Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №6-8 по курсу**

**«Операционные системы»**

**Сервер очереди сообщений**

Студент: Гаврилов Максим Сергеевич

Группа: М8О–206Б–20

Преподаватель: Соколов Андрей Алексеевич

Оценка: \_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_

Москва, 2020.

**Постановка задачи**

**Цель курсового проекта**

1. Приобретение практических навыков в использовании знаний, полученных в течении курса
2. Проведение исследования в выбранной предметной области

**Задание**

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа.

**Условия задачи**

Проект должен содержать 2 части:  
Библиотека по работе с сервером, которую можно подключить к любой программе на языке С. Основные функции библиотеки:  
   var connection = Connect(string serverAddress); // var connection = Connect(«127.0.0.1:8312»);  
   var queue = CreateQueue(Name, isLogged) // isLogged = true - должна создаваться логирующая очреедь для основной. В нее должны помещаться все сообщения, которые помещаются в основную очередь. Для каждой очереди должна быть возможность создать свою логирующую очередь. Размер логирущей очереди должен указываться в конфигурационном файле сервера. При переполнении очереди - удаляются старые сообщения.  
   SendMessage(connection, queue, message) //Необходимо в сообщение передавать еще приоритет  
   RecieveMessage(connection, queue)

Сам сервер сообщений. У него должен быть cli. То есть запуск должен осуществляться следующим образом:  
    server.out 8312 (поднимает сервер сообщений на порту 8312)  
    Сервер должен работать на пайпах

**Общие сведения о программе**

Программа состоит из двух частей – сервера (компилируется из [MSQserver.c](https://github.com/Megict/OS_KP/blob/main/yes%3Dserver/src/MSQserver.c), msgOPR.c, transformres.c) и оператора очереди (компилируется из src/msgOPR.c src/operator.c src/transformres.c). Первый отвечает за соединение с портом с помощью сокета и пересылку запросов с порта оператору. Оператор же осуществляет обработку запросов, отправку и прием сообщений через именованные пайпы и хранение сообщений. Также существует вспомогательная программа retranslator, которая запускается сервером при получении очередного запроса на соединение с портом, и непосредственно выполняет пересылку запросов с сокета, соединенного с портом на входной пайп очереди.

**Сервер** имеет два алгоритма запуска, выбор зависит от содержания config-файла:

1. При запуске создается новый оператор очереди. В таком слкчае в config должна быть прописана максимальная длина логирующей очереди.
2. При запуске сервер присоединяется к уже существующей очереди. В таком случае в config должно быть указано имя входного пайпа оператора.

После запуска сервер выполняет bind указанного аргументом порта и начинает слушать его на предмет запросов на подключение. Если запрос получен, то создается процесс, в котором исполняется retranslator, а созданный командой accept сокет вместе с именем входного пайпа оператора отправляется ретранслятору, как аргумент при запуске. После этого ретранслятор начинает переправлять все сообщения, полученные с входного сокета оператору.

Сообщения, переправляемые ретранслятором, представляют собой **запросы.** Вот их структура:

|  |
| --- |
| **typedef** struct request **{**  int action**;**  //0 - подписаться  //1 - отправить сообщение с localIN подписчику с reciverID  //2 - посмотреть, есть ли для localID сообщения и если есть, переслать их на LocalOUT  //3 - узнать localID процесса по его PID и переслать его на loclOUT  int localID**;** //кто обращается.  int Qid**;** //нужно для отправки сообщений  char localIN**[**9**];** //заранее созданные pipe-ы. Нужны для подписки  char localOUT**[**9**];** //заранее созданные pipe-ы. Нужны для подписки  **}** request**;** //запросы такого вида принимает оператор очереди через publicPort. Дальнейщее взаимодействие идет через локальные порты. |

Получая запрос, оператор выполняет одно из следующих действий:

**Подписка (код 1).** Для последующего взаимодействия с процессом, отправившем запрос понадобятся отдельные каналы. К моменту получения запроса оператором они уже будут созданы клиентскими функциями (эти функции будут рассмотрены позже). Имена этих каналов хранятся в строках localIN и localOUT. Оператор открывает их и помещает их файловые дескрипторы в структуру типа subscriber. Также туда добавляется LocalD процесса-подписчика (по сути, логин. В моем случае используется pid). Структура помещается в массив подписчиков.

**Создание очереди (код 4).** Информация о том, надо ли логировать очередь помещается в первый байт строки localIN (все равно эти строки нужны только для подписки). Индекс (имя) создаваемой очереди находится в поле Qid. Оператор создет очередь функцией queueMASadd или же queueMASaddLogged. Очередь сразу же помещается в массив очередей и с ней можно работать.

**Отправка сообщения в очередь (код 2).** Это запрос-оповещение, сообщающее оператору, что на localIN подписчика localID уже лежит сообщение, которое надо поместить в очередь Qid. Оператор находит открытый файловый дескриптор подписчика localID в массиве подписчиков, считывает с него сообщение и помещает в очередь.

**Извлечение сообщения из очереди (код 3).** Оператор находит очередь Qid в массиве очередей, проверят, есть ли в ней сообщения, если есть, достает последнее и отправляет его на localOUT подписчика localID. Иначе туда будет отправлено сообщение о его отсутствии.

**Клиентская библиотека**

Основные функции в ней:

|  |
| --- |
| int connectToQ**(**msQ**\*** queue**,**char**\*** ip**,**int port**);**  int subscribe**(**msQ**\*** queue**);** //во входной структуре должен быть прописан публичный порт. Локальные порты будут записаны после получения  int makeQ**(**msQ**\*** queue**,**int QID**,** int doLog**);**  int sendMSG**(**msQ**\*** queue**,** msg**\*** item**,** int senderLid**,** int adressateLid**);**  int getMSG**(**msQ**\*** queue**,** msg**\*** item**,** int senderLid**,** int adressateLid**);** |

Структура msQ содержит в себе файловые дескриптор основного ввода очереди (на который надо посылать запросы), а также дескриптор пайпов localIN и LocalOUT, через которые отправляются и получаются сообщения.

Функция connectToQ записывает полуденный дескриптор (сокет, который соединяется с портом, на котором развернут сервер) в поле основного ввода очереди. Остальные функции выполняют следующее:

1. Формируют запрос, соответствующий запрашиваемому действию.
2. Если необходимо, записывают отправляемое сообщение в LocaIN
3. Отправляют запрос в PublicPort
4. Если необходимо, считывают ответ с LocalOUT

**Логирование.**

Структура очереди содержит в себе указатель на еще одну очередь:

|  |
| --- |
| **typedef** struct msgQ msgQ**;**  **typedef** struct msgQ **{** //очередь сообщений  msg**\*\*** Elem**;**  int ListSize**;**  int ListCapacity**;**  int queueID**;** //идентефикатор очереди  char logFlag**;**  int logLen**;**  msgQ**\*** log**;**  **}** msgQ**;** |

Если оператор получил запрос на создание логированной очереди, то вместо функции queueMASadd используется функция queueMASaddLogged. Если очередь логируемая, то при ее создании создаётся еще одна очередь, указатель на нее помещается в log, а максимальная длина логирующей очереди помещается в logLen. При добавлении элемента в очередь он также добавляется в логирующую очередь. Если длина логирующей очереди больше logLen, то первый из добавленных в нее элементов удаляется.

В текущей реализации логирующие очереди переписываются в файл только при завершении работы оператора.

**Приоритет сообщений.**

Сообщения имеют следующую структуру:

|  |
| --- |
| **typedef** struct msg **{**  int priority**;**  int size**;**  char**\*** content**;**  **}** msg**;** |

При добавлении элемента в очередь сообщений приоритет добавляемого сообщения сравнивается с приоритетом последнего сообщения в очереди. Если приоритет добавляемого выше приоритета последнего, то позиция добавляения сдвигается к началу очереди, пока приоритеты добавляемого и элемента на позиции до него не уравняются (при извлечении элемент берется из начала).

Таким образом, приоритет сообщений обрабатывается на самом низком уровне, что позволяет оператору вообще не вникать в структуру передаваемых сообщений и делает его более универсальным.

**О критических участках.**

Их здесь нет. Одной из основных целей в своей работе я поставил себе избавление от необходимости применения мьютекса. Именно этой цели служит то что оператор использует для пересылки сообщений уникальные для каждого подписчика каналы. Таким образом, после получения запроса оператор работает только с локальными каналами одного процесса, отчего риск прочитать или переслать что-то, предназначающееся другому процессу, сводится к нулю. На мой взгляд, для данной архитектуры оператора очереди, это оптимальное решение. Если бы оператор был многопоточным, и единственным общим объектом для всех его потоков была бы единая очередь, то это было бы эффективнее за счет одновременной обработки нескольких запросов, однако это уже была бы принципиально иная программа, не имеющая с моей почти ничего общего.

