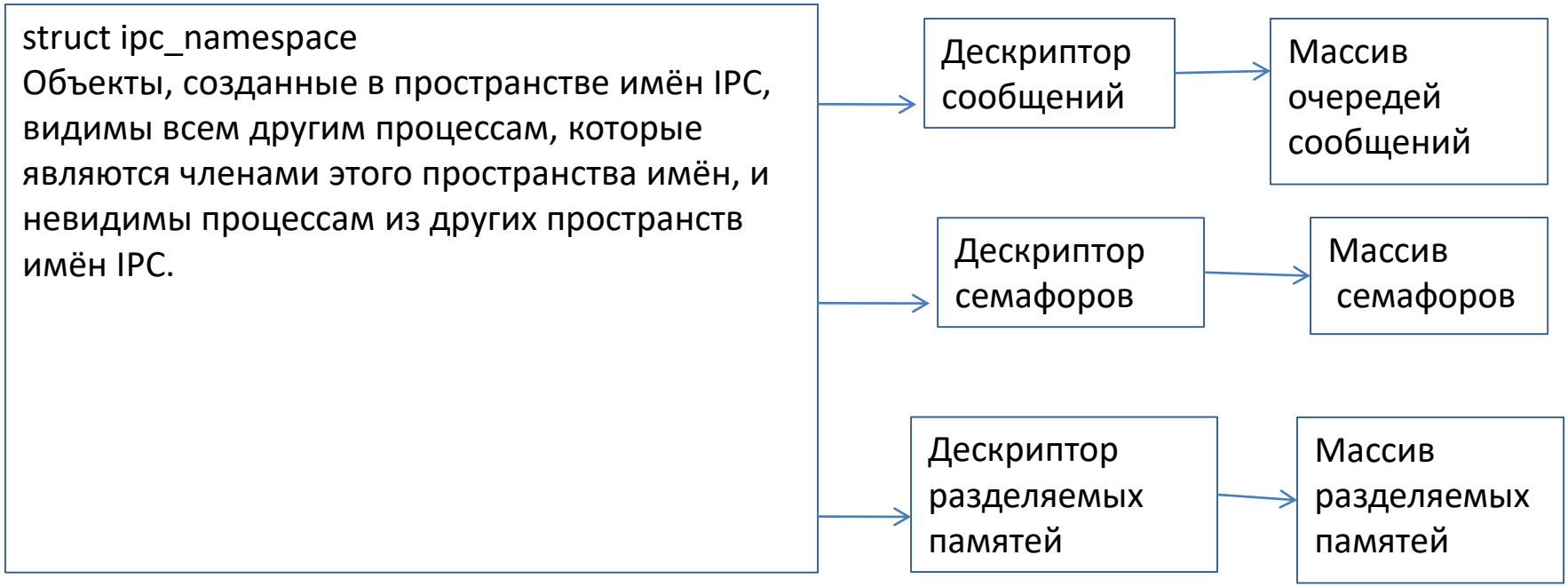
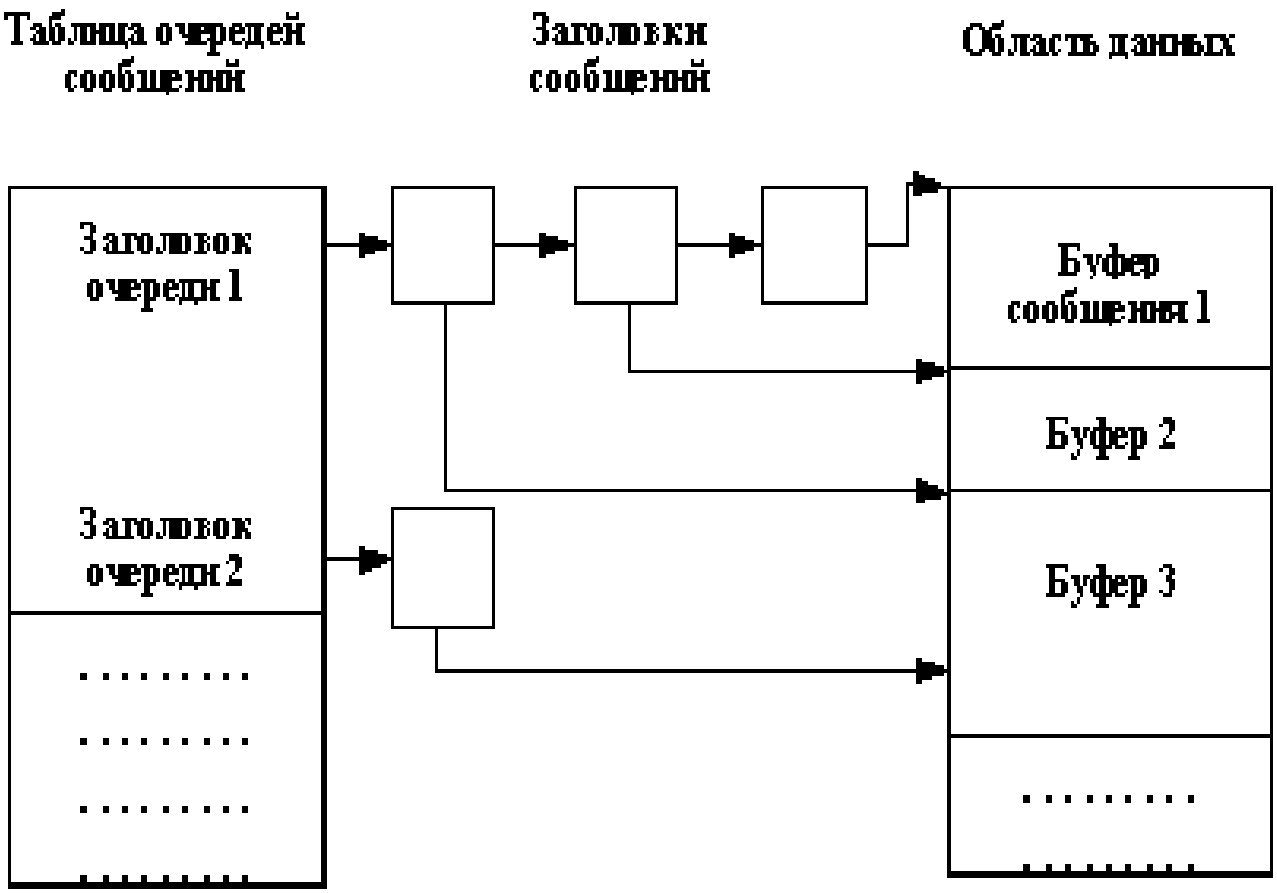
Организация межпроцессного взаимодействия




```
struct ipc_ids {
int size;
int in_use;
int max_id;
unsigned short seq; // порядковый номер IPC ресурса
unsigned short seq_max; //
struct semaphore sem; // семафор для доступа к дескриптору
spinlock_t ary; // блокировка доступа к массиву
struct ipc_id* entries; // указатель на массив
};
```

