd, &msg, 0);
146  }
```

```
148  дочернегопроцессаканалдлясвязисдочернимпроцессомсвободензанятОшибкапираспределенииисокета
149  main( void)
150  {
151
152  char * host = "0.0.0.0";
153  char * service = "2525";
154  char * proto = "tcp";
155  //char * proto = "sctp";
156  struct sockaddr_in sin, remote;
157  struct tm *tp;
158  time_t t;
159  int i, sd, rsd, rlen, readed, pid, ppid;
160  int sockfd[2];
161  char buf[513], t_str[512];
162
163  struct sigaction action;
164
165     if ( (sd = mksock( host, service, proto, &sin)) == -1)
166     {
167     perror( "Ошибка при создании сокета");
168     return 1;
169     }
170
171     printf( "Адрес сервера %s = %s\n", host, (char *) (inet_ntoa( sin.sin_addr)));
172     printf( "Адрес порта сервера %s = %X\n", service, sin.sin_port);
173
174
175    i = 1;
176    i = setsockopt( sd, SOL_SOCKET, SO_REUSEADDR, &i, sizeof( &i));
177    if( i != 0) perror("Опция сокета (SOL_SOCKET, SO_REUSEADDR))");
178
179    if( bind( sd, (struct sockaddr *) &sin, sizeof( sin)) < 0)
180    {
181      perror( "Ошибка при привязке сокета");
182      return 1;
183    }
```

```
185   дочернегопроцессаканалдлясвязисдочернимпроцессомсвободензанятОшибкаприраспределенииисокетаОшибк
186     for ( i=0; i<CHILDS; i++)
187     {
188       socketpair( AF_LOCAL, SOCK_STREAM, 0, sockfd);
189       pid = fork();
190       switch ( pid)
191       {
192         case -1:
193           perror( "Не создается дочерний процесс");
194           break;
195         case 0: //дочерний
196           pid = i+1;
197           i=CHILDS+1;
198           close( sd);
199           close( sockfd[0]);
200           ppid=getppid();
201
202           break;
203         default: // родительский
204           close( sockfd[1]);
205           cptr[i].cpid = pid;
206           cptr[i].cfd = sockfd[0];
207           cptr[i].status = 0;
208
209           pid=0;
210
211           action.sa_handler = signal_handler;
212           sigemptyset (&action.sa_mask);
213           action.sa_flags = SA_SIGINFO;
214           sigaction (SIGUSR1, &action, NULL);
215
216
217       }
218     }
```

```
219   дочернегопроцессаканалдлясвязисдочернимпроцессомсвободензанятОшибкапираспределениисокетаОшибк
220     if( pid == 0)
221     { // Управляющий сервер
222       if ( listen( sd, 0) == -1)
223       {
224         perror( "Ошибка при переводе сокета в пассивный режим");
225         return 1;
226       }
227
228       while(1)
229       {
230         rlen = sizeof( remote);
231         while ( (rsd = accept( sd, (struct sockaddr *)&remote, &rlen)) == -1);
232         i=0;
233         while(cptr[i].status != 0){
234           if ( i == CHILDS){
235             i=0;
236             fprintf( stderr, "Main process: all childs are busy\n");
237             sleep(1);
238           }
239           i++;
240         }
241         fprintf( stderr, "Main process accepted connection and pass it to [%d]\n", cptr[i].cpid);
242         cptr[i].status=1;
243         send_fd( cptr[i].cfd, rsd);
244
245         close( rsd);
246       }
```

```
247   дочернегопроцессаканалдлясвязисдочернимпроцессомсвободензанятОшибкапираспределенииисокетаОшибк
248     } else { // Управляемый сервер
249       while(1)
250       {
251         rsd=read_fd( sockfd[1]);
252         t = time( NULL);
253         tp = localtime( &t);
254         strftime( t_str, 512, "%a, %d %b %Y %T %z", tp);
255         snprintf( buf, 512, "Server [%d]: %s\n", pid, t_str );
256         send(rsd, buf, strlen(buf), 0);
257         close( rsd);
258         sleep( 3);
259         kill( ppid, SIGUSR1);
260       }
261
262     }
263
264       return 0;
265   }
```

Вывод