**Используемые системные вызовы**

Основные системные объекты, которые используемые в моей программе – сокеты и именованные пайпы.

Для работы с сокетами я использовал следующие команды:  
**int socket(int***domain***, int***type***, int***protocol***);**  **–** создание сокета

**int bind(int***sockfd***, struct sockaddr \****my\_addr***, socklen\_t***addrlen***);**  **–** привязывание к сокету локальный адрес my\_addr

**int listen(int***s***, int***backlog***);**   **–** определяет готовность сервера принимать соединения

**int accept(int***s***, struct sockaddr \****addr***, socklen\_t \****addrlen***);**   **–** извлекает из очереди запросов на соединение первый запрос и создает новый подключенный сокет.

**int connect(int***sockfd***, const struct sockaddr \****serv\_addr***,** **socklen\_t***addrlen***);**   **–** посылает запрос на соединение на адрес serv\_addr.

Команды для работы с именованными пайпами:

**int mkfifo(const char \****pathname***, mode\_t** *mode***); –** создает именованный канал. После создания с ним можно работать как с обычным файлом.

**int open(const char \****pathname***, int** *flags***); –** открыть файл. При работе с fifo-файлом необходимо, чтобы он был открыт и на чтение и на запись.

**int close(int** *fd***); –** закрыть файловый дескриптор

Также, разумеется, я использовал стандартные вызовы **read** и **write**.

При реализации сервера мне потребовалась многопоточность. Для ее реализации прменяем fork и execl:

**pid\_t fork(void);** – создает процесс-потомок, отсличающийся от родительского только значением pid.

**int execl(const char \****path***, const char \****arg***, ...); –** заменяет текущий образ проесса новым (программой, заданой параметорм path.

**Основные файлы программы**

**msgOPR.c**

Основная библиотека с функциями для работы с очередью сообщений. В начале идут функции для работы с единичными сообщениями, затем – с очередью, с массивом очередей, с массивом подписчиков и, наконец, клиентские функции, которые подключаются программой, для взаимодействия с оператором. Сигнатуры функций разнесены по разным хедерам, но библиотека подключается всегда одна и та же.