Организация очередей сообщений



Заголовок очереди сообщений

```
struct  msg_queue {
struct  kern_ipc_perm q_perm; /* атрибуты очереди*/
time_t q_stime; /* время последнего вызова msgsnd */
time_t q_rtime; /* время последнего вызова msgrcv */
time_t q_ctime; /* время последнего изменения */
unsigned long q_cbytes; /* текущий размер очереди в байтах */
unsigned long q_qnum; /* количество сообщений в очереди */
unsigned long q_qbytes; /* максимальный размер очереди в байтах */
pid_t q_lspid; /* pid последнего процесса вызвавшего msgsnd */
pid_t q_lrpid; /* pid последнего процесса-получателя */
struct list_head q_messages; /* очередь сообщений */
struct list_head q_receivers; /* очередь процессов , ожидающих  приема
сообщений */
struct list_head q_senders; /* очередь процессов , ожидающих  передачи
сообщений */
};
```

Атрибуты очереди сообщений

```
struct kern_ipc_perm {
        spinlock_t      lock;
        int             deleted;
        int             id; /*идентификатор очереди сообщений*/
        key_t           key; /*ключ очереди сообщений*/
        kuid_t          uid; /*идентификатор пользователя
                        (владельца)*/
        kgid_t          gid; /*групповой идентификатор
                        пользователя (владельца) */
        kuid_t          cuid; /*идентификатор пользователя
                        (создателя)*/
        kgid_t          cgid; /*групповой идентификатор
                                пользователя (создателя) */
        umode_t         mode; /* флаги, характеризующие допустимые
                                операции с очередью */
        unsigned long   seq; /*внутренний идентификатор   очереди */
        void            *security;
};
```

Предельные характеристики очереди сообщений

ipcs -q -l

------ Лимиты сообщений --------

максимум очередей для всей системы = 1738

максимальный размер сообщения (байт) = 8192

максимальный по умолчанию размер очереди сообщений (байт) = 16384

Подходы к работе с очередью сообщений

- в стиле UNIX System V, используя функции расширения POSIX:XSI;
- через функции POSIX (стандарт POSIX.1-2001)

Главные отличия заключаются в следующем:

- операция считывания из очереди сообщений Posix всегда возвращает самое старое сообщение с наивысшим приоритетом (типом), тогда как из очереди System V можно считать сообщение с произвольно указанным приоритетом;
- Если мы помещаем первое сообщение в пустую очередь, можно проверить, не зарегистрирован ли какой-нибудь процесс на уведомление об этом событии и нет ли потоков, заблокированных в вызове mq_receive очереди сообщений. Posix позволяют отправить сигнал или запустить программный поток при помещении сообщения в пустую очередь, тогда как для очередей System V ничего подобного не предусматривается.

Протокол обмена сообщениями

Если два неродственных процесса используют какой-либо вид IPC для обмена информацией, объект IPC должен иметь имя или идентификатор, чтобы один из процессов (называемый обычно сервером — server) мог создать этот объект, а другой процесс (обычно один или несколько клиентов — client) мог обратиться к этому конкретному объекту.

ПРОЦЕСС А:

- создает или получает очередь сообщений с заданным ключом
- работает со своими данными
- посылает сообщение процессу В
- получив ответ от процесса В - начинает передачу ему данных

ПРОЦЕСС В:

- создает или получает очередь сообщений с заданным ключом
- работает со своими данными
- ожидает сообщение от процесса А
- отвечает на сообщение
- принимает данные

Создание очереди сообщений

int msgget (key_t key, int msgflg)

Возвращает идентификатор очереди сообщений в случае успеха; -1 в случае ошибки.

При этом errno =

EACCESS (доступ отклонен)

EEXIST (такая очередь уже есть, создание невозможно)

EIDRM (очередь помечена как удаляемая)

ENOENT (очередь не существует)

ENOMEM (не хватает памяти для создания новой очереди)

ENOSPC (исчерпан лимит на количество очередей)

Параметры операции создания очереди сообщений

int msgget (key_t key, int msgflg)

key – ключ сообщения (IPC_PRIVATE или целое > 0)

msgflg – набор флагов

IPC_CREAT | IPC_EXCL | права доступа

0400 - разрешен прием сообщений пользователю, владеющему очередью;

0200 - разрешена передача сообщений пользователю, владеющему очередью;

0040 - разрешен прием сообщений пользователям, входящим в ту же группу, что и владелец очереди;

0020 - разрешена передача сообщений пользователям, входящим в ту же группу, что и владелец очереди;

0004 - разрешен прием сообщений всем остальным пользователям;

0002 - разрешена передача сообщений всем остальным пользователям.

Пример создание очереди сообщений

Процесс 1 создатель и владелец очереди:

int id_1 =msgget(121,0622 | IPC_CREAT);

Процессу 1 разрешен прием и передача сообщений, остальным разрешено только передавать сообщения.

