Федеральное агентство связи
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М. А. БОНЧ-БРУЕВИЧА»
(СПбГУТ)

Факультет инфокоммуникационных сетей и систем
Кафедра программной инженерии и вычислительной техники

ЛАБОРАТОРНАЯ РАБОТА №10

по дисциплине «Операционные системы и сети»
на тему «Моделирование системы»

Выполнил:
студент 3-го курса
дневного отделения
группы ИКПИ-85
Коваленко Леонид Александрович
Преподаватель:
доцент кафедры ПИиВТ
Дагаев Александр Владимирович

Санкт-Петербург
2020

Цель работы

Разработать программу, моделирующую стратегии замещения страниц в оперативной памяти.

Постановка задачи

Написать программу, моделирующую стратегии замещения страниц в оперативной памяти, и продемонстрировать ее работу.

Теоретическая часть

Работа выполняется в операционной системе Linux Debian.

Страничная память — способ организации виртуальной памяти, при котором единицей отображения виртуальных адресов на физические является регион постоянного размера (т. н. страница). Типичный размер страницы — 4096 байт, для некоторых архитектур — до 128 КБ.

На рис. 1 показана схема страничного распределения памяти. Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами (virtual pages). В общем случае размер виртуального адресного пространства процесса не кратен размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

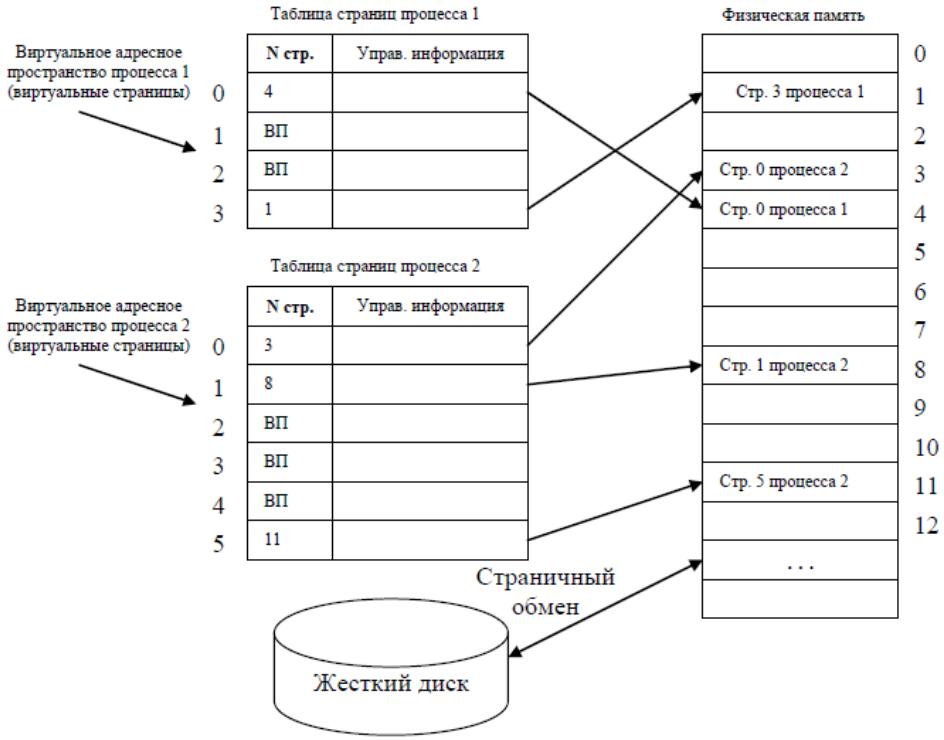

Рисунок 1 — Страничное распределение памяти

Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками, или кадрами). Размер страницы выбирается равным степени двойки: 512, 1024, 4096 байт и т. д. Это позволяет упростить механизм преобразования адресов. Операционная система при создании процесса загружает в оперативную память несколько его виртуальных страниц (начальные страницы кодового сегмента и сегмента данных). Копия всего виртуального адресного пространства процесса находится на диске. Смежные виртуальные страницы не обязательно располагаются в смежных физических страницах. Для каждого процесса операционная система создает таблицу страниц – информационную структуру, содержащую записи обо всех виртуальных страницах процесса.

Запись таблицы, называемая дескриптором страницы, включает следующую информацию:

— номер физической страницы, в которую загружена данная виртуальная страница;

— признак присутствия, устанавливаемый в единицу, если виртуальная страница находится в оперативной памяти;

— признак модификации страницы, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице;

— признак обращения к странице, называемый также битом доступа, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице.

Информация из таблиц страниц используется для решения вопроса о необходимости перемещения той или иной страницы между памятью и диском, а также для преобразования виртуального адреса в физический. При каждом обращении к памяти выполняется поиск номера виртуальной страницы, содержащей требуемый адрес, затем по этому номеру определяется нужный элемент таблицы страниц, и из него извлекается описывающая страницу информация. Далее анализируется признак

присутствия, и, если данная виртуальная страница находится в оперативной памяти, то выполняется преобразование виртуального адреса в физический, то есть виртуальный адрес заменяется указанным в записи таблицы физическим адресом.

Если же нужная виртуальная страница в данный момент выгружена на диск, то происходит так называемое страничное прерывание. Выполняющийся процесс переводится в состояние ожидания, и активизируется другой процесс из очереди процессов, находящихся в состоянии готовности. Параллельно программа обработки страничного прерывания находит на диске требуемую виртуальную страницу (для этого операционная система должна помнить положение вытесненной страницы в страничном файле диска) и пытается загрузить ее в оперативную память. Если в памяти имеется свободная физическая страница, то загрузка выполняется немедленно, если же свободных страниц нет, то на основании принятой в данной системе стратегии замещения страниц решается вопрос о том, какую страницу следует выгрузить из оперативной памяти. При страничной организации памяти есть 2 проблемы: 1) таблица страниц может быть слишком большой; 2) отображение страниц должно быть быстрым.