|  |
| --- |
| #include <string.h>  #include <unistd.h>  #include <stdlib.h>  #include <fcntl.h>  #include <math.h>  #include <signal.h>  #include <time.h>  #include <strings.h>  #include "transformres.h"  #include <sys/types.h>  #include <sys/stat.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <arpa/inet.h>  #include <stdio.h>  **typedef** struct msg **{**  int priority**;**  int size**;**  char**\*** content**;**  **}** msg**;**  msg**\*** createMsg**(**int \_priority**,** char**\*** \_content**)** **{**  **if(**\_priority **<** 0**)** **{**  **return** **NULL;**  **}**  msg**\*** mess **=** **(**msg**\*)**malloc**(sizeof(**msg**));**  mess**->**priority **=** \_priority**;**  mess**->**size **=** strlen**(**\_content**)** **+** 1**;**  mess**->**content **=** **(**char**\*)**malloc**(**mess**->**size**);**  mess**->**content **=** strcpy**(**mess**->**content**,** \_content**);**  **return** mess**;**  **}**  void destroyMSG**(**msg**\*** dbl**)** **{**  free**(**dbl**->**content**);**  free**(**dbl**);**  **}**  **typedef** struct msgQ msgQ**;**  **typedef** struct msgQ **{**  msg**\*\*** Elem**;**  int ListSize**;**  int ListCapacity**;**  int queueID**;**  char logFlag**;**  int logLen**;**  msgQ**\*** log**;**  **}** msgQ**;**  void Qinit **(**msgQ**\*** queue**,**int ID**)** **{**  queue**->**Elem **=** **NULL;**  queue**->**ListSize **=** 0**;**  queue**->**ListCapacity **=** 0**;**  queue**->**queueID **=** ID**;**  queue**->**logFlag **=** 0**;**  **}**  void QinitLogged **(**msgQ**\*** queue**,**int ID**,** int logLen**)** **{** //очередь логируется. В очереди - журнале хранятся все сообения, что когда-либо были в нее помещены  queue**->**Elem **=** **NULL;**  queue**->**ListSize **=** 0**;**  queue**->**ListCapacity **=** 0**;**  queue**->**queueID **=** ID**;**  queue**->**logFlag **=** 1**;**  queue**->**logLen **=** logLen**+**1**;**  queue**->**log **=** **(**msgQ**\*)**malloc**(sizeof(**msgQ**));**  Qinit **(**queue**->**log**,**0**);**  **}**  int Qadd **(**msgQ**\*** Li**,** msg**\*** adb**)** **{**  **if(**Li**->**ListSize **==** 0**)** **{**  Li**->**Elem **=** **(**msg**\*\*)**realloc**(**Li**->**Elem**,sizeof(**msg**\*));**  Li**->**ListCapacity **=** 1**;**  **}**  **if(**Li**->**ListSize **==** Li**->**ListCapacity**)** **{**  Li**->**ListCapacity**\*=**2**;**  Li**->**Elem **=** **(**msg**\*\*)**realloc**(**Li**->**Elem**,sizeof(**msg**\*)\***Li**->**ListCapacity**);**  **}**    int pos **=** Li**->**ListSize**;**  **for(**int i **=** Li**->**ListSize **-** 1**;**i **>=** 0**;--**i**)** **{**  **if(**Li**->**Elem**[**i**]->**priority **<** adb**->**priority**)** **{**  Li**->**Elem**[**i**+**1**]** **=** Li**->**Elem**[**i**];**  pos**--;**  **}**  **}**  Li**->**Elem**[**pos**]** **=** adb**;**  Li**->**ListSize**++;**  **if(**Li**->**logFlag **==** 1**)** **{**  Qadd**(**Li**->**log**,** adb**);**  **}**  **return** 1**;**  **}**  int Qremove **(**msgQ**\*** Li**,**int rmInd**)** **{**  **if(**Li**->**ListSize **==** 0**)** **{**  **return** 0**;**  **}**  **for(**int i**=**rmInd**;**i**<**Li**->**ListSize**-**1**;++**i**)** **{**  Li**->**Elem**[**i**]** **=** Li**->**Elem**[**i**+**1**];**  **}**  Li**->**ListSize**--;**  **if(**Li**->**ListSize **\***2 **>** Li**->**ListCapacity**)** **{**//при таком удалении теряется указатель на узел !исправить!  Li**->**ListCapacity**/=**2**;**  Li**->**Elem **=** **(**msg**\*\*)**realloc**(**Li**->**Elem**,sizeof(**msg**\*)\***Li**->**ListCapacity**);**  **}**  **return** 0**;**  **}**  msg**\*** QgetPointer**(**msgQ**\*** Li**,**int ind**)** **{**  **if(**Li**->**ListSize **<=** ind**){**  **return** **NULL;**  **}**  **else** **{**  **return** Li**->**Elem**[**ind**];**  **}**  **}**  msg**\*** Qget **(**msgQ**\*** Li**)** **{**//вытаскивает из очереди первый элемент, возвращает его  msg**\*** gtb **=** QgetPointer**(**Li**,**0**);**  Qremove**(**Li**,**0**);**  **if(**Li**->**logFlag **==** 1 **&&** Li**->**log**->**ListSize **>=** Li**->**logLen**)** **{** //убираем первыйй элемент из очереди-лога, если ее длина превышает лимит. (доавляются элементы в конец)  Qremove**(**Li**->**log**,**0**);**  **}**  **return** gtb**;**  **}**  void Qdestroy**(**msgQ**\*** Li**)** **{**  **if(**Li**->**logFlag **==** 0**)** **{**  **for(**int i **=** 0**;** i **<** Li**->**ListSize**;** **++**i**)** **{**  destroyMSG**(**Li**->**Elem**[**i**]);**  **}**  **}**  **else** **{**  Qdestroy**(**Li**->**log**);**  free**(**Li**->**log**);**  **}**  free**(**Li**->**Elem**);**  **}**  **typedef** struct Qmas **{**//массив очередей сообщений  msgQ**\*\*** queues**;**  int size**;**  **}** Qmas**;**  void queueMASinit**(**Qmas**\*** mas**)** **{**  mas**->**queues **=** **NULL;**  mas**->**size **=** 0**;**  **}**  int findQ**(**Qmas**\*** mas**,**int adressate**)** **{** // возвращает порядковый номер очереди с указанным адресатом в массиве  **for(**int i**=**0**;**i**<**mas**->**size**;++**i**)** **{**  **if(**mas**->**queues**[**i**]->**queueID **==** adressate**)** **{**  **return** i**;**  **}**  **}**  **return** **-**1**;**  **}**  msgQ**\*** queueMASlocate**(**Qmas**\*** mas**,**int ID**)** **{** //возвращает указатель на очередь с ID в массиве mas  **for(**int i**=**0**;**i**<**mas**->**size**;++**i**)** **{**  **if(**mas**->**queues**[**i**]->**queueID **==** ID**)** **{**  **return** mas**->**queues**[**i**];**  **}**  **}**  **return** **NULL;**  **}**  int queueMASadd**(**Qmas**\*** mas**,**int ID**)** **{**//добавляет в массив очередь с индексом  mas**->**size**++;**  mas**->**queues **=** **(**msgQ**\*\*)**realloc**(**mas**->**queues**,sizeof(**msgQ**\*)\***mas**->**size**);**  mas**->**queues**[**mas**->**size**-**1**]** **=** **(**msgQ**\*)**malloc**(sizeof(**msgQ**));**  Qinit**(**mas**->**queues**[**mas**->**size**-**1**],**ID**);**  **return** mas**->**size**-**1**;**  **}**  int queueMASaddLogged**(**Qmas**\*** mas**,**int ID**,** int LogMAX**)** **{**//добавляет в массив логируемую очередь  mas**->**size**++;**  mas**->**queues **=** **(**msgQ**\*\*)**realloc**(**mas**->**queues**,sizeof(**msgQ**\*)\***mas**->**size**);**  mas**->**queues**[**mas**->**size**-**1**]** **=** **(**msgQ**\*)**malloc**(sizeof(**msgQ**));**  QinitLogged**(**mas**->**queues**[**mas**->**size**-**1**],**ID**,**LogMAX**);**  **return** mas**->**size**-**1**;**  **}**  void destroyMAS**(**Qmas**\*** mas**)** **{**  **for(**int i**=**0**;** i**<**mas**->**size**;++**i**)** **{**  Qdestroy**(**mas**->**queues**[**i**]);**  free**(**mas**->**queues**[**i**]);**  **}**  free**(**mas**->**queues**);**  **}**  **typedef** struct subscriber **{**  int localID**;**  int localPortIn**;** //откуда очеердь будет читать сообщения  int localPortOut**;** //куда очередь будет писать сообщенияы  **}** subscriber**;**  subscriber**\*** createSUB **(**int lid**,**int in**,** int out**)** **{**  subscriber**\*** crt **=** **(**subscriber**\*)**malloc**(sizeof(**subscriber**));**  crt**->**localID **=** lid**;**  crt**->**localPortIn **=** in**;**  crt**->**localPortOut **=** out**;**  **return** crt**;**  **}**  **typedef** struct SUBlist **{** //список подписсчиков  subscriber**\*\*** Elem**;**  int ListSize**;**  int ListCapacity**;**  **}** SUBlist**;**  void subLISTinit **(**SUBlist**\*** list**)** **{**  list**->**Elem **=** **NULL;**  list**->**ListSize **=** 0**;**  list**->**ListCapacity **=** 0**;**  **}**  int subLISTadd **(**SUBlist**\*** Li**,** subscriber**\*** adb**)** **{**  **if(**Li**->**ListSize **==** 0**)** **{**  Li**->**Elem **=** **(**subscriber**\*\*)**malloc**(sizeof(**subscriber**\*));**  Li**->**ListCapacity **=** 1**;**  **}**  **if(**Li**->**ListSize **==** Li**->**ListCapacity**)** **{**  Li**->**ListCapacity**\*=**2**;**  Li**->**Elem **=** **(**subscriber**\*\*)**realloc**(**Li**->**Elem**,sizeof(**subscriber**\*)\***Li**->**ListCapacity**);**  **}**  Li**->**Elem**[**Li**->**ListSize**]** **=** adb**;**  Li**->**ListSize**++;**  **return** 1**;**  **}**  subscriber**\*** subLISTlocate **(**SUBlist**\*** Li**,** int fdbID**)** **{**  **for(**int i**=**0**;**i**<**Li**->**ListSize**;++**i**)** **{**  **if(**Li**->**Elem**[**i**]->**localID**==**fdbID**)** **{**  **return** Li**->**Elem**[**i**];**  **}**  **}**  **return** **NULL;**  **}**  void destroySUBlist **(**SUBlist**\*** Li**)** **{**  **for(**int i**=**0**;**i**<**Li**->**ListSize**;++**i**)** **{**  close**(**Li**->**Elem**[**i**]->**localPortIn**);**  close**(**Li**->**Elem**[**i**]->**localPortOut**);**  free**(**Li**->**Elem**[**i**]);**  **}**  free**(**Li**->**Elem**);**  **}**  **typedef** struct request **{**  int action**;**  //0 - подписаться  //1 - отправить сообщение с localIN подписчику с reciverID  //2 - посмотреть, есть ли для localID сообщения и если есть, переслать их на LocalOUT  //3 - узнать localID процесса по его PID и переслать его на loclOUT  int localID**;** //кто обращается.  