Процесс 2 получает доступ к очереди по ключу для передачи сообщений:

int id_2=msgget(121, IPC_CREAT);

Структура хранения сообщения

```
struct msg_msg {
    struct list_head m_list; /* следующее сообщение в очереди*/
    long  m_type; /* тип сообщения*/
    int m_ts;           /* размер сообщения*/
    struct msg_msgseg* next; /* следующая часть сообщения, если оно превышает DATALEN_MSG */
    void *security;
    /* Далее следует само сообщение */
};
#define DATALEN_MSG         (PAGE_SIZE-sizeof(struct msg_msg))
```

Передача сообщения

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);

Возвращает 0 в случае успеха -1 в случае ошибки:

errno =

EAGAIN (очередь переполнена, и установлен IPC_NOWAIT)

EACCES (доступ отклонен, нет разрешения на запись)

EFAULT (адрес msgp недоступен)

EIDRM (очередь сообщений удалена)

EINVAL (ошибочный идентификатор очереди сообщений, неположительный тип сообщения или неправильный размер сообщения)

ENOMEM (не хватает памяти для копии буфера сообщения)

Параметры операции передачи сообщения

int msgsnd(int msqid, const void *msgp, size_t msgsz, int msgflg);

msqid – идентификатор очереди сообщений;

msgp – адрес буфера сообщения;

msgsz – размер сообщения;

msgflg – флаги операции (0, либо IPC_NOWAIT- процесс не приостанавливается, если размер очереди > 16384).

Сообщение должно начинаться с элемента типа long int

```
struct msgbuf {
        long mtype;       /* тип сообщения  > 0 */
        char mtext[msgsz];   /* текст сообщения*/
    };
```

Пример передачи сообщения

```
struct msg_time {   /* сообщение клиенту */
        long pr_server;  /* идентификатор процесса сервера */
        time_t vtime;  /* дата и время */
     } msg_t ;
/* формирование сообщения клиенту */
        msg_t.pr_server=getpid();  /* идентификатор сервера */
        msg_t.vtime=time(0);            /* дата и время */
/* посылка сообщения клиенту */
    msgsnd(id_client,&msg_t,sizeof(time_t),0);
```

Прием сообщения

ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg) – возвращает размер прочитанного сообщения (ожидание, если сообщение не найдено и не установлен флаг IPC_NOWAIT) или -1 в случае ошибки:

errno =

E2BIG (длина сообщения больше, чем msgsz)

EACCES (нет права на чтение)

EFAULT (адрес, на который указывает msgp, ошибочен)

EIDRM (очередь была уничтожена в период изъятия сообщения)

EINTR (прервано поступившим сигналом)

EINVAL (msgqid ошибочен или msgsz меньше 0)

ENOMSG (установлен IPC_NOWAIT, но в очереди нет ни одного сообщения, удовлетворяющего запросу)

Параметры операции приема сообщения

ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg)

msqid – идентификатор очереди сообщений;

msgp – адрес буфера сообщения;

msgsz – максимальный размер сообщения;

msgtyp – тип сообщения (0-первое сообщение из очереди, >0- первое указанного типа и первое другого типа при флаге MSG_EXCEPT), <0 – первое, у которого тип не превышает абсолютного значения параметра) ;

msgflg – флаги операции (IPC_NOWAIT- процесс не ожидает сообщения, возвращает -1;

MSG_NOERROR – не возвращает код ошибки, если размер сообщения превышает msgsz;

MSG_EXCEPT - получает первое сообщение, у которого тип не равен msgtyp).

Пример приема сообщения

```
/* сообщение от сервера */
struct msg_time {
        long id_server;      /* идентификатор
   процесса сервера */
        time_t vtime[1];     /* дата и время */
    } msg_t ;
/*прием сообщения от сервера*/
msgrcv(id_client,&msg_t,sizeof(time_t),0,0);
```

Контроль и управление состоянием очереди

int msgctl(int msqid, int cmd, struct msqid_ds *buf);

Возвращает: 0 в случае успеха -1 в случае неудачи

errno =

EACCES (нет прав на чтение и в cmd указан IPC_STAT)

EFAULT (адрес, на который указывает buf, ошибочен для команд IPC_SET и IPC_STAT)

EIDRM (очередь была уничтожена во время запроса)

EINVAL (ошибочный msqid или msgsz меньше 0)

EPERM (IPC_SET- или IPC_RMID-команда была послана процессом, не имеющим прав на запись в очередь)

Параметры операции контроля и управления состоянием очереди

int msgctl(int msqid, int cmd, struct msqid_ds *buf);

msqid – идентификатор очереди;

cmd – режим работы функции (IPC_STAT, IPC_SET, IPC_RMID);

buf – информация о состоянии очереди

Для режимов IPC_SET и IPC_RMID эффективный идентификатор пользователя процесса, вызывающего эту функцию, должен принадлежать либо root, либо создателю или владельцу очереди сообщений.

