Анализ сервера:

```
#include <unistd.h> // системные вызовы open,...
#include <stdio.h>
#include <stdlib.h>
#include <signal.h> // подключение сигналов
#include <sys/file.h> // для констант
#include <map>  // удобная структура из STL

using namespace std;

#define TRUE 1

#define LOG_FILE "/tmp/unique_my_log"    // лог-файл
int handler_log_file;
int global_count=0;
FILE *log=NULL;      // указатель на лог-файл

#define error_in_server_ "/tmp/error_in_server_"  // файл ошибок создаётся в
                                                  //случае проблем в работе сервера

#define REPORT_ERROR(a) \   // для отчета об ошибке
{   FILE *f=fopen(error_in_server_,"w+"); \
    fprintf(f,"%s\n",a); \
    fclose(f); \
    fclose(log); \
    _exit(0); \
}

void finish_work(int sig_num) // освобождение ресурсов при выходе
{
    fclose(log);
    remove(error_in_server_);
    remove("/tmp/error_in_client_");
    _exit(0);
}

#define _POSIX_C_SOURCE 199310  // для возможности взведения флага sa_flags
                                //необходимо объявить константу

static  map<unsigned int,unsigned int> pid_numbers;   // структура для
                                                      //хранения прав процесса
static  map<unsigned int,unsigned int>::iterator it_iterator; // просто //итератор


     //обработчик SIGALRM,  по приходу этого сигнала выполняется задание

void signal_work_is_done(int sig_num,siginfo_t *siginfo, void *ucontext) //SIGALRM
{
  int prev_mask=sigsetmask(0xFFFFffff);

  if(pid_numbers.find(siginfo->si_pid)==pid_numbers.end() )
      ; //silently ignore
  else
```

```
 {
     if (pid_numbers.find(siginfo->si_pid)->second==false)
         return;

   if( fprintf(log,"sender_id  %d    |   %d\n",siginfo->si_pid,global_count++) <0
)
        REPORT_ERROR("fprintf server error happened");

      if( kill(siginfo->si_pid,sig_num)==-1)
       { REPORT_ERROR("cannot send syncronization signal"); _exit(0); }
  }
  sigsetmask(prev_mask);
}

void send_signal_syncronization(int sig_num, siginfo_t *siginfo,  void *ucontext)
// SIGWINCH
{
   int prev_mask=sigsetmask(0xFFFFffff);
   static int id_numb_of_signal;

  //protection from other users and root haha

    if( (pid_numbers.find(siginfo->si_pid))!=pid_numbers.end() ) //    если нет в
множестве, а там и не должно быть!
      REPORT_ERROR("уже есть там - нехорошо")  //сворачиваемся == wind up
    else
     {  pid_numbers.insert(make_pair(siginfo->si_pid,false));

      if( kill(siginfo->si_pid,SIGINT)==-1)
         REPORT_ERROR("cannot send syncronization signal");
     }
   sigsetmask(prev_mask);
}

void multiple_existense()
{
    int handler_log_file=open(LOG_FILE,O_CREAT | O_RDWR, S_IRUSR | S_IWUSR);

//функция открытия используется с параметрами O_CREAT(файл будет создан,     //если
он не существует) и O_RDWR(файл будет открыт для WR). Тоесть, каждый         //
новый экземпляр процесса будет пытаться создать файл, если его нет.



    if(lockf(handler_log_file,F_TLOCK,1)==-1)  // так процесс может заблокировать
                                               //файл, при этом другие процессы
                                               //пытающиеся открыть файл для себя
                                               //получат ошибку.
REPORT_ERROR(“cannot lock file, there can be only one server in your OS”);
}

void realtime_wait(int sig_num, siginfo_t *siginfo,  void *ucontext) // проверка
сигнала
{
    int a=siginfo->si_value.sival_int;
```

```
  static int ar[5]={1,2,3,4,5};  //  super confidential cipher
  static int glob_var;  // index of array

  if (glob_var==4)
     {
     if( (it_iterator=pid_numbers.find(siginfo->si_pid))==pid_numbers.end())
//there is no even this signal in list!
             REPORT_ERROR("somebody fools us here! in realtime handler")
          else
             it_iterator->second=TRUE;  // it is our signal
     }

    if(glob_var>4)
     glob_var=0;

    if(a!=ar[glob_var])
      REPORT_ERROR("somebody tries to fool us again");
  glob_var++;
}

int  becomeDaemon()
{
 switch (fork())
  {
        case -1: return -1; //error  ненормальное поведение
        case 0: break;   //child  мы в новосозданном процессе, продолжаем
        default: _exit(0);  //parent в родителе, выходим
  }
 struct sigaction neu;
 struct sigaction work_done;
 struct sigaction realtime;

 multiple_existense();  //  обеспечение присутствия только одного сервера.
//Присутствие более одного, вызовет хаос

 if(chdir("/")==-1)//сменим текущую директорию, потому что сервер может быть
                   //запущен в директории, которая может быть удалена,
                   // а “/” единственная директория наличие, которой
                  //гарантируется на всех *nix
 {   printf("cannot change directory");
     return 0;
 };

 signal(SIGTERM,finish_work); // установка сигнала для корректного
                              //завершения работы сервера, явно выполняется
                              // закрытие лог-файла log

 log=fopen(LOG_FILE,"w+");

realtime.sa_sigaction=realtime_wait; // привязка обработчика сигнала к
                                        //к структуре. При этом параметры
                                       //обработчика должны соответствовать
                                      // шаблону. Для более подробной информа-
                                      //ции man 2 sigaction.

realtime.sa_flags=SA_SIGINFO; // функция sigaction может работать с двумя
```

```
                              //видами структур. Тип структуры определяется
      // полем sa_flags. Для более подробной информа-
//ции man 2 sigaction

                      // аналогичные действия

neu.sa_sigaction=send_signal_syncronization;
neu.sa_flags=SA_SIGINFO;

work_done.sa_sigaction=signal_work_is_done;
work_done.sa_flags=SA_SIGINFO;

if(sigaction(SIGRTMIN,&realtime,NULL)==-1)   // вызов функции установки
                  //обработчика сигнала SIGRTMIN
{ REPORT_ERROR("cannot establish\n"); exit(0);}

 if(sigaction(SIGWINCH,&neu,NULL)==-1)
 { REPORT_ERROR("cannot establish\n"); exit(0);}

 if(sigaction(SIGALRM,&work_done,NULL)==-1)
 { REPORT_ERROR("cannot establish\n"); exit(0);}

 while(1)
   pause(); //переходим в sleep состоняние,т.к. основной поток стоит
      //можно делать в handler, "всё"(то,что делается), отдельный,
      //поток, не перекликается с main thread
}

main()
{
 becomeDaemon();  // создаём новый процесс для отвязки от терминала
}
```