int Qid**;** //нужно для отправки сообщений  char localIN**[**9**];** //заранее созданные pipe-ы. Нужны для подписки  char localOUT**[**9**];** //заранее созданные pipe-ы. Нужны для подписки  **}** request**;** //запросы такого вида принимает оператор очереди через publicPort. Дальнейщее взаимодействие идет через локальные порты.  **typedef** struct msQ **{**  int publicPort**;** //куда уведомлять оператор  int localPortIn**;** //куда посылать сообщения оператору очереди  int localPortOut**;** //откуда принимать сообщения  **}** msQ**;** //структура, которую получает процесс после подписки на очередь.  char**\*** nameGenerator**(**int seed**)** **{**  srand**(**time**(NULL)\***seed**);**  char**\*** ret **=** **(**char**\*)**malloc**(**4**);**  **for(**int i **=** 0**;** i **<** 4**;** **++**i**)** **{**  ret**[**i**]** **=** rand**()%**20 **+** 'a'**;**  **}**  char nst**[**9**]** **=** "tmp/"**;**  strncat**(**nst**,**ret**,**4**);**  int k **=** open**(**nst**,**O\_WRONLY**);**  **while(**k **!=** **-**1**)** **{**  close**(**k**);**  **for(**int i **=** 0**;** i **<** 4**;** **++**i**)** **{**  ret**[**i**]** **=** rand**()%**20 **+** 'a'**;**  **}**  char nst**[**9**]** **=** "tmp/"**;**  strncat**(**nst**,**nameGenerator**(**1**),**4**);**  k **=** open**(**nst**,**O\_WRONLY**);**  **}**  **return** ret**;**  **}**  int CreateQueue**(**msQ**\*** queue**,** int maxlen**)** **{** //принимается указатель на массив из 2-х элементов. Процесс, создавший оператор получает его публичные порты. Возвращается PID оператора  //fd[0] - общий дескриптор для отправки сообщений оператору.  // int fd1[2];  // pipe(fd1);  // fd[0] = fd1[1];  char**\*** globalFIF**;**  char nst**[**9**]** **=** "tmp/"**;**  char**\*** gen **=** nameGenerator**(**3**\***getpid**());**  strncat**(**nst**,**gen**,**4**);**  free**(**gen**);**  mkfifo**(**nst**,** 0777**);**  globalFIF **=** **(**char**\*)**malloc**(sizeof(**nst**));**  strcpy**(**globalFIF**,**nst**);**  int queueOprID **=** fork**();** //запуск процесса оператора очереди  **if(**queueOprID **==** **-**1**)** **{**  **return** **-**1**;**  **}**  **else**  **if(**queueOprID **==** 0**)** **{**  execl**(**"operator.exe"**,**"operator.exe"**,**globalFIF**,**transform**(**maxlen**),NULL);**  free**(**globalFIF**);**  **return** 0**;**  **}**  **else** **{**  queue**->**publicPort **=** open**(**globalFIF**,**O\_WRONLY**);**  free**(**globalFIF**);**  **return** queueOprID**;**  **}**  **}**  int connectToQ**(**msQ**\*** queue**,**char**\*** ip**,**int port**)** **{**  int sockfd **=** socket**(**AF\_INET**,** SOCK\_STREAM**,** 0**);**  **if(**sockfd **==** **-**1**)** **{**  **return** **-**1**;**  **}**    struct sockaddr\_in adr**;**  bzero**((**char **\*)** **&**adr**,** **sizeof(**adr**));**  //if(inet\_pton(AF\_INET,ip,&adr.sin\_addr.s\_addr) < 0)  adr**.**sin\_addr**.**s\_addr **=** htonl**(**INADDR\_LOOPBACK**);**  adr**.**sin\_port **=** htons**(**port**);**  adr**.**sin\_family **=** AF\_INET**;**  **if(**connect**(**sockfd**,** **(**struct sockaddr **\*)&**adr**,** **sizeof(**adr**))** **<** 0**)** **{**  **return** **-**2**;**  **}**  **else** **{**  queue**->**publicPort **=** sockfd**;**  **return** 0**;**  **}**  **}**  int subscribe**(**msQ**\*** queue**)** **{** //во входной структуре должен быть прописан публичный порт. Локальные порты будут записаны после получения  //возвращаемое значение - код успеха  //после отправки запроса по дескриптору 0 необходимо уведомить оператор сообщением на глобальный дескриптор.  char**\*** fif1**;**  char**\*** fif2**;**  char nst**[**9**]** **=** "tmp/"**;**  char**\*** gen **=** nameGenerator**(**1**\***getpid**());**  strncat**(**nst**,**gen**,**4**);**  free**(**gen**);**  mkfifo**(**nst**,** 0777**);**  fif1 **=** **(**char**\*)**malloc**(sizeof(**nst**));**  strcpy**(**fif1**,**nst**);**  strcpy**(**nst**,**"tmp/"**);**  gen **=** nameGenerator**(**2**\***getpid**());**  strncat**(**nst**,**gen**,**4**);**  free**(**gen**);**  mkfifo**(**nst**,** 0777**);**  fif2 **=** **(**char**\*)**malloc**(sizeof(**nst**));**  strcpy**(**fif2**,**nst**);**  printf**(**"<%s|%s>\n"**,**fif1**,**fif2**);**  request qCall**;** //эта структура будет отправлена оператору  qCall**.**action **=** 1**;**  qCall**.**localID **=** getpid**();**  qCall**.**Qid**=**0**;**  printf**(**"<%s|%s>\n"**,**fif1**,**fif2**);**  strcpy**(**qCall**.**localIN**,** fif1**);** //это на чтение  strcpy**(**qCall**.**localOUT**,** fif2**);** //это открыть на запись  //write(fd2[1],"1",sizeof("1"));  //printf("sub: sent (%d)\n",qCall.localOUT);  write**(**queue**->**publicPort**,&**qCall**,sizeof(**qCall**));**  queue**->**localPortIn **=** open**(**fif1**,**O\_WRONLY**);**  queue**->**localPortOut **=** open**(**fif2**,**O\_RDONLY**);**  free**(**fif1**);**  free**(**fif2**);**  int lid**;**    read**(**queue**->**localPortOut**,&**lid**,sizeof(**int**));**  **return** lid**;**  //сформировать запрос на подписку  //считать с выходного дескриптора сигнал об успешной подписке  //вернуть localID?  **}**  int makeQ**(**msQ**\*** queue**,**int QID**,** int logged**)** **{**  request qCall**;** //эта структура будет отправлена оператору  qCall**.**action **=** 4**;**  qCall**.**localID **=** getpid**();**  qCall**.**Qid **=** QID**;**  **if(**logged **==** 0**)** **{**  strcpy**(**qCall**.**localIN**,**"00000000"**);** //далеко не только чтобы valgrind не ругался  **}**  **else** **{**  strcpy**(**qCall**.**localIN**,**"00000000"**);** //далеко не только чтобы valgrind не ругался  qCall**.**localIN**[**0**]** **=** 1**;**  **}**  strcpy**(**qCall**.**localOUT**,**"00000000"**);** //но он все равно ругается (  write**(**queue**->**publicPort**,&**qCall**,sizeof(**qCall**));**  int resp **=** **-**1**;**  read**(**queue**->**localPortOut**,&**resp**,sizeof(**int**));**  **return** resp**;**  //-1 - чето прям ваще не так  //0 - очередь создана  //1 - такой идентефикатор уже есть  **}**  int sendMSG**(**msQ**\*** queue**,** msg**\*** item**,** int senderLid**,** int QID**)** **{**  request qCall**;** //эта структура будет отправлена оператору  qCall**.**action **=** 2**;**  qCall**.**localID **=** senderLid**;**  qCall**.**Qid **=** QID**;**  strcpy**(**qCall**.**localIN**,**"00000000"**);** //исключительно чтобы valgrind не ругался  strcpy**(**qCall**.**localOUT**,**"00000000"**);**  //write(queue->localPortIn,item,sizeof(msg)); //отправляем сообщение на localIN очереди  write**(**queue**->**localPortIn**,&**item**->**priority**,sizeof(**int**));**  write**(**queue**->**localPortIn**,&**item**->**size**,sizeof(**int**));**  write**(**queue**->**localPortIn**,**item**->**content**,**item**->**size**);**  write**(**queue**->**publicPort**,&**qCall**,sizeof(**qCall**));** //отправляем уведомление очереди о том, что надо переслать сообщение  **return** 0**;**  **}**  int getMSG**(**msQ**\*** queue**,** msg**\*** item**,** int senderLid**,** int QID**)** **{**  request qCall**;** //эта структура будет отправлена оператору  qCall**.**action **=** 3**;**  qCall**.**localID **=** senderLid**;**  qCall**.**Qid **=** QID**;**  strcpy**(**qCall**.**localIN**,**"00000000"**);** //исключительно чтобы valgrind не ругался  strcpy**(**qCall**.**localOUT**,**"00000000"**);**  write**(**queue**->**publicPort**,&**qCall**,sizeof(**qCall**));** //отправляем уведомление очереди о том, что надо проверить наличие сообщений  //read(queue->localPortOut,item,sizeof(msg)); //считываем сообщение с localOUT очереди  read**(**queue**->**localPortOut**,&**item**->**priority**,sizeof(**int**));**  **if(**item**->**priority **==** **-**1**)** **{** //значит, собщения нету  **return** **-**1**;**  **}**  read**(**queue**->**localPortOut**,&**item**->**size**,sizeof(**int**));**  item**->**content **=** **(**char**\*)**realloc**(**item**->**content**,**item**->**size**);**  read**(**queue**->**localPortOut**,**item**->**content**,**item**->**size**);**  **return** 0**;**  **}** |

