Министерство образования и науки РФ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

УПРАВЛЕНИЕ ПРОЦЕССАМИ И ПОТОКАМИ В ОС LINUX

Лабораторная работа № 7
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 изучить инструменты работы с процессами, потоками и их синхронизацией в GNU/Linux.

Задание

Продемонстрировать использование процессов и потоков при решении задачи, согласно варианту: вычисление значений матрицы по правилу M[i,j] = i+j.

Теоретическая справка

В ОС Linux управление процессами является ключевой технологией при разработке многих программ. Процесс - это находящаяся в состоянии выполнения программа вместе со средой ее выполнения.

Так как Linux - это настоящая многозадачная система, в ней одновременно (или квази одновременно, если процессор один) могут выполняться несколько программ (процессов, задач).

Каждый процесс имеет собственное виртуальное адресное пространство. Это необходимо, чтобы гарантировать, что ни один процесс не будет подвержен помехам или влиянию со стороны других.

Отдельные процессы получают доступ к ЦП по очереди. Планировщик процессов решает, как долго и в какой последовательности процессы будут занимать ЦП. При этом создается впечатление, что процессы протекают параллельно.

В Linux реализована вытесняющая многозадачность. Это значит, что система сама решает, как долго конкретный процесс может использовать ЦП, и когда наступит очередь следующего процесса. Если пользователь желает вмешаться в процесс планирования, он может сделать это как root с помощью команды nice.

Системный планировщик использует таблицу процессов, описанную в заголовочном файле /usr/include/linux/sched.h

Внутри структуры struct task_struct находятся все сведения о состоянии процесса. Они достаточно хорошо прокомментированы. Основными являются следующие сведения:

• Идентификация процесса четко устанавливает его права, которые определяются исходя из эффективных или реальных номеров пользователя и номеров групп. Здесь также содержится идентификатор процесса (PID).

• Приоритет процесса определяет очерёдность его выполнения. Каждый процесс имеет в своем распоряжении определенное время для выполнения. Если это время превышено, он должен прервать работу, перейти в состояние неготовности и ждать, пока до него дойдет очередь в следующий раз. По приоритету процесса ядро может решить, какой процесс будет запущен следующим.

• Учетные сведения (accounting) - это информация о возможности получения доступа к определенной области памяти, которая еще не загружена. При этом аппаратура сообщает об отсутствии страницы, после чего ядро занимается загрузкой этой страницы в память.

• Контрольный терминал - каждый процесс, кроме процессов-демонов, нуждается в контрольном терминале, в который выводятся сообщения стандартного ввода / вывода и ошибки.

• Основные характеристики процесса:

- Процесс располагает определенными ресурсами. Он размещен в некотором виртуальном адресном пространстве, содержащем образ этого процесса. Кроме того, процесс управляет другими ресурсами (файлы, устройства ввода / вывода и т.д.).

- Процесс подвержен диспетчеризации. Он определяет порядок выполнения одной или нескольких программ, при этом выполнение может перекрываться другими процессами. Каждый процесс имеет состояние выполнения и приоритет диспетчеризации.

Поток представляет собой облегченную версию процесса. Все потоки процесса разделяют общие ресурсы. Изменения, вызванные одним потоком, становятся немедленно доступны другим.

Важнейшее преимущество потоков перед самостоятельными процессами заключается в том, что накладные расходы на создание нового потока в многопоточном приложении оказываются ниже, чем накладные расходы на

создание нового самостоятельного процесса. Уровень контроля над потоками в многопоточном приложении выше, чем уровень контроля приложения над дочерними процессами.

Тем, кто впервые познакомился с концепцией потоков, изучая программирование для Windows, модель потоков Linux покажется непривычной. В среде Microsoft Windows процесс представляет собой контейнер для потоков (именно этими словами о процессах говорит Джефри Рихтер в своей классической книге «Программирование приложений для Microsoft Windows»). Процесс-контейнер содержит, как минимум, один поток. Если потоков в процессе несколько, приложение (процесс) становится многопоточным. В мире Linux все выглядит иначе. В Linux каждый поток является процессом и для того чтобы создать новый поток, нужно создать новый процесс. В чем же, в таком случае, заключается преимущество многопоточности Linux перед многопроцессностью?

В многопоточных приложениях Linux для создания дополнительных потоков используются процессы особого типа. Эти процессы представляют собой обычные дочерние процессы главного процесса, но они разделяют с главным процессом адресное пространство, файловые дескрипторы и обработчики сигналов. Для обозначения процессов этого типа, применяется специальный термин – легкие процессы (lightweight processes). Прилагательное «легкий» в названии процессов-потоков вполне оправдано. Поскольку этим процессам не нужно создавать собственную копию адресного пространства (и других ресурсов) своего процесса-родителя, создание нового легкого процесса требует значительно меньших затрат, чем создание полновесного дочернего процесса.