Конечно же при работе с памятью возникают исключительные ситуации, которые ОС должна как-то обрабатывать. Например нет нужной страницы в памяти или мы пытаемся обратиться к странице, к которой доступ запрещен. В данном случае при такой попытке обращения к виртуальной странице памяти возникает исключительная ситуация именуемая «страничным нарушением» или «отказом страницы» (page fault). Страничное нарушение может происходить в самых разных случаях: при отсутствии страницы в оперативной памяти, при попытке записи в страницу с атрибутом «только чтение» или при попытке чтения или записи страницы с атрибутом «только выполнение». В любом из этих случаев вызывается обработчик страничного нарушения, являющийся частью операционной системы. Ему обычно передается причина возникновения исключительной ситуации и виртуальный адрес, обращение к которому вызвало нарушение.

Все это сказывается на производительности системы, поэтому чтобы повысить эффективность вычислительной системы следует применить различные алгоритмы замещения страниц, где уменьшается количество страничных нарушений.

Есть три задачи:

1. Задача обслуживания исключительной ситуации (отказ страницы);

2. Задача чтения (подкачки) страницы из вторичной памяти (иногда, при недостатке места в основной памяти, необходимо вытолкнуть одну из страниц из основной памяти во вторичную, то есть осуществить замещение страницы);

3. Задача возобновления выполнения процесса, вызвавшего «отказ страницы».

Для решения первой и третьей задач ОС выполняет до нескольких сот машинных инструкций в течение нескольких десятков микросекунд. Время подкачки страницы близко к нескольким десяткам миллисекунд. Проведенные исследования показывают, что вероятности отказа страниц 5×10^{-7} оказывается достаточно, чтобы снизить производительность страничной схемы управления памятью на 10%. Таким образом, уменьшение частоты отказов страниц является одной из ключевых задач системы управления памятью. Ее решение обычно связано с правильным выбором алгоритма замещения страниц.

Алгоритмы замещения страниц

Приведем примеры алгоритмов замещения страниц на неудобных для них входных данных.

FIFO

Алгоритм замены страниц в порядке очереди FIFO (англ. first in, first out — «первым пришёл — первым ушёл») — это самый простой алгоритм замены страниц. В этом алгоритме операционная система отслеживает все страницы в памяти в очереди, самая старая страница находится в начале очереди. Она же и выбирается в первую очередь для удаления.

Рассмотрим пример. Допустим, дана ссылочная строка 1 3 0 3 5 6 и число страниц — 3. Посмотрим, как работает алгоритм FIFO.

Изначально все слоты пусты, поэтому после первых 3-х проходов они будут заполнены {1, 3, 0}. `Page_Fault` = 3.

3: `Frame` = {1, 3, 0}, `Page_Fault` = 3 (без изменений).

5: `Frame` = {3, 0, 5}, `Page_Fault` = 4 (отказ страницы).

6: `Frame` = {0, 5, 6}, `Page_Fault` = 5 (отказ страницы).

Clock (Second-Chance)

При алгоритме «Часы» (или «Второй шанс») страницы-кандидаты на удаление рассматриваются в порядке циклического перебора. Заменяемая страница — та, к которой при циклическом рассмотрении не было доступа с момента ее последнего рассмотрения.

Алгоритм часов хранит в памяти круговой список страниц, при этом «рука» (указатель — `Pointer`) указывает на последнюю рассмотренную страницу в списке. Когда происходит отказ страницы, то бит `R` проверяется в местоположении руки. Если `R` равно 0, новая страница помещается на место страницы, на которую указывает «рука», и рука перемещается на одну позицию. В противном случае бит `R` очищается, затем стрелка часов увеличивается, и процесс повторяется, пока страница не будет заменена.

Допустим, дана ссылочная строка 0 4 1 4 2 4 3 4 2 4 0 4 1 4 2 4 3 4 и число страниц — 3. Посмотрим, как работает алгоритм.

Изначально все слоты пусты, поэтому после первых 3-х проходов они будут заполнены {0, 4, 1}, а массив второго шанса будет {0, 0, 0}.

4: `Frame` = {0, 4, 1}, `Second_Chance` = {0, 1, 0} [4 получил второй шанс], `Pointer` = 0 (обновлять страницу не нужно), `Page_Fault` = 3 (нет увеличения числа ошибок страницы).

2: `Frame` = {2, 4, 1}, `Second_Chance` = {0, 1, 0} [0 заменен на 2; у 0 второго шанса не было], `Pointer` = 1 (обновлено), `Page_Fault` = 4.

4: `Frame` = {2, 4, 1}, `Second_Chance` = {0, 1, 0}, `Pointer` = 1, `Page_Fault` = 4 (без изменений).

3: Frame = {2, 4, 3}, Second_Chance = {0, 0, 0} [4 воспользовался вторым шансом; но у 1 не было: заменен на 3], Pointer = 0 (поскольку третий элемент был заменен, указатель перемещается далее), Page_Fault = 5.

4: Frame = {2, 4, 3}, Second_Chance = {0, 1, 0} [4 получил второй шанс], Pointer = 0, Page_Fault = 5.

2: Frame = {2, 4, 3}, Second_Chance = {1, 1, 0} [2 получил второй шанс], Pointer = 0, Page_Fault = 5.

4: Frame = {2, 4, 3}, Second_Chance = {1, 1, 0}, Pointer = 0, Page_Fault = 5 (без изменений).

0: Frame = {2, 4, 0}, Second_Chance = {0, 0, 0}, Pointer = 0, Page_Fault = 6 (2 и 4 воспользовались вторым шансом).

4: Frame = {2, 4, 0}, Second_Chance = {0, 1, 0}, Pointer = 0, Page_Fault = 6 (4 получил второй шанс).

1: Frame = {1, 4, 0}, Second_Chance = {0, 1, 0}, Pointer = 1, Page_Fault = 7 (Pointer обновлен, Page_Fault обновлен).

4: Frame = {1, 4, 0}, Second_Chance = {0, 1, 0}, Pointer = 1, Page_Fault = 7 (без изменений).

