Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

**МЕЖПРОЦЕССНОЕ ВЗАИМОДЕЙСТВИЕ И СИНХРОНИЗАЦИЯ В LINUX**

Лабораторная работа № 8

по курсу «Операционные системы»

Вариант № 3

Выполнил: студент группы 220601 \_\_\_\_\_\_\_\_\_ Белым А.А.

(подпись)

Проверил: \_\_\_\_\_\_\_\_\_ Попов А.И.

(подпись)

Тула 2012

# Цель работы

Цель работы состоит в том, чтобы познакомиться со средствами обмена информации между процессами и средствами межпроцессной синхронизации и научиться их использовать.

# Задание

Написать программы, демонстрирующие взаимодействие процессов при помощи разделяемой памяти (System V IPC).

# Теоретическая справка

В основе межпроцессного взаимодействия (IPC - interprocess communication) лежит обмен данными между работающими процессами.

Не существует единого универсального метода взаимодействия процессов. Выбор того или иного способа взаимодействия зависит от поставленных задач.

Межпроцессное взаимодействие в Linux можно классифицировать на:

- локальное (обмен данными между процессами, работающими в одной linux-системе) и удаленное (обмен данными между процессами, работающими в разных системах),

- однонаправленное (один процесс может быть только отправителем информации, другой – только получателем) и двунаправленное,

- закрытое (когда взаимодействие осуществляется только между двумя процессами) и открытое (когда какой-нибудь еще процесс может присоединиться к обмену данными) .

Локальные методы межпроцессного взаимодействия:

- сигналы,

- сообщения,

- общая память,

- общие файлы,

- программные каналы

Удаленные методы межпроцессного взаимодействия:

- сокеты,

- удаленный вызов процедур

Средства синхронизации процессов:

- семафоры System V.

- семафоры и мьютексы POSIX (описанные в библиотеке pthreads).

Часть механизмов синхронизации и межпроцессного взаимодействия, впервые появившихся в UNIX System V и впоследствии перекочевавших оттуда практически во все современные версии операционной системы UNIX, получила общее название System V IPC (IPC – сокращение от interprocess communications). В группу System V IPC входят: очереди сообщений, разделяемая память и семафоры. Эти средства организации взаимодействия процессов связаны не только общностью происхождения, но и обладают схожим интерфейсом для выполнения подобных операций, например, для выделения и освобождения соответствующего ресурса в системе.

Все средства связи System V IPC требуют предварительных инициализирующих действий (создания) для организации взаимодействия процессов.

* Системный вызов shmget предназначен для выполнения операции доступа к сегменту разделяемой памяти и, в случае его успешного завершения, возвращает дескриптор System V IPC для этого сегмента (целое неотрицательное число, однозначно характеризующее сегмент внутри вычислительной системы и использующееся в дальнейшем для других операций с ним).
* Системный вызов shmat предназначен для включения области разделяемой памяти в адресное пространство текущего процесса.
* Системный вызов shmdt предназначен для исключения области разделяемой памяти из адресного пространства текущего процесса.

# Инструкция пользователю

Программа позволяет организовать чат на локальном копьютере.

Программа позволяет задавать ник пользователя и текст сообщения. После нажатия кнопки на всех клиентах появляется указанное сообщение, ник, и время создания сообщения.

# Инструкция программисту

**Перечисление:**

В анонимном перечислении хранятся уникальные номера для генерации ключей System V с помощью функции ftok.

enum **{**

shmem\_key\_id**=**1**,** - для ключа разделяемой памяти.

shmem\_lock\_key\_id**,** - для ключа семафора, защищающего разделяемую память

sem\_notified\_key\_id**,** - для ключа семафора для уведомления о новом сооб-

щении

**};**

**Константы:**

const char **\***key\_name**=**"/home/"**;**

Имя файла для создания ключа System V.

const size\_t default\_size**=**0x1000**;**

Размер разделяемой памяти.

**Переменные:**

int shmem**,** - дескриптор System V разделяемой памяти.

shmem\_lock**,** - дескриптор семафора защиты разделяемой памяти.

sem\_notified**;** - дескриптор семафора для уведомления о новом сообщении.

char **\***shmdata**=NULL;**

Указатель на сообщение, хранящееся в разделяемой памяти.

int **\***clients\_count**;**

Указатель на количество клиентов, хранящееся в разделяемой памяти.

**Функции:**

void SemLock**(**int semid**)**

Уменьшает значение указанного семафора на 1.

void SemUnlock**(**int semid**)**

Увеличивает значение указанного семафора на 1.

void **\***updater**(**void **\***args**)**

Поток для библиотеки Pthreads, считывает новые сообщения из разделяемой памяти. Ожидает нового собщения на семафоре sem\_notified.

bool CreateShMemLock**(**int semid**,**const int key\_id**)**

Создает семафор для защиты разделяемой памяти. Семафор создается и его значение сразу увеличивается на 1. Если семафор уже был создан другим потоком, оставляет его значение без изменений.

bool CreateSem**(**int **&**semid**,**const int key\_id**)**

Создает семафор со значением 0.

bool CreateShMem**(**int **&**shmid**,**void **\*&**data**)**

Создает объект разделяемй памяти.

void SendMessage**(**const char **\***msg**)**

Сохраняет сообщение в разделяемой памяти и разблокирует процессы, ожидающие на семафоре sem\_notified.