Первая подсистема потоков в Linux появилась в 1996 году и называлась, без лишних затей, – LinuxThreads. Рудимент этой подсистемы, который вы найдете в любой современной системе Linux, – файл /usr/include/pthread.h, указывает год релиза – 1996 и имя разработчика – Ксавье Лерой (Xavier Leroy). Библиотека LinuxThreads была попыткой организовать поддержку потоков в Linux в то время, когда ядро системы еще не предоставляло никаких специальных механизмов для работы с потоками. Позднее разработку потоков для Linux вели сразу две конкурирующие группы – NGPT и NPTL. В 2002 году группа NGPT фактически

присоединилась к NPTL и теперь реализация потоков NPTL является стандартом Linux. Подсистема потоков Linux стремится соответствовать требованиям стандартов POSIX, так что новые многопоточные приложения Linux должны без проблем компилироваться в POSIX-совместимых системах.

Инструкция пользователю

Программа позволяет заполнить матрицу – таблицу сложения.

Задайте размеры матрицы и количество потоков, используемых для вычислений. Нажмите кнопку для расчета матрицы. Матрицу можно очистить с помощью соответствующей кнопки.

Инструкция программисту

Типы данных:

Структура – параметры потока.

```
typedef struct{
    int num, - номер потока. Поток заполняет строки матрицы с такими
               номерами, которые при делении на общее число потоков дают
               в остатке номер потока.
    int n,m, - размеры матрицы – n строк, m столбцов.
    int count; - общее количество потоков.
} thread_arg;
```

Переменные:

```
pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;
```

Мьютекс, защищающий рассчитываемую матрицу.

```
double **A;
```

Рассчитываемая матрица.

Функции:

```
void *my_thread(void *arg);
```

Функция-потока. Принимает указатель на thread_arg. Поток заполняет строки матрицы с такими номерами, которые при делении на общее число потоков дают в остатке номер потока.

```
void calcMatrix(int n,int m,int count);
```

Вычисляет матрицу размером n строк на m столбцов с помощью count потоков.

Текст программы

Ниже представлен текст программы для заполнения матрицы, 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 "pthread.h"

#include <QMainWindow>

typedef struct{
    int num,n,m,count;
} thread_arg;

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();

private slots:
    void on_spinBox_valueChanged(int arg1);

    void on_spinBox_2_valueChanged(int arg1);

    void on_pushButton_clicked();

    void on_pushButton_2_clicked();

private:
    Ui::MainWindow *ui;
};

#endif // MAINWINDOW_H
```

mainwindow.cpp:

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <QDebug>
extern "C"{
#include <unistd.h>
}

pthread_mutex_t mutexA = PTHREAD_MUTEX_INITIALIZER;
```

```
double **A;

void *my_thread(void *arg){
    thread_arg* p=(thread_arg*)arg;
    int num=p->num,
        n=p->n,
        m=p->m,
        count=p->count;
    for(int i=num;i<n;i+=count){
        for(int j=0;j<m;++j){
            pthread_mutex_lock(&mutexA);
            A[i][j]=i+j+2;
            pthread_mutex_unlock(&mutexA);
        }
    }

    pthread_exit(NULL);
}

void calcMatrix(int n,int m,int count){
    QList<pthread_t*> mythreads;
    QList<thread_arg*> myargs;
    for(int i=0;i<count;++i){

        thread_arg* arg=new thread_arg;
        arg->count=count;
        arg->n=n;
        arg->m=m;
        arg->num=i;

        pthread_t* new_thread = new pthread_t;

        if(pthread_create(new_thread,NULL,&my_thread,arg)){
                qDebug()<<"Ошибка создания потока!";
                delete arg;
                delete new_thread;
        }
        myargs<<arg;
        mythreads<<new_thread;
    }

    while(mythreads.size()){

        if(pthread_join(*mythreads.first(),NULL))
            qDebug()<<"Ошибка ожидания потока!";

        delete mythreads.first();
        delete myargs.first();
        myargs.removeFirst();
        mythreads.removeFirst();
    }

}

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

```
void MainWindow::on_spinBox_valueChanged(int arg1)
{
    int d=ui->tableWidget->rowCount()-arg1;
    if(d<0){
        for(;d<0;d++){
            int i=ui->tableWidget->rowCount();
            ui->tableWidget->insertRow(i);
            for(int j=0;j<ui->tableWidget->columnCount();++j){
                QTableWidgetItem* item=new QTableWidgetItem;
                ui->tableWidget->setItem(i,j,item);