2: Frame = {1, 4, 2}, Second_Chance = {0, 0, 0}, Pointer = 0, Page_Fault = 8 (4 воспользовался вторым шансом).

4: Frame = {1, 4, 2}, Second_Chance = {0, 1, 0}, Pointer = 0, Page_Fault = 8 (4 получил второй шанс).

3: Frame = {3, 4, 2}, Second_Chance = {0, 1, 0}, Pointer = 1, Page_Fault = 9 (Pointer и Page_Fault обновлен).

4: Frame = {3, 4, 2}, Second_Chance = {0, 1, 0}, Pointer = 1, Page_Fault = 9 (без изменений).

LRU

LRU на самом деле является семейством алгоритмов.

Суть алгоритмов LRU (Least Recently Used) заключается в вытеснении давно неиспользуемой страницы.

Допустим, дана ссылочная строка 1 2 3 4 1 2 5 1 2 3 4 5 и число страниц — 3. Посмотрим, как работает алгоритм LRU сначала в прямом и затем в обратном порядке (ReverseLRU).

Изначально все слоты пусты, поэтому после первых 3-х проходов они будут заполнены {1, 2, 3}. `Page_Fault` = 3.

4: `Frame` = {2, 3, 4}, `Page_Fault` = 4 (отказ страницы).

1: `Frame` = {3, 4, 1}, `Page_Fault` = 5 (отказ страницы).

2: `Frame` = {4, 1, 2}, `Page_Fault` = 6 (отказ страницы).

5: `Frame` = {1, 2, 5}, `Page_Fault` = 7 (отказ страницы).

1: `Frame` = {2, 5, 1}, `Page_Fault` = 7 (без изменений).

2: `Frame` = {5, 1, 2}, `Page_Fault` = 7 (без изменений).

3: `Frame` = {1, 2, 3}, `Page_Fault` = 8 (отказ страницы).

4: `Frame` = {2, 3, 4}, `Page_Fault` = 9 (отказ страницы).

5: `Frame` = {3, 4, 5}, `Page_Fault` = 10 (отказ страницы).

Теперь посмотрим, как работает алгоритм ReverseLRU.

Изначально все слоты пусты, поэтому после первых 3-х проходов они будут заполнены {3, 2, 1} (обратный порядок). `Page_Fault` = 3.

4: `Frame` = {4, 3, 2}, `Page_Fault` = 4 (отказ страницы).

1: `Frame` = {1, 4, 3}, `Page_Fault` = 5 (отказ страницы).

2: `Frame` = {2, 1, 4}, `Page_Fault` = 6 (отказ страницы).

5: `Frame` = {5, 2, 1}, `Page_Fault` = 7 (отказ страницы).

1: `Frame` = {1, 5, 2}, `Page_Fault` = 7 (без изменений).

2: `Frame` = {2, 1, 5}, `Page_Fault` = 7 (без изменений).

3: `Frame` = {3, 2, 1}, `Page_Fault` = 8 (отказ страницы).

4: `Frame` = {4, 3, 2}, `Page_Fault` = 9 (отказ страницы).

5: `Frame` = {5, 4, 3}, `Page_Fault` = 10 (отказ страницы).

Оба алгоритма приводят к одному результату (`Page_Fault`), но реализованы по-разному.

LFU

Суть алгоритма LFU (Least Frequently Used) заключается в вытеснении наименее часто используемой страницы.

Допустим, дана ссылочная строка 1 2 3 4 1 2 5 1 2 3 4 5 и число страниц — 3. Посмотрим, как работает алгоритм LFU.

Изначально все слоты пусты, поэтому после первых 3-х проходов они будут заполнены {1, 2, 3}. `Page_Fault` = 3.

4: `Frame` = {2, 3, 4}, `Page_Fault` = 4 (отказ страницы).

1: `Frame` = {1, 3, 4}, `Page_Fault` = 5 (отказ страницы).

2: `Frame` = {2, 3, 4}, `Page_Fault` = 6 (отказ страницы).

5: `Frame` = {3, 4, 5}, `Page_Fault` = 7 (отказ страницы).

1: `Frame` = {1, 4, 5}, `Page_Fault` = 8 (отказ страницы).

2: `Frame` = {2, 4, 5}, `Page_Fault` = 9 (отказ страницы).

3: `Frame` = {3, 4, 5}, `Page_Fault` = 10 (отказ страницы).

4: `Frame` = {3, 5, 4}, `Page_Fault` = 10 (без изменений).

5: `Frame` = {3, 4, 5}, `Page_Fault` = 10 (без изменений).

Практическая часть

Реализуем все рассмотренные ранее алгоритмы при помощи фреймворка Qt (https://www.qt.io/, версия 5.7.1; GCC 6.3.0 от 15.04.2017), используя язык C++.

Код приведен в следующем пункте отчета.

Запустим программу (рис. 1).

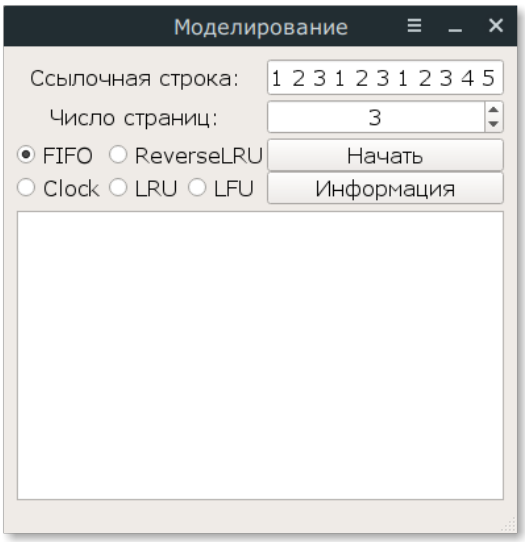

Рисунок 1 — Первый запуск

Будем проверять работу на ссылочной строке «1 2 3 4 1 2 5 1 2 3 4 5» и при числе страниц, равным 3.