Информация о состоянии очереди

```
struct msqid_ds {
        struct ipc_perm msg_perm;     /* информация о владельце и правах */
        time_t          msg_stime; /* время последнего вызова msgsnd */
        time_t          msg_rtime; /* время последнего вызова msgrcv */
        time_t          msg_ctime; /* время последнего изменения  параметров*/
        unsigned long   __msg_cbytes; /* количество байт в очереди */
        msgqnum_t       msg_qnum;  /* количество сообщений в очереди */
        msglen_t        msg_qbytes;   /* максимальное число байт в очереди */
        pid_t           msg_lspid;    /* pid последнего процесса-отправителя */
        pid_t           msg_lrpid;    /* pid последнего процесса-получателя */
      };
```

Информация о владельце и правах очереди

```
struct ipc_perm {
    key_t    __key;      /* ключ очереди */
    uid_t    uid;        /* эффективный UID владельца */
    gid_t    gid;        /* эффективный GID владельца */
    uid_t    cuid;       /* эффективный UID создателя */
    gid_t    cgid;       /* эффективный GID создателя */
    unsigned short mode;       /* права */
unsigned short __seq; /*внутренний идентификатор*/
        };
```

Функции для работы с очередью сообщений POSIX

Метод	**Описание**
mq_open	Открытие/создание очереди сообщений.
mq_close	Закрытие очереди сообщений.
mq_getattr	Получение атрибутов очереди сообщений.
mq_setattr	Установка атрибутов очереди сообщений.
mq_send	Отправка сообщения в очередь.
mq_receive	Приём сообщения из очереди.
mq_timedsend	Отправка сообщения в очередь. Блокируется в течение заданного времени.
mq_timedreceive	Приём сообщения из очереди. Блокируется в течение заданного времени.
mq_notify	Регистрация для получения уведомления всякий раз, когда получено сообщение в пустой очереди сообщений.
mq_unlink	Удаление очереди сообщений.

Использование функций mq_timedsend и mq_timedreceive

```
/* Отправка сообщения */
struct timespec ts;

/* С этого момента указывает время ожидания как 10 с. */
ts.tv_sec = time(NULL) + 10;
ts.tv_nsec = 0;
if (mq_timedsend(ds, text, SIZE,PRIOTITY, &ts) == -1){
  if (errno == ETIMEDOUT){
    printf(«Очередь заполнена.");
    return 0;
  }
  return -1;
}

/* Приём сообщения */
if (mq_timedreceive(ds, new_text, SIZE, &prio, &ts) == -1){
  if (errno == ETIMEDOUT){
    printf(«Очередь пуста.");
    return 0;
  }
  return -1;
}
```

ПРОГРАММА СЕРВЕР - НА ЗАПРОС КЛИЕНТА ВЫДАЕТ ДАТУ И ВРЕМЯ ЗАВЕРШАЕТ РАБОТУ ПО СИГНАЛУ

```
int id_server;  /* идентификатор очереди сервера */
/* перехватывающая функция для сигнала SIGINT */
 void end_server(int sig) {
     msgctl(id_server,IPC_RMID,0);  /* уничтожение очереди */
     raise(SIGKILL);  /* уничтожение процесса */   }
main()
 {
    struct msg_client {   /* сообщение клиента */
       long id_client;  /* идентификатор процесса клиента */
       int port_client;   /* идентификатор очереди клиента */
     }msg_cl;
    struct msg_time {   /* сообщение клиенту */
        long pr_server;  /* идентификатор процесса сервера */
        time_t vtime;  /* дата и время */
     } msg_t ;
   id_server=msgget(121,0622 |  IPC_CREAT); /* создание очереди */
   signal(SIGINT,end_server);  /* перехват сигнала SIGINT*/
   for (;;) {  /* цикл работы сервера, выход по сигналу SIGINT */
      msgrcv(id_server, &msg_cl,sizeof(int),0,0);  /* прием сообщения от клиента */
 /* вывод информации  о клиенте */
      printf("idn_client= %d id_port_client= %d\n", msg_cl.id_client,msg_cl.port_client);
/* формирование сообщения клиенту */
       msg_t.pr_server=getpid();     /* идентификатор сервера */
       msg_t.vtime=time(0);          /* дата и время */
    msgsnd(msg_cl.port_client,&msg_t,sizeof(time_t),0); /* посылка сообщения  клиенту */
     }
  }
```