**Operator.c** Оператор очереди сообщений. Работает полностью на пайпах. За его подключение к порту отвечает сервер.

|  |
| --- |
| #include <string.h>  #include <unistd.h>  #include <stdlib.h>  #include <fcntl.h>  #include <math.h>  #include <signal.h>  #include "transformres.h"  #include <sys/types.h>  #include <sys/socket.h>  #include <sys/stat.h>  #include <netinet/in.h>  #include <stdio.h>  #include "msgOPR.h"  #define MAP\_FILE\_SIZE 100  #define DEBUG 1  int MAXLOGLEN **=** 0**;**  Qmas allQUEUES**;**  SUBlist subscribers**;**  char**\*** nameIN **=** **NULL;**  char**\*** nameOUT **=** **NULL;**  int printLogs**(**Qmas**\*** queues**,** char**\*** filename**)** **{**  FILE**\*** fdFILE **=** fopen**(**filename**,**"w+"**);**  **if(**fdFILE **==** **NULL)** **{**  **return** **-**1**;**  **}**  fprintf**(**fdFILE**,**"-----------\n"**);**  **for(**int i **=** 0**;**i**<**queues**->**size**;++**i**)** **{**  **if(**queues**->**queues**[**i**]->**logFlag **==** 1**)** **{**  fprintf**(**fdFILE**,**"[%d]: "**,**queues**->**queues**[**i**]->**queueID**);**  **for(**int j**=**0**;**j**<**queues**->**queues**[**i**]->**log**->**ListSize**;++**j**)** **{**  fprintf**(**fdFILE**,**"%s "**,**queues**->**queues**[**i**]->**log**->**Elem**[**j**]->**content**);**  **}**  fprintf**(**fdFILE**,**"\n"**);**  **}**  **}**  fprintf**(**fdFILE**,**"-----------\n"**);**  fclose**(**fdFILE**);**  **return** 0**;**  **}**  void taghdl**(**int sig**)**  **{**  **if(**sig **==** SIGTERM **||** sig **==** SIGINT**)**  **{**  printLogs**(&**allQUEUES**,**"logfile.txt"**);**  destroyMAS**(&**allQUEUES**);**  destroySUBlist**(&**subscribers**);**  **if(**nameIN **!=** **NULL)** **{**  free**(**nameIN**);**  **}**  **if(**nameOUT **!=** **NULL)** **{**  free**(**nameOUT**);**  **}**  system**(**"rm tmp/\*"**);**  exit**(**0**);**  **}**  **}**  int main**(**int argc**,**char**\*\*** argv**)** **{**  struct sigaction act\_sync**;**  memset**(&**act\_sync**,** 0**,** **sizeof(**act\_sync**));**  act\_sync**.**sa\_handler **=** taghdl**;**  sigset\_t set\_sync**;**  sigemptyset**(&**set\_sync**);**  sigaddset**(&**set\_sync**,** SIGTERM**);**  sigaddset**(&**set\_sync**,** SIGINT**);**  act\_sync**.**sa\_mask **=** set\_sync**;**  sigaction**(**SIGTERM**,** **&**act\_sync**,** **NULL);**  sigaction**(**SIGINT**,** **&**act\_sync**,** **NULL);**    //configurate();  **if(**argc **<** 3**)** **{**  **return** 2**;**  **}**  MAXLOGLEN **=** back\_transform**(**argv**[**2**]);**  **if(**DEBUG **==** 1**)**printf**(**"operator activatrd |%d|\n"**,**MAXLOGLEN**);**  **if(**argc **<** 2**)** **{**  **return** **-**3**;**  **}**  int public\_port\_in **=** open**(**argv**[**1**],**O\_RDONLY**);**  **if(**public\_port\_in **==** **-**1**)** **{**  //printf("made\n");  **if(**mkfifo**(**argv**[**1**],** 0777**)** **==** **-**1**)** **{**  **return** **-**1**;**  **}**  **else** **{**  public\_port\_in **=** open**(**argv**[**1**],**O\_RDONLY**);**  **}**  **}**  //на публичные порты отправляется запрос о подписке.  //также на публичный порт отправляются сигнализации о запросе передачи  queueMASinit**(&**allQUEUES**);**  subLISTinit**(&**subscribers**);**  nameIN **=** **(**char**\*)**malloc**(**9**);**  nameOUT **=** **(**char**\*)**malloc**(**9**);**  **for(;;)** **{**  request cur\_ask**;**  char a**[**2**];**    //read(public\_port\_in,&cur\_ask,sizeof(cur\_ask));  **while(**read**(**public\_port\_in**,&**cur\_ask**.**action**,sizeof(**int**))** **==** 0**)** **{}**  read**(**public\_port\_in**,&**cur\_ask**.**localID**,sizeof(**int**));**  read**(**public\_port\_in**,&**cur\_ask**.**Qid**,sizeof(**int**));**  read**(**public\_port\_in**,**nameIN**,**9**);**  read**(**public\_port\_in**,**nameOUT**,**9**);**  read**(**public\_port\_in**,&**a**,**2**);**    **if(**DEBUG**)**printf**(**"operator| got ask | from: %d\n"**,**cur\_ask**.**localID**);**  **if(**DEBUG**)** **{**  **for(**int i**=**0**;**i**<**allQUEUES**.**size**;++**i**)** **{**  printf**(**"|%d|-|"**,**allQUEUES**.**queues**[**i**]->**queueID**);**  **for(**int j**=**0**;**j**<**allQUEUES**.**queues**[**i**]->**ListSize**;++**j**)** **{**  printf**(**"%s|"**,**allQUEUES**.**queues**[**i**]->**Elem**[**j**]->**content**);**  **}**  printf**(**"(%d)\n"**,**allQUEUES**.**queues**[**i**]->**ListSize**);**  **}**  **}**  **if(**DEBUG **==** 1**)**printf**(**"operator ask: %d | %d\n"**,**cur\_ask**.**action**,**cur\_ask**.**Qid**);**    **if(**cur\_ask**.**action **==** 1**)** **{** //подписка  //printf("%d\n",cur\_ask.localIN);  int fdR **=** open**(**nameIN**,**O\_RDONLY**);**  int fdW **=** open**(**nameOUT**,**O\_WRONLY**);**  subLISTadd**(&**subscribers**,**createSUB**(**cur\_ask**.**localID**,**fdR**,**fdW**));**  **if(**DEBUG**)**printf**(**"\tsub added\n"**);**  write**(**fdW**,&**cur\_ask**.**localID**,sizeof(**int**));**  **if(**DEBUG**)**printf**(**"\trespond written (%d)\n"**,**fdW**);**  **}**  **else**  **if(**cur\_ask**.**action **==** 2**)** **{** //отправка сообещния в очередь  subscriber**\*** curSUB **=** subLISTlocate**(&**subscribers**,**cur\_ask**.**localID**);**  **if(**curSUB **==** **NULL)** **{**  **if(**DEBUG**)**printf**(**"ask handled (ask error)\n"**);**  **continue;**  **}**  msgQ**\*** curQ **=** queueMASlocate**(&**allQUEUES**,**cur\_ask**.**Qid**);**  msg**\*** curMSG **=** **(**msg**\*)**malloc**(sizeof(**msg**));**  **if(**curQ **==** **NULL** **||** curMSG **==** **NULL)** **{**  **if(**DEBUG**)**printf**(**"ask handled (ask error)\n"**);**  **continue;**  **}**    read**(**curSUB**->**localPortIn**,&**curMSG**->**priority**,sizeof(**int**));**  read**(**curSUB**->**localPortIn**,&**curMSG**->**size**,sizeof(**int**));**  curMSG**->**content **=** **(**char**\*)**malloc**(**curMSG**->**size**);**  read**(**curSUB**->**localPortIn**,**curMSG**->**content**,**curMSG**->**size**);**  Qadd**(**curQ**,**curMSG**);**  **}**  **else**  **if(**cur\_ask**.**action **==** 3**)** **{** //чтение сообщения из очереди  subscriber**\*** curSUB **=** subLISTlocate**(&**subscribers**,**cur\_ask**.**localID**);**  **if(**curSUB **==** **NULL)** **{**  **if(**DEBUG**)**printf**(**"ask handled (ask error)\n"**);**  **continue;**  **}**  msgQ**\*** curQ **=** queueMASlocate**(&**allQUEUES**,**cur\_ask**.**Qid**);**  msg**\*** curMSG **=** **NULL;**  **if(**curQ **==** **NULL)** **{**  curMSG **=** **NULL;**  **}**  **else** **{**  curMSG **=** Qget**(**curQ**);**  **}**  **if(**curMSG **==** **NULL)** **{**  //написать чето чтоб было понятно что сообщение пусктое  int resp **=** **-**1**;**  write**(**curSUB**->**localPortOut**,&**resp**,sizeof(**int**));**  **}**  **else** **{**  write**(**curSUB**->**localPortOut**,&**curMSG**->**priority**,sizeof(**int**));**  write**(**curSUB**->**localPortOut**,&**curMSG**->**size**,sizeof(**int**));**  write**(**curSUB**->**localPortOut**,**curMSG**->**content**,**curMSG**->**size**);**  //write(curSUB->localPortOut,curMSG,sizeof(\*curMSG));  **if(**curQ**->**logFlag **!=** 1**)** **{**  free**(**curMSG**->**content**);**  free**(**curMSG**);**  **}**  **}**  **}**  **else**  **if(**cur\_ask**.**action **==** 4**)** **{** //создание очереди  //printf(">|%d|%d|\n",nameIN[0],nameIN[1]);  subscriber**\*** curSUB **=** subLISTlocate**(&**subscribers**,**cur\_ask**.**localID**);**  int resp **=** 0**;**  **if(**curSUB **==** **NULL)** **{**  **if(**DEBUG**)**printf**(**"ask handled (ask error)\n"**);**  **continue;**  **}**  **if(**queueMASlocate**(&**allQUEUES**,**cur\_ask**.**Qid**)** **!=** **NULL)** **{**  resp **=** 1**;**  write**(**curSUB**->**localPortOut**,&**resp**,sizeof(**int**));**  **if(**DEBUG**)**printf**(**"ask handled (ask error)\n"**);**  **continue;**  **}**  **if(**nameIN**[**0**]** **!=** '0'**)** **{**  //printf("making log\n");  queueMASaddLogged**(&**allQUEUES**,**cur\_ask**.**Qid**,**MAXLOGLEN**);**  **}**  **else** **{**  queueMASadd**(&**allQUEUES**,**cur\_ask**.**Qid**);**  **}**  write**(**curSUB**->**localPortOut**,&**resp**,sizeof(**int**));**  **}**  **if(**DEBUG**)**printf**(**"ask handled\n"**);**  **}**  free**(**nameIN**);**  free**(**nameOUT**);**  **}** |