Ссылочная строка может содержать только числа, разделенные пробелами, запятыми, точками с запятыми и точками. Т. е. выбранная нами ссылочная строка может выглядеть как «1, 2; 3, 4, 1; 2; 5. 1, 2,;,;,3 ;,;.;.; 4.,.5».

Проверим FIFO (рис. 2).

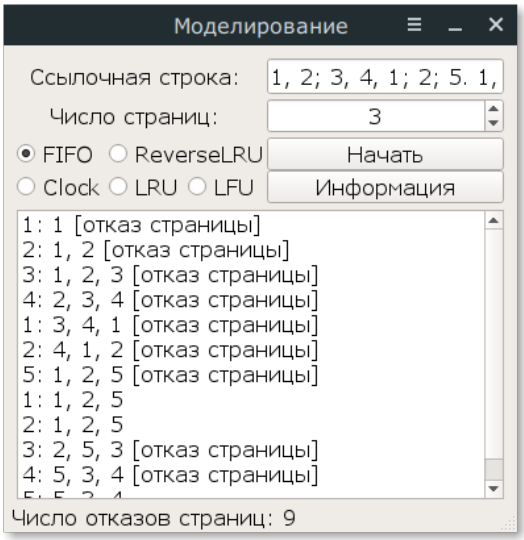

Рисунок 2 — Работа алгоритма FIFO при трех страницах

Проверим Clock (рис. 3).

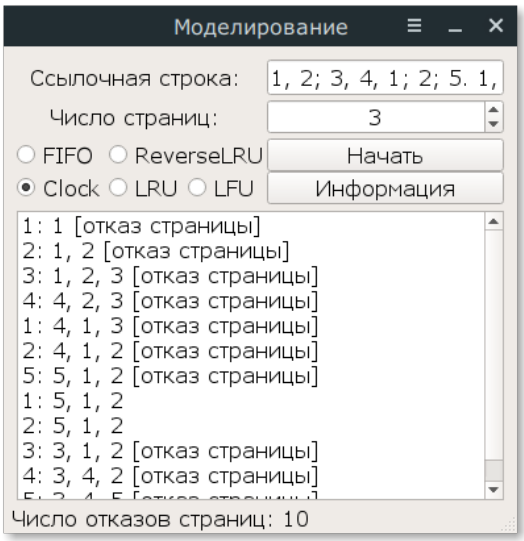

Рисунок 3 — Работа алгоритма Clock при трех страницах

Проверим ReverseLRU (рис. 4).

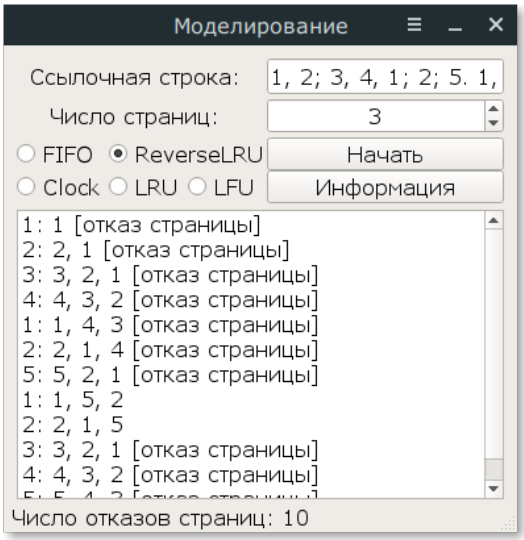

Рисунок 4 — Работа алгоритма ReverseLRU при трех страницах

Проверим LRU (рис. 5).

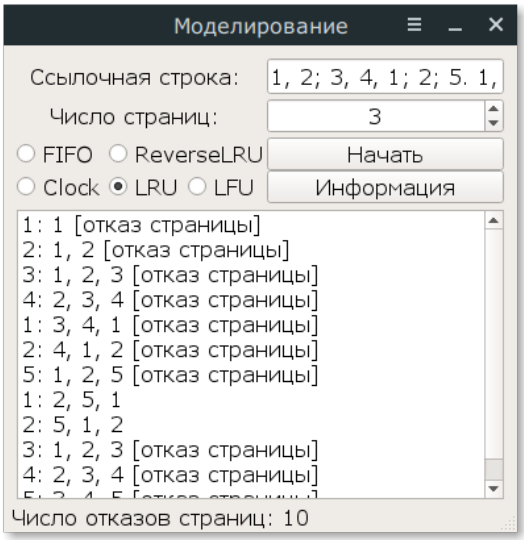

Рисунок 5 — Работа алгоритма LRU при трех страницах

Проверим LFU (рис. 6).

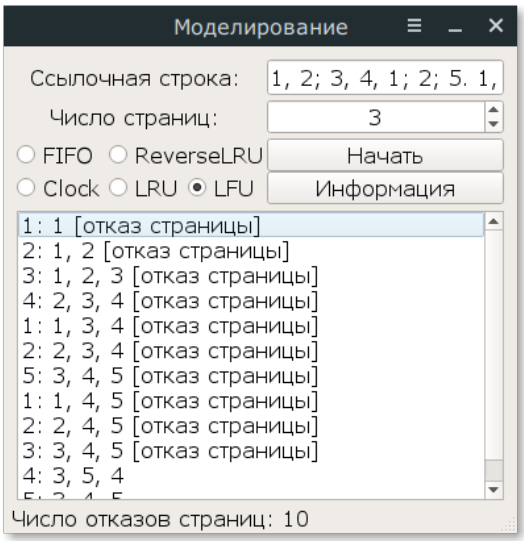

Рисунок 6 — Работа алгоритма LFU при трех страницах

Попробуем теперь проверять работу на ссылочной строке «1 2 3 4 1 2 5 1 2 3 4 5» и при числе страниц, равным 4.

Проверим FIFO (рис. 7).