            }
        }
    } else {
        for(;d>0;d--){
            ui->tableWidget->removeRow(ui->tableWidget->rowCount()-1);
        }
    }

}

void MainWindow::on_spinBox_2_valueChanged(int arg1)
{
    int d=ui->tableWidget->columnCount()-arg1;
    if(d<0){
        for(;d<0;d++){
            int j=ui->tableWidget->columnCount();
            ui->tableWidget->insertColumn(j);
            for(int i=0;i<ui->tableWidget->rowCount();++i){
                QTableWidgetItem* item=new QTableWidgetItem;
                ui->tableWidget->setItem(i,j,item);

            }
        }
    } else {
        for(;d>0;d--){
            ui->tableWidget->removeColumn(ui->tableWidget->columnCount()-1);
        }
    }
}

void MainWindow::on_pushButton_clicked()
{
    int n=ui->tableWidget->rowCount(),
        m=ui->tableWidget->columnCount(),
        count=ui->spinBox_3->value();


    A=new double*[n];
    for(int i=0;i<n;++i){
        A[i]=new double[m];
    }


    calcMatrix(n,m,count);

    for(int i=0;i<n;++i){
        for(int j=0;j<m;++j){
            QTableWidgetItem *item=ui->tableWidget->item(i,j);
            if(item!=NULL)
                item->setText(QString::number(A[i][j]));
        }
    }

    for(int i=0;i<n;++i)
```

```
        delete A[i];
    delete A;
}

void MainWindow::on_pushButton_2_clicked()
{
    int n=ui->tableWidget->rowCount(),
        m=ui->tableWidget->columnCount();
    for(int i=0;i<n;++i){
        for(int j=0;j<m;++j){
            QTableWidgetItem *item=ui->tableWidget->item(i,j);
            if(item!=NULL)
                item->setText(QString());
        }
    }
}
```

Тестовый пример

На рисунке 1 представлен пример работы программы для заполнения матрицы.

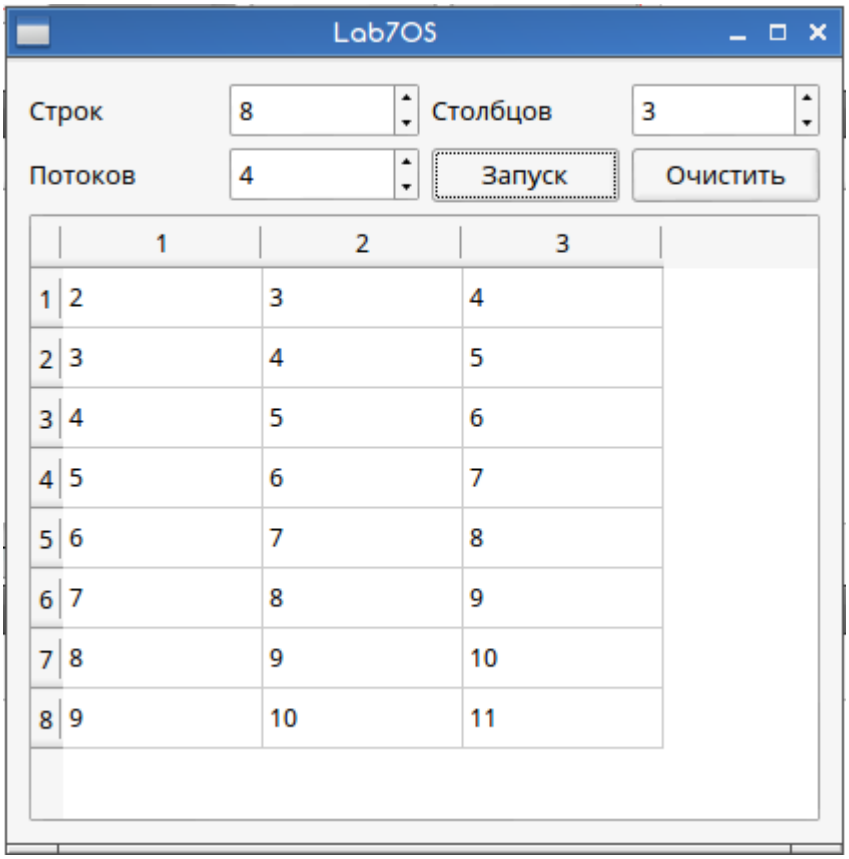

Рисунок 1— Пример работы программы

Вывод

Linux предлагает классические средства POSIX для работы с процессами и потоками. Кроме того, Linux имеет уникальную модель потоков, согласно которой поток является специфическим типом процесса.