**MSQserver.c**

|  |
| --- |
| #include <string.h>  #include <unistd.h>  #include <stdlib.h>  #include <fcntl.h>  #include <math.h>  #include <signal.h>  #include "transformres.h"  #include <strings.h>  #include <sys/types.h>  #include <sys/stat.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <stdio.h>  //#include "msgOPR.h"  #include "msQ.h"  #define MAP\_FILE\_SIZE 100  #define DEBUG 0  int MAXLOGLEN **=** 10**;**  int doStartOpr **=** 0**;**  char portname**[**20**];**  void configurate**()** **{**  FILE**\*** conf **=** **NULL;**  conf **=** fopen**(**"config"**,**"r+"**);**  char tmp**[**10**];**  **if(**conf **==** **NULL)** **{**  conf **=** fopen**(**"config"**,**"w+"**);**  fprintf**(**conf**,**"DOCONF %d\n"**,**doStartOpr**);**  **}**  **else** **{**  //поиск  fseek**(**conf**,**0**,**SEEK\_SET**);**  **if(**fscanf**(**conf**,**"%s"**,**tmp**)** **==** EOF**)** **{**  printf**(**"bad config\n"**);**  exit**(-**123**);**  **}**  **while(**strcmp**(**tmp**,**"DOCONF"**)!=**0**)** **{**  **if(**fscanf**(**conf**,**"%s"**,**tmp**)** **==** EOF**)** **{**  printf**(**"bad config\n"**);**  exit**(-**123**);**  **}**  **}**  //нашли  fscanf**(**conf**,**"%d"**,&**doStartOpr**);**  **if(**doStartOpr **==** 1**)**  **{**  //поиск  fseek**(**conf**,**0**,**SEEK\_SET**);**  **if(**fscanf**(**conf**,**"%s"**,**tmp**)** **==** EOF**)** **{**  printf**(**"bad config\n"**);**  exit**(-**123**);**  **}**  **while(**strcmp**(**tmp**,**"MAXLEN"**)!=**0**)** **{**  **if(**fscanf**(**conf**,**"%s"**,**tmp**)** **==** EOF**)** **{**  printf**(**"bad config\n"**);**  exit**(-**123**);**  **}**  **}**  //нашли  fscanf**(**conf**,**"%d"**,&**MAXLOGLEN**);**  printf**(**"will create OPR | maxLOGlen: %d\n"**,**MAXLOGLEN**);**  **}**  **else** **{**  //поиск  fseek**(**conf**,**0**,**SEEK\_SET**);**  **if(**fscanf**(**conf**,**"%s"**,**tmp**)** **==** EOF**)** **{**  printf**(**"bad config\n"**);**  exit**(-**123**);**  **}**  **while(**strcmp**(**tmp**,**"PORTNM"**)!=**0**)** **{**  **if(**fscanf**(**conf**,**"%s"**,**tmp**)** **==** EOF**)** **{**  printf**(**"bad config\n"**);**  exit**(-**123**);**  **}**  **}**  //нашли  fscanf**(**conf**,**"%s"**,**portname**);**  printf**(**"will connect OPR | ipn: %s\n"**,**portname**);** //input pipe name  **}**  **}**  **}**  int main**(**int argc**,**char**\*\*** argv**)** **{**  int sockfd1 **=** socket**(**AF\_INET**,** SOCK\_STREAM**,** 0**);**  **if(**argc **<** 2**)** **{**  **return** **-**2**;**  **}**  configurate**();**  int fdTOopr**;**  **if(**doStartOpr **==** 0**)** **{**  fdTOopr **=** open**(**portname**,**O\_WRONLY**);**  **}**  **else** **{**  msQ queueMain**;**  CreateQueue**(&**queueMain**,**MAXLOGLEN**);**  fdTOopr **=** queueMain**.**publicPort**;**  **}**    **if(**sockfd1 **==** **-**1**)** **{**  printf**(**"error: sock\n"**);**  **return** 0**;**  **}**  **if(**fdTOopr **==** **-**1**)** **{**  printf**(**"error: opr\n"**);**  **return** 0**;**  **}**  int port **=** atoi**(**argv**[**1**]);**  printf**(**"port: %d\n"**,**port**);**  struct sockaddr\_in adr**;**  bzero**((**char **\*)** **&**adr**,** **sizeof(**adr**));**  adr**.**sin\_port **=** htons**(**port**);**  adr**.**sin\_family **=** AF\_INET**;**  adr**.**sin\_addr**.**s\_addr **=** htonl**(**INADDR\_ANY**);**  **if(**bind**(**sockfd1**,** **(**struct sockaddr **\*)&**adr**,** **sizeof(**adr**))** **<** 0**)** **{**  printf**(**"ERROR binding: bad port\n"**);**  **return** 0**;**  **}**  listen**(**sockfd1**,** 1**);**  printf**(**"server started\n"**);**  **for(;;)** **{**  int sockfdCON **=** accept**(**sockfd1**,NULL,NULL);**  int pid **=** fork**();**  **if(**pid **!=** 0**)** **{**  **continue;**  **}**  **else** **{**  execl**(**"retranslator.exe"**,**"retranslator.exe"**,**transform**(**sockfdCON**),**transform**(**fdTOopr**),NULL);**//запуск ретранслятора  **}**  **}**  **}** |

**Retranslator.c**

Программа, запускаемая сервером и передающая запросы с порта на оператор

|  |
| --- |
| #include <string.h>  #include <unistd.h>  #include <stdlib.h>  #include <fcntl.h>  #include <math.h>  #include <signal.h>  #include "transformres.h"  #include <sys/types.h>  #include <sys/stat.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <stdio.h>  #include "msgOPR.h"  #define MAP\_FILE\_SIZE 100  #define DEBUG 0  int main**(**int argc**,**char**\*\*** argv**)** **{**  **if(**argc **<** 3**)** **{**  **return** **-**1**;**  **}**  int sockfd **=** back\_transform**(**argv**[**1**]);**  int fdTOopr **=** back\_transform**(**argv**[**2**]);**    **if(**sockfd **==** **-**1**)** **{**  printf**(**"error: sock\n"**);**  **return** **-**1**;**  **}**  **else** **{**  printf**(**"retranslator activated\n"**);**  **}**  request rqt**;**  **for(;;)** **{**  **if(**read**(**sockfd**,&**rqt**,sizeof(**request**))** **==** **sizeof(**request**))**  write**(**fdTOopr**,&**rqt**,sizeof(**request**));**  **}**  **}** |

**Transformres.c**

Библиотека, содержащая функции примерно аналогичные по действию с atoi и itoa. Наследуется некоторыми частями моих программ с тех времен, когда я еще не знал о существовании вышеупомянутых функций.

|  |
| --- |
| #include <stdlib.h>  char**\*** transform**(**int inp**)**  **{**  int i**=**0**;**  char**\*** str **=** **(**char**\*)**malloc**(sizeof(**char**)\***1**);**  **while(**inp **!=** 0**)**  **{**  str **=** **(**char**\*)**realloc**(**str**,sizeof(**char**)\*(**i**+**1**));**  str**[**i**]** **=** **(**inp**%**10**)+**'0'**;**  **++**i**;**  inp**/=**10**;**  **}**  str**[**i**]** **=** 0**;**  **return** str**;**  **}**  int back\_transform**(**char**\*** inp**)**  **{**  int res **=** 0**;**  int mult **=** 1**;**  int i **=** 0**;**  **while(**inp**[**i**]!=** 0**)**  **{**  res **+=** **(**inp**[**i**]-**'0'**)\***mult**;**  **++**i**;**  mult**\*=**10**;**  **}**  //free(inp);  **return** res**;**  **}** |