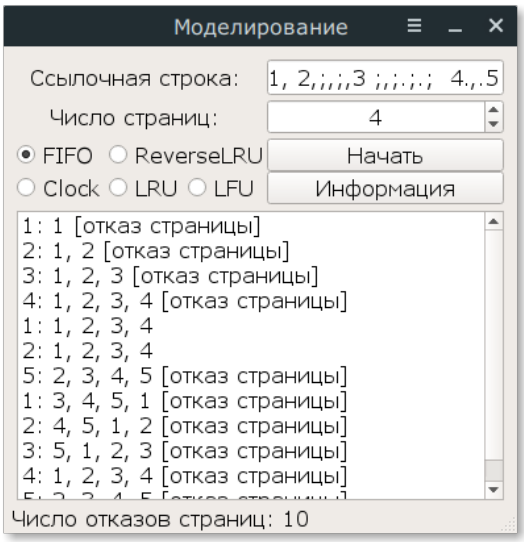

Рисунок 7 — Работа алгоритма FIFO при четырех страницах

Проверим Clock (рис. 8).

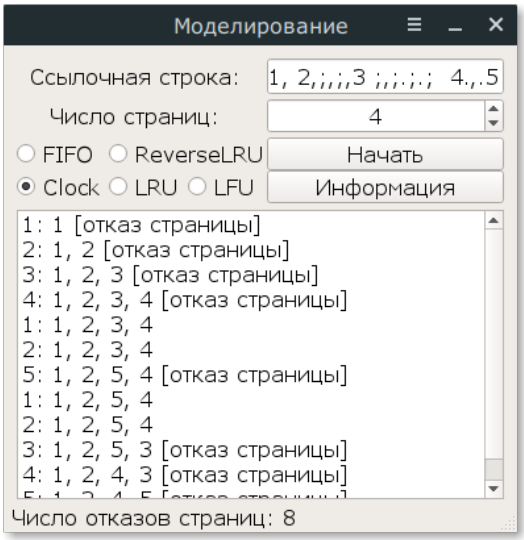

Рисунок 8 — Работа алгоритма Clock при четырех страницах

Проверим ReverseLRU (рис. 9).

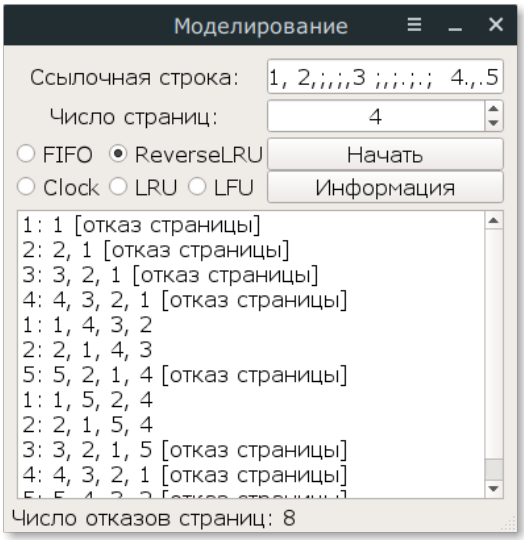

Рисунок 9 — Работа алгоритма ReverseLRU при четырех страницах

Проверим LRU (рис. 10).

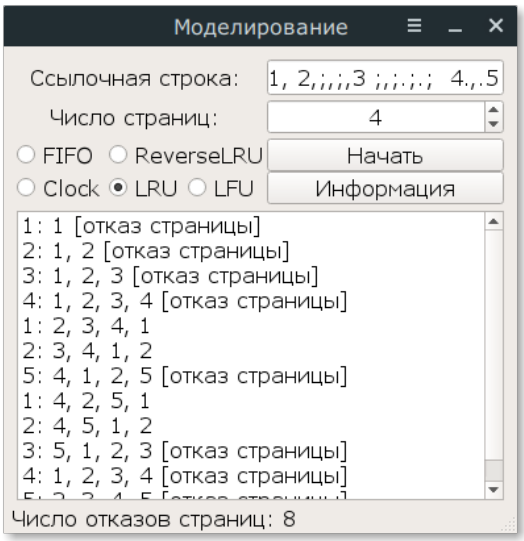

Рисунок 10 — Работа алгоритма LRU при четырех страницах

Проверим LFU (рис. 11).

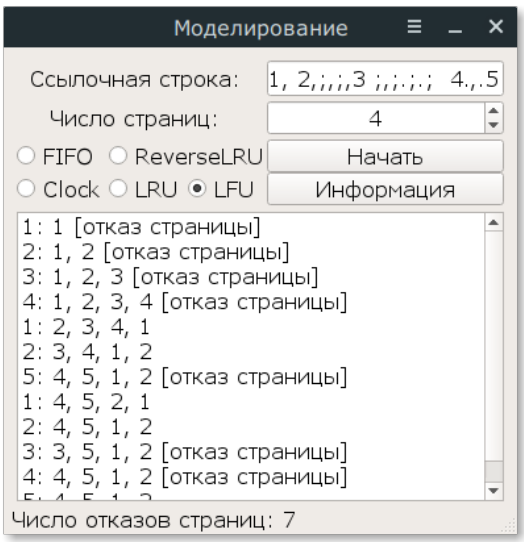

Рисунок 11 — Работа алгоритма LFU при четырех страницах

По нажатию на кнопку «Информация» можно узнать информацию о программе (рис. 12).

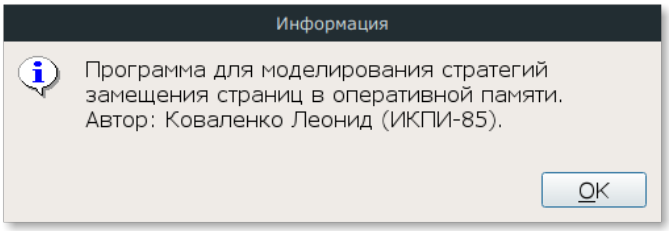

Рисунок 12 — Информация о программе

Если пользователь некорректно ввел данные в ссылочную строку и нажал на кнопку «Начать», отображается сообщение об ошибке (рис. 13).