# Текст программы

Ниже представлен текст программы для локального чата, написанной на языке C++, в среде Qt Creator 2.6.0rc + GCC 4.7.2 с использованием библиотеки Qt.

mainwindow.h:

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

**namespace** Ui **{**

class MainWindow**;**

**}**

class MainWindow **:** public QMainWindow

**{**

Q\_OBJECT

public**:**

explicit MainWindow**(**QWidget **\***parent **=** 0**);**

**~**MainWindow**();**

private slots**:**

void addMessage**(**char **\***msg**);**

void on\_pushButton\_clicked**();**

private**:**

Ui**::**MainWindow **\***ui**;**

**};**

#endif // MAINWINDOW\_H

mainwindow.cpp:

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include <QDebug>

extern "C"**{**

#include <sys/ipc.h>

#include <sys/sem.h>

#include <sys/shm.h>

**}**

#include <cerrno>

#include <cstdio>

#include <QTime>

#include "threadhelper.h"

ThreadHelper threadHelper**;**

const char **\***html\_template**=**"<html>"

"<head/>"

"<body>"

"(%1) <span style=\"color:blue;

font-style:italic\">%2</span>:"

"</body>"

"</html>"

"%3"**;**

const char **\***key\_name**=**"/home/"**;**

const size\_t default\_size**=**0x1000**;**

enum **{**

shmem\_key\_id**=**1**,**

shmem\_lock\_key\_id**,**

sem\_notified\_key\_id**,**

**};**

int shmem**,**shmem\_lock**,**sem\_notified**;** char **\***shmdata**=NULL;**int **\***clients\_count**;**