```
Server [1]: Mon,  24 Oct 2011 13:16:52 +0300
Server [2]: Mon,  24 Oct 2011 13:16:52 +0300
Server [3]: Mon,  24 Oct 2011 13:16:52 +0300
Server [4]: Mon,  24 Oct 2011 13:16:52 +0300
Server [5]: Mon,  24 Oct 2011 13:16:52 +0300
Server [6]: Mon,  24 Oct 2011 13:16:52 +0300
Server [7]: Mon,  24 Oct 2011 13:16:52 +0300
Server [8]: Mon,  24 Oct 2011 13:16:52 +0300
Server [9]: Mon,  24 Oct 2011 13:16:52 +0300
Server [10]: Mon,  24 Oct 2011 13:16:52 +0300
Server [2]: Mon,  24 Oct 2011 13:16:55 +0300
Server [1]: Mon,  24 Oct 2011 13:16:55 +0300
Server [3]: Mon,  24 Oct 2011 13:16:55 +0300
Server [4]: Mon,  24 Oct 2011 13:16:55 +0300
Server [5]: Mon,  24 Oct 2011 13:16:55 +0300
Server [6]: Mon,  24 Oct 2011 13:16:55 +0300
Server [7]: Mon,  24 Oct 2011 13:16:55 +0300
Server [8]: Mon,  24 Oct 2011 13:16:55 +0300
Server [10]: Mon,  24 Oct 2011 13:16:55 +0300
Server [2]: Mon,  24 Oct 2011 13:16:58 +0300
```

Сервер с предварительным созданием процессов с блокировкой вызова accept

- С помощью файла
- С помощью взаимного исключения (Семафор, мьютекс, критические секции windows)

Мультисервисные серверы

Синоним «Суперсервер»
Используется не отдельный сервер, а "враппер"для сервисов.

- уменьшение надежности
- ограничения по количеству открытых сокетов
- потребляет меньше ресурсов

Мультисервисный сервер без установления соединения

1. Сервер открывает набор сокетов UDP и привязывает к портам служб.
2. Используется таблица соответствия сокетов службам
3. с помощью select сервер переводится в состояние ожидания дейтаграммы
4. для обработки каждой отдельной дейтаграммы вызывается соответствующая функция

Мультисервисный сервер с установлением соединения

1. Сервер открывает набор сокетов TCP и привязывает к портам служб.
2. Используется таблица соответствия сокетов службам
3. с помощью select сокеты переводится в состояние ожидания нового соединения
4. Если готов один из первоначальных сокетов, то создаем новый сокет.
5. Для обработки каждого соединения вызывается соответствующая функция

Модульный мультисервисный сервер с устанавлением соединения

1. Сервер открывает набор сокетов TCP и привязывает их к портам служб.
2. Используется таблица соответствия сокетов службам
3. с помощью select сокеты переводится в состояние ожидания нового соединения
4. После поступления запроса вызываем fork для создания ведомого процесса
5. ведомый процесс закрывает все ненужные сокеты
6. ведомый процесс производит замещение процесса с помощью вызова из семейства exec

Работа через initd/xinitd

```
1  /* Если работаем с помощью xinetd,то
2   перенаправляем stdout и stderr в сокет) */
3  if( xinetd)
4  {
5    close( stdout);
6    close( stderr);
7    if ( dup2( 0,  1) == -1) return 1;
8    if ( dup2( 0,  2) == -1) return 1;
9  }
```

Общие рекомендации по работе сервера

- Функционирование в фоновом режиме:
  с помощью fork запускаем копию программы, а родительский процесс убивается. При этом должны закрываться все унаследованные дескрипторы файлов.
- Сервер должен отключать управляющий терминал, чтобы не получать от него сигналы:

```
fd=open(“/dev/tty”,  O_RDWR);
ioctl( fd,  TIOCNOTTY,  0);
close( fd);
```

- Сервер должен переходить в безопасный каталог используя `chdir`
- Необходимо изменить маску по умолчанию для создания файлов используя `umask( 027)`
- Необходимо открыть стандартные дескрипторы для корректной работы библиотечных процедур:

```
fd=open(“/dev/null”,  O_RDWR); //ввод
dup( fd); //вывод
dup( fd); //ошибки
```

- Сервер должен предотвращать запуск нескольких копий (лок-файлы или др. системные функции)
- Желательно игнорировать сигналы, не относящиеся к работе сервера
- Если есть возможность, то желательно использовать системные методы ведения журнала событий

Спасибо за внимание!
Вопросы?