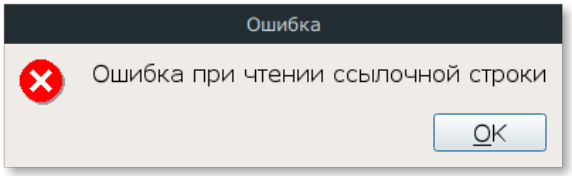

Рисунок 13 — Информация об ошибке чтения

Код программы

Таблица 1 — Файл `Paging.pro`

```
#-------------------------------------------------
#
# Project created by QtCreator 2020-11-12T20:43:06
#
#-------------------------------------------------

QT += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = Paging
TEMPLATE = app

# The following define makes your compiler emit warnings if you use
# any feature of Qt which as been marked as deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if you use deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version
of Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs
deprecated before Qt 6.0.0

SOURCES += main.cpp\
        mainwindow.cpp \
    fifo_paging.cpp \
    lru_paging.cpp \
    clock_paging.cpp \
    lfu_paging.cpp \
    reverselru_paging.cpp
```

```
HEADERS += mainwindow.h \
    abstract_paging.h \
    fifo_paging.h \
    lru_paging.h \
    clock_paging.h \
    lfu_paging.h \
    reverselru_paging.h

FORMS += mainwindow.ui
```

Таблица 2 — Файл `abstract_paging.h`

```
#ifndef ABSTRACT_PAGING_H
#define ABSTRACT_PAGING_H

#include <list>
#include <stdexcept>

// Абстрактный класс замещения страниц
class AbstractPaging {
  public:
    // Конструктор с параметром
    AbstractPaging(int pageCount) : mPageCount(pageCount) {
        if (pageCount <= 0)
            throw std::invalid_argument("FrameCount <= 0");
    }
    // Передача на рассмотрение страницы page
    virtual void refer(int page) = 0;
    // Получение списка страниц
    virtual const std::list<int> getList() = 0;
    // Получение числа отказов страниц
    virtual int getPageFaultCount() { return mPageFault; }
    // Очистка данных, возврат к начальному состоянию
    virtual void clear() = 0;

  protected:
    // Максимальное число страниц в памяти
    int mPageCount;
    // Текущее число отказов страниц
    int mPageFault = 0;
};

#endif // ABSTRACT_PAGING_H
```

Таблица 3 — Файл `clock_paging.h`

```
#ifndef CLOCK_PAGING_H
#define CLOCK_PAGING_H

#include "abstract_paging.h"
#include <vector>

// Конкретный класс замещения страниц Clock
class ClockPaging : public AbstractPaging {
  public:
    // Конструктор с параметром
    ClockPaging(int pageCount);
    // Передача на рассмотрение страницы page
    void refer(int page) override;
    // Получение списка страниц
    const std::list<int> getList() override;
    // Очистка данных, возврат к начальному состоянию
    void clear() override;
```

```
  private:
    // Текущий индекс элемента
    int mCurrent = 0;
    // Вектор страниц
    std::vector<int> mPages;
    // Вектор состояний страниц
    std::vector<bool> mBits;
};

#endif // CLOCK_PAGING_H
```

Таблица 4 — Файл `clock_paging.cpp`

```
#include "clock_paging.h"
#include <algorithm>

// Конструктор с параметром
ClockPaging::ClockPaging(int pageCount) : AbstractPaging(pageCount) {}

// Передача на рассмотрение страницы page
void ClockPaging::refer(int page) {
    auto duplicate = std::find(mPages.begin(), mPages.end(), page);
    if (duplicate != mPages.end()) {
        mBits[duplicate - mPages.begin()] = true;
        return;
    }
    ++mPageFault;
    if ((long)mPages.size() != (long)mPageCount) {
        mPages.emplace_back(page);
        mBits.emplace_back(false);
        return;
    }
    while (true) {
        if (!mBits[mCurrent]) {
            mPages[mCurrent] = page;
            mCurrent = (mCurrent + 1) % mPageCount;
            return;
        }
        mBits[mCurrent] = false;
        mCurrent = (mCurrent + 1) % mPageCount;
    }
}

// Получение списка страниц
const std::list<int> ClockPaging::getList() {
    return std::list<int>(mPages.begin(), mPages.end());
}

// Очистка данных, возврат к начальному состоянию
void ClockPaging::clear() {
    mPageFault = 0;
    mCurrent = 0;
    mPages.clear();
    mBits.clear();
}
```

Таблица 5 — Файл `fifo_paging.h`

```
#ifndef FIFO_PAGING_H
#define FIFO_PAGING_H

#include "abstract_paging.h"

class FifoPaging : public AbstractPaging {
```

```
  public:
    // Конструктор с параметром
    FifoPaging(int pageCount);
    // Передача на рассмотрение страницы page
    void refer(int page) override;
    // Получение списка страниц
    const std::list<int> getList() override;
    // Очистка данных, возврат к начальному состоянию
    void clear() override;

  private:
    // Дек страниц
    std::list<int> mPages;
};

#endif // FIFO_PAGING_H
```

Таблица 6 — Файл `fifo_paging.cpp`

```
#include "fifo_paging.h"
#include <algorithm>

// Конструктор с параметром
FifoPaging::FifoPaging(int pageCount) : AbstractPaging(pageCount) {}

// Передача на рассмотрение страницы page
void FifoPaging::refer(int page) {
    if (std::find(mPages.begin(), mPages.end(), page) == mPages.end()) {
        ++mPageFault;
        if ((long)mPages.size() == (long)mPageCount) {
            mPages.pop_front();
        }
        mPages.emplace_back(page);
    }
}

// Получение списка страниц
const std::list<int> FifoPaging::getList() { return mPages; }

// Очистка данных, возврат к начальному состоянию
void FifoPaging::clear() {
    mPageFault = 0;
    mPages.clear();
}
```