void SemLock**(**int semid**){**

sembuf ops**;**

ops**.**sem\_num**=**0**;**

ops**.**sem\_flg**=**0**;**

ops**.**sem\_op**=-**1**;**

semop**(**semid**,&**ops**,**1**);**

**}**

void SemUnlock**(**int semid**){**

sembuf ops**;**

ops**.**sem\_num**=**0**;**

ops**.**sem\_flg**=**0**;**

ops**.**sem\_op**=**1**;**

semop**(**semid**,&**ops**,**1**);**

**}**

void **\***updater**(**void **\***args**){**

sembuf ops**;**

ops**.**sem\_flg**=**0**;**

ops**.**sem\_num**=**0**;**

**while(true){**

ops**.**sem\_op**=-**1**;**

**if(**semop**(**sem\_notified**,&**ops**,**1**)<**0**)**

pthread\_exit**(NULL);**

threadHelper**.**emitAddMessage**(**shmdata**);**

ops**.**sem\_op**=**0**;**

**if(**semop**(**sem\_notified**,&**ops**,**1**)<**0**)**

pthread\_exit**(NULL);**

**}**

pthread\_exit**(NULL);**

**}**

bool CreateShMemLock**(**int semid**,**const int key\_id**){**

key\_t key**;**

**if((**key**=**ftok**(**key\_name**,** key\_id**))<**0**){**

perror**(NULL);**

qDebug**()<<**"Key error!"**;**

**return** **false;**

**}**

**if((**semid**=**semget**(**key**,**1**,**IPC\_CREAT**|**0600**))<**0**){**

**if(**errno**==**EEXIST**){**

**if((**semid**=**semget**(**key**,**1**,**0600**))>=**0**)**

**return** **true;**

**}**

perror**(NULL);**

qDebug**()<<**"init sem error!"**;**

**return** **false;**

**}** **else** **{**

sembuf ops**;**

ops**.**sem\_op**=**1**;**

ops**.**sem\_num**=**0**;**

ops**.**sem\_flg**=**0**;**

semop**(**semid**,&**ops**,**1**);**

**};**

**return** **true;**

**}**

bool CreateSem**(**int **&**semid**,**const int key\_id**){**

key\_t key**;**

**if((**key**=**ftok**(**key\_name**,**key\_id**))<**0**){**

perror**(NULL);**

qDebug**()<<**"Key error!"**;**

**return** **false;**

**}**

**if((**semid**=**semget**(**key**,**1**,**IPC\_CREAT**|**0666**))<**0**){**

perror**(NULL);**

qDebug**()<<**"init sem error!"**;**

**return** **false;**

**};**

**return** **true;**

**}**

bool CreateShMem**(**int **&**shmid**,**void **\*&**data**){**

key\_t key**;**

**if((**key**=**ftok**(**key\_name**,**shmem\_key\_id**))<**0**){**

perror**(NULL);**

qDebug**()<<**"Key error!"**;**

**return** **false;**

**}**

**if((**shmid**=**shmget**(**key**,**default\_size**,**IPC\_CREAT**|**0666**))<**0**){**

perror**(NULL);**

qDebug**()<<**"shmget error!"**;**

**return** **false;**

**}**

**if((**data**=**shmat**(**shmid**,NULL,**0**))==(**void**\*)-**1**){**

perror**(NULL);**

qDebug**()<<**"shmat error!"**;**

**return** **false;**

**}**

**return** **true;**

**}**

void SendMessage**(**const char **\***msg**){**

sembuf ops**;**

ops**.**sem\_op**=**0**;**

ops**.**sem\_flg**=**0**;**

ops**.**sem\_num**=**0**;**

semop**(**sem\_notified**,&**ops**,**1**);**

SemLock**(**shmem\_lock**);**

strcpy**(**shmdata**,**msg**);**

SemUnlock**(**shmem\_lock**);**

ops**.**sem\_op**=\***clients\_count**;**

semop**(**sem\_notified**,&**ops**,**1**);**

**}**

MainWindow**::**MainWindow**(**QWidget **\***parent**)** **:**

QMainWindow**(**parent**),**

ui**(new** Ui**::**MainWindow**)**

**{**

ui**->**setupUi**(this);**

connect**(&**threadHelper**,**SIGNAL**(**addMessage**(**char**\*)),this,**SLOT**(**addMessage**(**char**\*)));**

**if(!**CreateShMemLock**(**shmem\_lock,shmem\_lock\_key\_id**))**

exit**(-**1**);**

**if(!**CreateSem**(**sem\_notified**,**sem\_notified\_key\_id**))**

exit**(-**1**);**

void **\***temp**;**

**if(!**CreateShMem**(**shmem**,**temp**))**

exit**(-**1**);**

clients\_count**=(**int**\*)**temp**;**

SemLock**(**shmem\_lock**);**

**(\***clients\_count**)++;**

SemUnlock**(**shmem\_lock**);**

shmdata**=(**char**\*)(**clients\_count**+**1**);**

pthread\_t new\_thread**;**

pthread\_create**(&**new\_thread**,NULL,&**updater**,NULL);**

**}**

MainWindow**::~**MainWindow**()**

**{**

**delete** ui**;**

SemLock**(**shmem\_lock**);**

**(\***clients\_count**)--;**

bool last**=\***clients\_count**==**0**;**

SemUnlock**(**shmem\_lock**);**

**if(**last**){**

shmdt**(**shmdata**);**

shmctl**(**shmem**,**IPC\_RMID**,NULL);**

semctl**(**sem\_notified**,**0**,**IPC\_RMID**);**

semctl**(**shmem\_lock**,**0**,**IPC\_RMID**);**

**}** **else** **{**

shmdt**(**shmdata**);**

**}**

**}**

void MainWindow**::**addMessage**(**char **\***msg**){**

ui**->**textEdit\_2**->**insertHtml**(**QString**::**fromUtf8**(**msg**));**

ui**->**textEdit\_2**->**append**(**QString**(**""**));**

**}**

void MainWindow**::**on\_pushButton\_clicked**()**

**{**

QString msg**=**QString**(**html\_template**).**arg**(**

QTime**::**currentTime**().**toString**(**"HH:mm:ss"**),**

ui**->**lineEdit**->**text**(),**

ui**->**textEdit**->**toHtml**());**

SendMessage**(**msg**.**toUtf8**().**constData**());**

**}**

threadhelper.h:

#ifndef THREADHELPER\_H

#define THREADHELPER\_H

#include <QObject>

class ThreadHelper **:** public QObject

**{**

Q\_OBJECT

public**:**

explicit ThreadHelper**(**QObject **\***parent **=** 0**);**

void emitAddMessage**(**char**\***msg**){**emit addMessage**(**msg**);}**

signals**:**

void addMessage**(**char **\*);**

**};**

#endif // THREADHELPER\_H

threadhelper.cpp:

#include "threadhelper.h"

ThreadHelper**::**ThreadHelper**(**QObject **\***parent**)** **:**

QObject**(**parent**)**

**{**

**}**

# Тестовый пример

На рисунке 1 представлен пример работы программы для локального чата, работающей с разделяемой памятью.

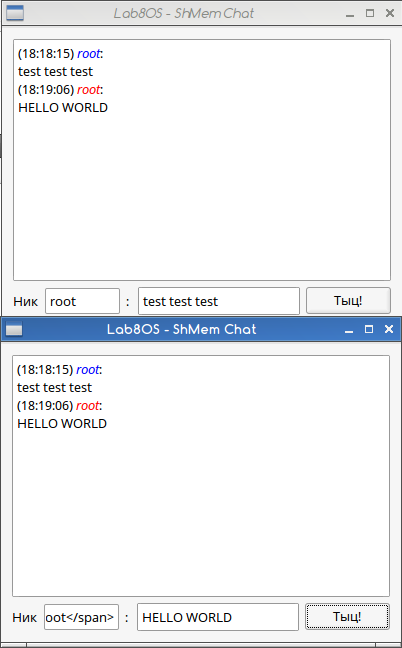


Рисунок 1— Пример работы программ

# Вывод

Linux предлагает средства UNIX System V для межпроцессного взаимодействия. Это очень мощные и чрезвычайно распространенный средства, что позволяет писать сложные приложения для множества систем, не затрачивая усилия для портирования.