ПРОГРАММА КЛИЕНТ ЗАПРАШИВАЕТ У СЕРВЕРА ДАТУ И ВРЕМЯ, ЕСЛИ ЕСТЬ ВХОДНОЙ ПАРАМЕТР, ТО ПЕРЕДАЕТ СИГНАЛ ЗАВЕРШЕНИЯ РАБОТЫ СЕРВЕРА

```
main(int argc, char** argv)
 {
    int id_port_server;  /* идентификатор очереди сервера */
    int id_port_cl;       /* идентификатор очереди клиента */
    struct msg_client {      /* сообщение серверу */
        long id_client;       /* идентификатор процесса клиента */
         int port_client;      /* идентификатор очереди клиента */
    }  msg_server;
    struct msg_time {       /* сообщение от сервера */
          long id_server;     /* идентификатор процесса сервера */
          time_t vtime[1];    /* дата и время */
   } msg_t ;
    id_port_server=msgget(121, IPC_CREAT);  /* связь с очередью сервера */
    id_port_cl=msgget(IPC_PRIVATE,0622);  /* выделение очереди для клиента */
    /* формирование сообщения серверу */
    msg_server.id_client=getpid();   /* идентификатор процесса клиента */
    msg_server.port_client=id_port_cl;  /* идентификатор порта клиента */
  msgsnd(id_port_server,&msg_server,sizeof(int),0); /* посылка сообщения */
    msgrcv(id_port_cl,&msg_t,sizeof(time_t),0,0); /*прием сообщения от сервера*/
    printf("%s\n",ctime(msg_t.vtime));     /* вывод даты и времени */
    msgctl(id_port_cl,IPC_RMID,0);       /* уничтожение очереди клиента */
    printf("%d\n",msg_t.id_server);         /* вывод идентификатора сервера */
    if (argv[1] )
    kill(msg_t.id_server,SIGINT);  /* посылка сигнала завершения серверу */
  }
```

Задание 1 к лабораторной работе 8

Напишите **две программы**, которые обмениваются сообщениями. Первая программа создает очередь и ожидает сообщение от второй программы определенное время, которое задается при запуске первой программы и выводится на экран. Если за это время сообщение от второй программы не поступило, то первая программа завершает свою работу и уничтожает очередь. Вторая программа может запускаться несколько раз и только при условии, что первая программа работает, в противном случае она заканчивает свою работу. При запуске второй программы указывается новое время ожидания для первой программы.

Задание 2 к лабораторной работе 8

Напишите **три программы**, выполняющиеся параллельно и читающие один и тот же файл. Программа, которая хочет прочитать файл, должна передать другим программам запрос на разрешение операции и ожидать их ответа. Эти запросы программы передают через одну очередь сообщений. В запросе указываются: номер программы, которой посылается запрос, идентификатор **локальной** очереди, куда надо передать ответ и время посылки запроса. Начать выполнять операцию чтения файла программе разрешается только при условии получения ответов от двух других программ. Каждая программа перед отображением файла на экране должна вывести следующую информацию: номер программы и времена ответов, полученных от других программ.

Программа, которая получила запрос от другой программы, должна реагировать следующим образом:

- если программа прочитала файл, то сразу передается ответ, он должен содержать номер отвечающей программы и время ответа;
- если файл не читался, то ответ передается только при условии, что время посылки запроса в сообщении меньше, чем время запроса на чтение у данной программы;

Запросы, на которые ответы не были переданы, должны быть запомнены и после чтения файла обслужены.