Таблица 7 — Файл `lfu_paging.h`

```
#ifndef LFU_PAGING_H
#define LFU_PAGING_H

#include "abstract_paging.h"
#include <set>

class LfuPaging : public AbstractPaging {
  public:
    // Конструктор с параметром
    LfuPaging(int pageCount);
    // Передача на рассмотрение страницы page
    void refer(int page) override;
    // Получение списка страниц
    const std::list<int> getList() override;
    // Очистка данных, возврат к начальному состоянию
    void clear() override;
```

```
  private:
    // Упорядоченное множество страниц
    std::set<std::pair<int, int>> mPages;
};

#endif // LFU_PAGING_H
```

Таблица 8 — Файл `lfu_paging.cpp`

```
#include "lfu_paging.h"
#include <algorithm>

// Конструктор с параметром
LfuPaging::LfuPaging(int pageCount) : AbstractPaging(pageCount) {}

// Передача на рассмотрение страницы page
void LfuPaging::refer(int page) {
    auto duplicate =
        std::find_if(mPages.begin(), mPages.end(),
                     [&page](auto pair) { return pair.second == page; });
    if (duplicate == mPages.end()) {
        ++mPageFault;
        if ((long)mPages.size() == (long)mPageCount) {
            mPages.erase(*mPages.begin());
        }
        mPages.insert({0, page});
    } else {
        int t = duplicate->first;
        mPages.erase(duplicate);
        mPages.insert({t + 1, page});
    }
}

// Получение списка страниц
const std::list<int> LfuPaging::getList() {
    std::list<int> list;
    for (auto &pair : mPages) {
        list.emplace_back(pair.second);
    }
    return list;
}

// Очистка данных, возврат к начальному состоянию
void LfuPaging::clear() {
    mPageFault = 0;
    mPages.clear();
}
```

Таблица 9 — Файл `lru_paging.h`

```
#ifndef LRU_PAGING_H
#define LRU_PAGING_H

#include "abstract_paging.h"

class LruPaging : public AbstractPaging {
  public:
    // Конструктор с параметром
    LruPaging(int pageCount);
    // Передача на рассмотрение страницы page
    void refer(int page) override;
    // Получение списка страниц
    const std::list<int> getList() override;
    // Очистка данных, возврат к начальному состоянию
```

```
    void clear() override;

  private:
    // Список страниц
    std::list<int> mPages;
};

#endif // LRU_PAGING_H
```

Таблица 10 — Файл `lru_paging.cpp`

```
#include "lru_paging.h"
#include <algorithm>

// Конструктор с параметром
LruPaging::LruPaging(int pageCount) : AbstractPaging(pageCount) {}

// Передача на рассмотрение страницы page
void LruPaging::refer(int page) {
    auto duplicate = std::find(mPages.begin(), mPages.end(), page);
    if (duplicate == mPages.end()) {
        ++mPageFault;
        if ((long)mPages.size() == (long)mPageCount) {
            mPages.pop_front();
        }
        mPages.emplace_back(page);
    } else {
        mPages.remove(page);
        mPages.emplace_back(page);
    }
}

// Получение списка страниц
const std::list<int> LruPaging::getList() { return mPages; }

// Очистка данных, возврат к начальному состоянию
void LruPaging::clear() {
    mPageFault = 0;
    mPages.clear();
}
```

Таблица 11 — Файл `reverselru_paging.h`

```
#ifndef REVERSELRU_PAGING_H
#define REVERSELRU_PAGING_H

#include "abstract_paging.h"
#include <unordered_map>

class ReverseLruPaging : public AbstractPaging {
  public:
    // Конструктор с параметром
    ReverseLruPaging(int pageCount);
    // Передача на рассмотрение страницы page
    void refer(int page) override;
    // Получение списка страниц
    const std::list<int> getList() override;
    // Очистка данных, возврат к начальному состоянию
    void clear() override;

  private:
    // Список страниц
    std::list<int> mPages;
    // Ассоциативный массив (страница, положение в списке)
```

```
    std::unordered_map<int, std::list<int>::iterator> mRefMap;
};

#endif // REVERSELRU_PAGING_H
```

Таблица 12 — Файл reverselru_paging.cpp

```
#include "reverselru_paging.h"

// Конструктор с параметром
ReverseLruPaging::ReverseLruPaging(int pageCount) : AbstractPaging(pageCount)
{}

// Передача на рассмотрение страницы page
void ReverseLruPaging::refer(int page) {
    if (mRefMap.find(page) == mRefMap.end()) {
        ++mPageFault;
        if ((long)mPages.size() == (long)mPageCount) {
            mRefMap.erase(mPages.back());
            mPages.pop_back();
        }
    } else {
        mPages.erase(mRefMap[page]);
    }
    mPages.emplace_front(page);
    mRefMap[page] = mPages.begin();
}

// Получение списка страниц
const std::list<int> ReverseLruPaging::getList() { return mPages; }

// Очистка данных, возврат к начальному состоянию
void ReverseLruPaging::clear() {
    mPageFault = 0;
    mPages.clear();
    mRefMap.clear();
}
```

Таблица 13 — Файл 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 on_startButton_clicked();
    void on_infoButton_clicked();