**Makefile**

|  |
| --- |
| all: src/queueCHEK.c src/msgOPR.c src/operator.c src/transformres.c src/MSQserver.c src/retranslator.c  gcc -Wall -std=c99 -D\_POSIX\_C\_SOURCE=199309L -o ch.exe src/queueCHEK.c src/msgOPR.c src/transformres.c  gcc -Wall -std=c99 -D\_POSIX\_C\_SOURCE=199309L -o operator.exe src/operator.c src/msgOPR.c src/transformres.c  gcc -Wall -std=c99 -D\_POSIX\_C\_SOURCE=199309L -o MSQserver.exe src/MSQserver.c src/msgOPR.c src/transformres.c  gcc -Wall -std=c99 -D\_POSIX\_C\_SOURCE=199309L -o retranslator.exe src/retranslator.c src/msgOPR.c src/transformres.c  operator: src/msgOPR.c src/operator.c  gcc -Wall -std=c99 -D\_POSIX\_C\_SOURCE=199309L -o operator.exe src/msgOPR.c src/operator.c src/transformres.c  server: src/MSQserver.c src/retranslator.c src/msgOPR.c src/transformres.c  gcc -Wall -std=c99 -D\_POSIX\_C\_SOURCE=199309L -o MSQserver.exe src/MSQserver.c src/msgOPR.c src/transformres.c  gcc -Wall -std=c99 -D\_POSIX\_C\_SOURCE=199309L -o retranslator.exe src/retranslator.c src/msgOPR.c src/transformres.c  client: src/queueCHEK.c src/msgOPR.c src/transformres.c  gcc -Wall -std=c99 -D\_POSIX\_C\_SOURCE=199309L -o ch.exe src/queueCHEK.c src/msgOPR.c src/transformres.c |

**Пример работы**

Для тестов сервера я создал тестового клиента с предопределенным порядком действий. Это программа, которая:

1. Создает дочерний процесс
2. Оба процесса одновременно создают разные очереди и отправляют/принимают сообщения в них. (показывает, что сервер нормально обрабатывает одновременные запросы). В ходе этого теста проверяются также логирующие очереди и приоритеты.
3. Один процесс посылает другому сообщение через сервер.

Исходник тестового клиента:

|  |
| --- |
| #include <string.h>  #include <unistd.h>  #include <stdlib.h>  #include <fcntl.h>  #include <math.h>  #include <signal.h>  #include "transformres.h"  #include <strings.h>  #include <sys/types.h>  #include <sys/socket.h>  #include <netinet/in.h>  #include <netinet/in.h>  #include <stdio.h>  #include "msQ.h"  #define MAP\_FILE\_SIZE 100  #define DEBUG 0  int main**(**int argc**,**char**\*\*** argv**)** **{**  printf**(**"creating\n"**);**  int mn **=** 1**;**  int pid **=** fork**();**  **if(**pid **==** 0**)** **{**  mn **=** 5**;**  **}**  printf**(**"\t\t <<<%d\n"**,**mn**);**  msQ mainQ**;**  int res **=** connectToQ**(&**mainQ**,**argv**[**1**],**atoi**(**argv**[**2**]));**  **if(**res **<** 0**)** **{**  printf**(**"error connecting\n"**);**  **return** 0**;**  **}**    printf**(**"[%d] subscribing\n"**,**getpid**());**  int myID **=** subscribe**(&**mainQ**);**    printf**(**"[%d] makingQ (ind %d)\n"**,**getpid**(),**mn**);**  **if(**0 **!=** makeQ**(&**mainQ**,**1**\***mn**,**0**))** **{**  printf**(**"[%d] error making\n"**,**getpid**());**  **}**  **else** **{**  printf**(**"[%d] made\n"**,**getpid**());**  **}**  msg**\*** ms**;**  ms **=** createMsg**(**0**,**"1924"**);**  printf**(**"[%d] sending to %d (%s)\n"**,**getpid**(),**mn**,**ms**->**content**);**  sendMSG**(&**mainQ**,**ms**,**myID**,**1**\***mn**);**  msg**\*** msGOT**;**  msGOT **=** createMsg**(**0**,**""**);**  printf**(**"[%d] reciving from %d\n"**,**getpid**(),**mn**);**  getMSG**(&**mainQ**,**msGOT**,**myID**,**1**\***mn**);**  printf**(**"[%d] recived: %s\n"**,**getpid**(),**msGOT**->**content**);**    sleep**(**2**);**  strcpy**(**ms**->**content**,**"300"**);**  printf**(**"[%d] sending to %d (%s)\n"**,**getpid**(),**mn**,**ms**->**content**);**  sendMSG**(&**mainQ**,**ms**,**myID**,**1**\***mn**);**  printf**(**"[%d] reciving from %d\n"**,**getpid**(),**mn**);**  getMSG**(&**mainQ**,**msGOT**,**myID**,**1**\***mn**);**  printf**(**"[%d] recived: %s\n"**,**getpid**(),**msGOT**->**content**);**    printf**(**"[%d] makingQ (ind %d)\n"**,**getpid**(),**mn**\***2**);**  **if(**0 **!=** makeQ**(&**mainQ**,**2**\***mn**,**1**))** **{** //очереди с индексами 2 и 10 - логируемые.  printf**(**"[%d] error making\n"**,**getpid**());**  **}**  **else** **{**  printf**(**"[%d] made\n"**,**getpid**());**  **}**  ms**->**priority **=** 1**;**  strcpy**(**ms**->**content**,**"400"**);**  printf**(**"[%d] sending to %d (%s)\n"**,**getpid**(),**mn**\***2**,**ms**->**content**);**  sendMSG**(&**mainQ**,**ms**,**myID**,**2**\***mn**);**  ms**->**priority **=** 8**;** //сообщение "310" было отправлено позже "400", однако у него больше приоритет. Оно должно быть принято первым  strcpy**(**ms**->**content**,**"310"**);**  printf**(**"[%d] sending to %d (%s)\n"**,**getpid**(),**mn**\***2**,**ms**->**content**);**  sendMSG**(&**mainQ**,**ms**,**myID**,**2**\***mn**);**  printf**(**"[%d] reciving from %d\n"**,**getpid**(),**2**\***mn**);**  **if(**0 **==** getMSG**(&**mainQ**,**msGOT**,**myID**,**2**\***mn**))**  printf**(**"[%d] recived: %s\n"**,**getpid**(),**msGOT**->**content**);**  printf**(**"[%d] reciving from %d\n"**,**getpid**(),**2**\***mn**);**  **if(**0 **==** getMSG**(&**mainQ**,**msGOT**,**myID**,**2**\***mn**))**  printf**(**"[%d] recived: %s\n"**,**getpid**(),**msGOT**->**content**);**  printf**(**"[%d] reciving from %d\n"**,**getpid**(),**2**\***mn**);**  **if(**0 **==** getMSG**(&**mainQ**,**msGOT**,**myID**,**2**\***mn**))**  printf**(**"[%d] recived: %s\n"**,**getpid**(),**msGOT**->**content**);**  sleep**(**1**);**  printf**(**"LAST TEST\n"**);**  **if(**mn **==** 5**)** **{**  free**(**ms**);**  ms **=** createMsg**(**0**,**"sentTOother"**);**  printf**(**"[%d] sending to %d (%s)\n"**,**getpid**(),**mn**\***2**,**ms**->**content**);**  sendMSG**(&**mainQ**,**ms**,**myID**,**2**\***mn**);**  **}**  **else** **{**  sleep**(**1**);**  **}**  **if(**mn **==** 1**)** **{**  printf**(**"[%d] reciving from %d\n"**,**getpid**(),**4**\***mn**);**  **if(**0 **==** getMSG**(&**mainQ**,**msGOT**,**myID**,**5**\***2**\***mn**))**  printf**(**"[%d] recived: %s\n"**,**getpid**(),**msGOT**->**content**);**  **}**  **return** 0**;**  **}** |