  private:
    Ui::MainWindow *ui;
};
```

```
#endif // MAINWINDOW_H
```

Таблица 14 — Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "abstract_paging.h"
#include "clock_paging.h"
#include "fifo_paging.h"
#include "lfu_paging.h"
#include "lru_paging.h"
#include "reverselru_paging.h"
#include "ui_mainwindow.h"
#include <QMessageBox>
#include <memory>

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

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

void MainWindow::on_startButton_clicked() {
    ui->statusBar->showMessage("");
    std::unique_ptr<AbstractPaging> paging;
    if (ui->clockButton->isChecked()) {
        paging = std::unique_ptr<AbstractPaging>(
            new ClockPaging(ui->frameCountSpin->value()));
    } else if (ui->fifoButton->isChecked()) {
        paging = std::unique_ptr<AbstractPaging>(
            new FifoPaging(ui->frameCountSpin->value()));
    } else if (ui->lfuButton->isChecked()) {
        paging = std::unique_ptr<AbstractPaging>(
            new LfuPaging(ui->frameCountSpin->value()));
    } else if (ui->lruButton->isChecked()) {
        paging = std::unique_ptr<AbstractPaging>(
            new LruPaging(ui->frameCountSpin->value()));
    } else {
        paging = std::unique_ptr<AbstractPaging>(
            new ReverseLruPaging(ui->frameCountSpin->value()));
    }
    QStringList stringList = ui->referLine->text().split(QRegExp("[.,;\\s]
+"));
    ui->resultWidget->clear();
    for (QString string : stringList) {
        bool isOk = true;
        int page = string.toInt(&isOk);
        if (isOk == false) {
            QMessageBox::critical(this, "Ошибка",
                                  "Ошибка при чтении ссылочной строки");
            ui->resultWidget->clear();
            return;
        }
        int pageFaultCountBefore = paging->getPageFaultCount();
        paging->refer(page);
        int pageFaultCountAfter = paging->getPageFaultCount();
        QString resultLine = QString::number(page) + ": ";
        const std::list<int> list = paging->getList();
        for (auto page : list) {
            resultLine += QString::number(page) + ", ";
        }
        resultLine.chop(2);
        if (pageFaultCountBefore != pageFaultCountAfter) {
            resultLine += " [отказ страницы]";
        }
```

```
        ui->resultWidget->addItem(resultLine);
    };
    ui->statusBar->showMessage("Число отказов страниц: " +
                               QString::number(paging->getPageFaultCount()));
}

void MainWindow::on_infoButton_clicked() {
    QMessageBox::information(this, "Информация",
                             "Программа для моделирования стратегий замещения"
                             "страниц в оперативной памяти.\nАвтор: Коваленко"
                              "Леонид (ИКПИ-85).\n");
}
```

Таблица 15 — Файл `main.cpp`

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Таблица 16 — Файл `mainwindow.ui`

```
<?xml version="1.0" encoding="UTF-8"?><ui
version="4.0"><class>MainWindow</class><widget class="QMainWindow"
name="MainWindow"><property
name="geometry"><rect><x>0</x><y>0</y><width>390</width><height>370</
height></rect></property><property name="sizePolicy"><sizepolicy
hsizetype="Expanding"
vsizetype="Expanding"><horstretch>0</horstretch><verstretch>0</verstretch></
sizepolicy></property><property
name="minimumSize"><size><width>390</width><height>370</height></size></
property><property
name="maximumSize"><size><width>390</width><height>370</height></size></
property><property
name="windowTitle"><string>Моделирование</string></property><widget
class="QWidget" name="centralWidget"><widget class="QListWidget"
name="resultWidget"><property
name="geometry"><rect><x>10</x><y>125</y><width>370</width><height>220</
height></rect></property><property
name="editTriggers"><set>QAbstractItemView::NoEditTriggers</set></
property><property
name="defaultDropAction"><enum>Qt::IgnoreAction</enum></property></
widget><widget class="QLineEdit" name="referLine"><property
name="geometry"><rect><x>200</x><y>10</y><width>180</width><height>27</
height></rect></property><property name="text"><string>1 2 3 1 2 3 1 2 3 4
5</string></property></widget><widget class="QLabel" name="label"><property
name="geometry"><rect><x>10</x><y>13</y><width>180</width><height>20</
height></rect></property><property name="text"><string>Ссылочная
строка:</string></property><property
name="alignment"><set>Qt::AlignCenter</set></property></widget><widget
class="QRadioButton" name="fifoButton"><property
name="geometry"><rect><x>10</x><y>70</y><width>70</width><height>25</
height></rect></property><property
name="text"><string>FIFO</string></property><property
name="checked"><bool>true</bool></property></widget><widget
class="QRadioButton" name="reverseLRUButton"><property
name="geometry"><rect><x>80</x><y>70</y><width>121</width><height>25</
height></rect></property><property
name="text"><string>ReverseLRU</string></property></widget><widget
```

```
class="QPushButton" name="startButton"><property
name="geometry"><rect><x>200</x><y>70</y><width>180</width><height>25</
height></rect></property><property
name="text"><string>Начать</string></property></widget><widget class="QLabel"
name="label_2"><property
name="geometry"><rect><x>10</x><y>43</y><width>180</width><height>20</
height></rect></property><property name="text"><string>Число
страниц:</string></property><property
name="alignment"><set>Qt::AlignCenter</set></property></widget><widget
class="QSpinBox" name="frameCountSpin"><property
name="geometry"><rect><x>200</x><y>40</y><width>180</width><height>27</
height></rect></property><property
name="alignment"><set>Qt::AlignCenter</set></property><property
name="minimum"><number>1</number></property><property
name="maximum"><number>1000</number></property><property
name="singleStep"><number>1</number></property><property
name="value"><number>3</number></property></widget><widget
class="QRadioButton" name="clockButton"><property
name="geometry"><rect><x>10</x><y>95</y><width>70</width><height>25</
height></rect></property><property
name="text"><string>Clock</string></property><property
name="checked"><bool>false</bool></property></widget><widget
class="QRadioButton" name="lfuButton"><property
name="geometry"><rect><x>140</x><y>95</y><width>60</width><height>25</
height></rect></property><property
name="text"><string>LFU</string></property></widget><widget
class="QRadioButton" name="lruButton"><property
name="geometry"><rect><x>80</x><y>95</y><width>60</width><height>25</
height></rect></property><property
name="text"><string>LRU</string></property></widget><widget
class="QPushButton" name="infoButton"><property
name="geometry"><rect><x>200</x><y>95</y><width>180</width><height>25</
height></rect></property><property
name="text"><string>Информация</string></property></widget></widget><widget
class="QStatusBar" name="statusBar"/></widget><layoutdefault spacing="6"
margin="11"/><resources/><connections/></ui>
```

Заключение

В результате выполнения лабораторной работы мы разработали программу, моделирующую стратегии замещения страниц в оперативной памяти.