Команда для запуска сервера (сразу с трассировкой):

strace -f -o traces/serv-tr.t ./MSQserver.exe 8311 #запуск сервера

Команда для запуска тест-клиента:

strace -f -o traces/clit-tr.t ./ch.exe 127.0.0.1 8311 #запуск тест-клиента

Config-файл:

|  |
| --- |
| DOCONF 1  MAXLEN 10  PORTNM tmp/mainfif |

Запускаем эти две команды в двух отдельных терминалах. Имеем такой вывод тест-клиента:

|  |
| --- |
| max@max-Swift:~/Рабочий стол/OS/КП/prog/yes=server$ strace -f -o traces/clit-tr.t ./ch.exe 127.0.0.1 8311 #запуск тест-клиента  creating  <<<1  <<<5  [119932] subscribing  [119933] subscribing  <tmp/tmrc|tmp/fmjh>  <tmp/lgph|tmp/orjj>  <tmp/tmrc|tmp/fmjh>  <tmp/lgph|tmp/orjj>  [119933] makingQ (ind 5)  [119932] makingQ (ind 1)  [119933] made  [119933] sending to 5 (1924)  [119933] reciving from 5  [119932] made  [119932] sending to 1 (1924)  [119932] reciving from 1  [119933] recived: 1924  [119932] recived: 1924  [119933] sending to 5 (300)  [119933] reciving from 5  [119932] sending to 1 (300)  [119932] reciving from 1  [119933] recived: 300  [119933] makingQ (ind 10)  [119932] recived: 300  [119932] makingQ (ind 2)  [119933] made  [119933] sending to 10 (400)  [119933] sending to 10 (310)  [119932] made  [119932] sending to 2 (400)  [119932] sending to 2 (310)  [119933] reciving from 10  [119932] reciving from 2  [119933] recived: 310 **<-ВИДИМ, ЧТО НЕСМОТРЯ НА ТО, ЧТО СООБЩЕНИЕ ОТПРАВЛЕНО ПОЗЖЕ, ЧЕМ 400 ОНО ПРИНЯТО ПЕРВЫМ, ВЕДЬ ЕГО ПРИОРИТЕТ ВЫШЕ**  [119933] reciving from 10  [119932] recived: 310  [119932] reciving from 2  [119933] recived: 400  [119933] reciving from 10  [119932] recived: 400  [119932] reciving from 2  LAST TEST  LAST TEST  [119933] sending to 10 (sentTOother)  [119932] reciving from 10  [119932] recived: sentTOother **<-ВИДИМ, ЧТО ПРОЦЕССЫ МОГУТ УСПЕШНО ПЕРЕДАВАТЬ СООБЩЕНИЯ МЕЖДУ ДРУГ ДРУГОМ**  max@max-Swift:~/Рабочий стол/OS/КП/prog/yes=server$ |

Теперь посмотрим вывод сервера (сервер запущен в режиме отладки, так что все что на нем происходит выводится на экран)

|  |
| --- |
| max@max-Swift:~/Рабочий стол/OS/КП/prog/yes=server$ strace -f -o traces/serv-tr.t ./MSQserver.exe 8311 #запуск сервера  will create OPR | maxLOGlen: 1  operator activatrd |10| **<-АКТИВАЦИЯ ОПРЕАТОРА**  port: 8311  server started  retranslator activated**<-АКТИВАЦИЯ РЕТРАНСЛЯТОРА (ПО ОДНОМУ НА ПОДПИСЧИКА)**  retranslator activated  operator| got ask | from: 119933  operator ask: 1 | 0  sub added  respond written (7)  ask handled  operator| got ask | from: 119932  operator ask: 1 | 0  sub added  respond written (9)  ask handled  operator| got ask | from: 119933  operator ask: 4 | 5  ask handled  operator| got ask | from: 119932  |5|-|(0)  operator ask: 4 | 1  ask handled  operator| got ask | from: 119933  |5|-|(0) **<-ВОТ ЗДЕСЬ НАХОДИТСЯ СПИСОК ТЕКУЩИХ ОЧЕРЕДЕЙ И ИХ СОДЕРЖИМОГО**  |1|-|(0)  operator ask: 2 | 5 **<-ИНДЕКС ЗАПРОСА | ИНДЕКС ОЧЕРЕДИ**  ask handled  operator| got ask | from: 119933  |5|-|1924|(1)  |1|-|(0)  operator ask: 3 | 5  ask handled  operator| got ask | from: 119932  |5|-|(0)  |1|-|(0)  operator ask: 2 | 1  ask handled  operator| got ask | from: 119932  |5|-|(0)  |1|-|1924|(1)  operator ask: 3 | 1  ask handled  operator| got ask | from: 119933  |5|-|(0)  |1|-|(0)  operator ask: 2 | 5  ask handled  operator| got ask | from: 119933  |5|-|300|(1)  |1|-|(0)  operator ask: 3 | 5  ask handled  operator| got ask | from: 119932  |5|-|(0)  |1|-|(0)  operator ask: 2 | 1  ask handled  operator| got ask | from: 119932  |5|-|(0)  |1|-|300|(1)  operator ask: 3 | 1  ask handled  operator| got ask | from: 119933  |5|-|(0)  |1|-|(0)  operator ask: 4 | 10  ask handled  operator| got ask | from: 119932  |5|-|(0)  |1|-|(0)  |10|-|(0)  operator ask: 4 | 2  ask handled  operator| got ask | from: 119933  |5|-|(0)  |1|-|(0)  |10|-|(0)  |2|-|(0)  operator ask: 2 | 10  ask handled  operator| got ask | from: 119933  |5|-|(0)  |1|-|(0)  |10|-|400|(1)  |2|-|(0)  operator ask: 2 | 10  ask handled  operator| got ask | from: 119932  |5|-|(0)  |1|-|(0)  |10|-|310|400|(2)  |2|-|(0)  operator ask: 2 | 2  ask handled  operator| got ask | from: 119932  |5|-|(0)  |1|-|(0)  |10|-|310|400|(2)  |2|-|400|(1)  operator ask: 2 | 2  ask handled  operator| got ask | from: 119933  |5|-|(0)  |1|-|(0)  |10|-|310|400|(2)  |2|-|310|400|(2)  operator ask: 3 | 10  ask handled  operator| got ask | from: 119932  |5|-|(0)  |1|-|(0)  |10|-|400|(1)  |2|-|310|400|(2)  operator ask: 3 | 2  ask handled  operator| got ask | from: 119933  |5|-|(0)  |1|-|(0)  |10|-|400|(1)  |2|-|400|(1)  operator ask: 3 | 10  ask handled  operator| got ask | from: 119932  |5|-|(0)  |1|-|(0)  |10|-|(0)  |2|-|400|(1)  operator ask: 3 | 2  ask handled  operator| got ask | from: 119933  |5|-|(0)  |1|-|(0)  |10|-|(0)  |2|-|(0)  operator ask: 3 | 10  ask handled  operator| got ask | from: 119932  |5|-|(0)  |1|-|(0)  |10|-|(0)  |2|-|(0)  operator ask: 3 | 2  ask handled  operator| got ask | from: 119933  |5|-|(0)  |1|-|(0)  |10|-|(0)  |2|-|(0)  operator ask: 2 | 10  ask handled  operator| got ask | from: 119932  |5|-|(0)  |1|-|(0)  |10|-|sentTOother|(1)  |2|-|(0)  operator ask: 3 | 10  ask handled  ^C  max@max-Swift:~/Рабочий стол/OS/КП/prog/yes=server$ |

А вот лог-файл. Он был создан перед остановкой оператора

|  |
| --- |
| -----------  [2]: 310 400  [10]: 310 400 sentTOother  ----------- |

Если провести тот же тест, но с длиной логирующей очереди равной 1, то лог-файл выглядит так:

|  |
| --- |
| -----------  [2]: 400  [10]: sentTOother  ----------- |

**Вывод**

Работа над этим КП дала мне довольно много опыта в работе с такими новыми для меня системами передачи данных, как именованные пайпы и сокеты. Также я по сути впервые использовал strace для серьезного дебага, что, как оказалось, довольно удобно (в контексте настолько повязанной на системные вызовы программы, разумеется). Сама программа получилась, как мне кажется, относительно строго делящейся по уровням обработки данных и, как следствие, достаточно ххорошо поддающейся модификации (как минимум, мне удалось почти ничего на меняя приделать к ней в последний момент сервер). Конечно же, раз я делаю сервер на пайпах, с ним нельзя взаимодействовать удаленно, просто отправляя на порт сообщения с другой машины. Однако, возможно, если для удаленного взаимодействия удастся как-то примонтировать файловую систему машины клиента, то из этого что-нибудь выйдет. Также стоит отметить, что я в принципе представляю себе, какие модификации нужно сделать, чтобы перевести все взаимодействие с клиентом на порты, не меняя при этом кардинально архитектуру программы, однако это уже будет скорее сервер на сокетах, а не на пайпах. Также стоит отметить, что из-за полностью линейной обработки запросов не исключено, что при работе с большим числом клиентов серер будет неэффективным, однако, не имея сформировавшегося представления о том, как эту обработку можно прям полностью распараллелить, я не возьмусь рассуждать об этом хоть сколько-нибудь глубоко.