Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 **Информатика и вычислительная техника**

О Т Ч Е Т

по практикуму № 1

Дисциплина: Организация ЭВМ и систем

Название: Разработка и отладка программ в вычислительном комплексе Тераграф

Преподаватель _____ Ибрагимов
(Подпись, дата) (И.О. Фамилия)

Студент гр. ИУ7-52Б _____ Фам М.Х.
(Подпись, дата) (И.О. Фамилия)

Москва, 2023

Цель работы: Практикум посвящен освоению принципов работы вычислительного комплекса Тераграф и получению практических навыков решения задач обработки множеств на основе гетерогенной вычислительной структуры. В ходе практикума необходимо ознакомиться с типовой структурой двух взаимодействующих программ: хост-подсистемы и программного ядра sw_kernel.

Ход работы

Запуск демо проекта 1

```
iu7049@dl580:~/lab1$ host/host_main sw-kernel/sw_kernel.rawbinary
Open gpc on /dev/gpc2
Rawbinary loaded from sw-kernel/sw_kernel.rawbinary
sw_kernel version: 0x28102023
Leonhard clock frequency (LNH_CF) 180.036194 MHz
Test done
```

Рисунок 1 – Результат запуска проекта 1

Пример демонстрирует основные механизмы инициализации гетерогенных ядер gpc и взаимодействие хост-подсистем с Graph Processor Core, используются аппаратные очереди.

Запуск демо проекта 2

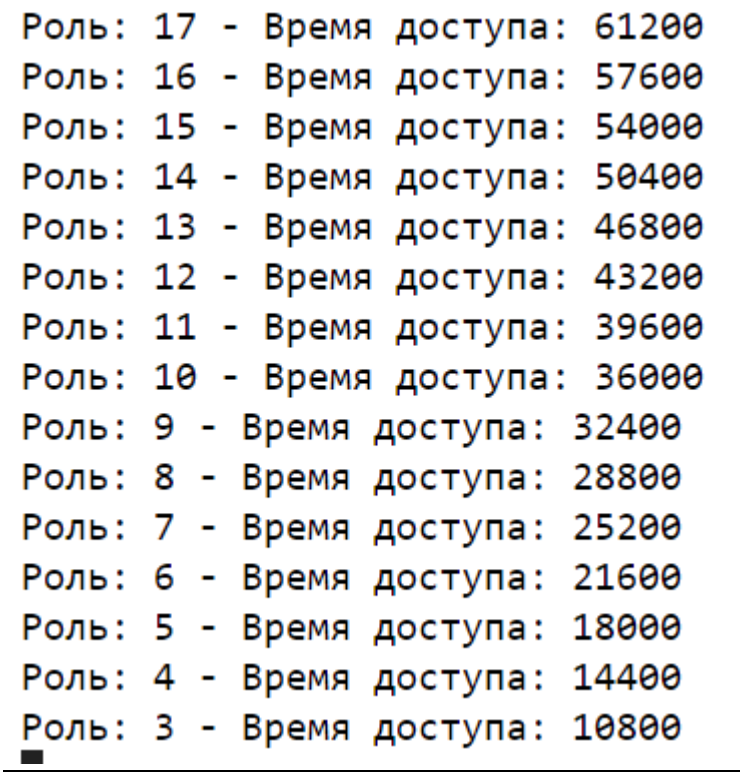

Рисунок 2 – Результат запуска проекта 2

Программа демонстрирует принципы взаимодействия устройств в системе и примеры хранения и обработки множеств в микропроцессоре Lnh64.

Запуск демо проекта 3

```
iu7049@d1580:~/lab3$ host/host_main sw-kernel/sw_kernel.rawbinary
 Открывается доступ к /dev/gpc3
 Программное ядро загружено из файла sw-kernel/sw_kernel.rawbinary
 Ядро /dev/gpc3. Эксперимент 1 - тест вставки и поиска. Ошибок: 0
 Ядро /dev/gpc3. Эксперимент 2 - тест вставки в заполненную ассоциативную память. Ошибок: 0
 Ядро /dev/gpc3. Эксперимент 3 - тест команд ближайшего меньшего и большего (nsm и ngr). Ошибок: 0
 Ядро /dev/gpc3. Эксперимент 4 - тест вставки и удаления случайных ключей. Ошибок: 0
 Ядро /dev/gpc3. Эксперимент 5 - тест команды NOT (дополнение). Ошибок: 0
 Ядро /dev/gpc3. Эксперимент 6 - тест команды AND (пересечение). Ошибок: 0
 Ядро /dev/gpc3. Эксперимент 7 - тест команды OR (объединение). Ошибок: 0
 Ядро /dev/gpc3. Эксперимент 8 - тест пересечения множеств на случайных ключах. Ошибок: 0
 Ядро /dev/gpc3. Эксперимент 9 - тест дополнения множеств на случайных ключах. Ошибок: 0
 Ядро /dev/gpc3. Эксперимент 10 - тест объединения множеств на случайных ключах. Ошибок: 0
```

Рисунок 3 – Результат запуска проекта 3

Пример демонстрирует основные механизмы взаимодействия микропроцессоров CPE (resv64) и SPE (lnh64), и выполняет тестирование корректности команд DISC.

Индивидуальные задания

Вариант 19:

Коммутатор с QoS. Сформировать в хост-подсистеме и передать в SPE таблицу коммутации из 32 ip адресов 195.19.32.1/24 (адреса 195.19.32.1 .. 195.19.32.32), где для каждого адреса доступны 8 вариантов интерфейсов. Вариант определяется по уровню QoS, принимающему значения от 0 до 7 (в таблице коммутации 256 записей). Выполнить тестирование работы коммутатора, посылая из хост-подсистемы уровень QoS и ip адрес, и сравнивая полученный от GPC номер интерфейса с ожидаемым.

Основные программы:

common_struct.h:

```
#ifndef COMMON_STRUCT
#define COMMON_STRUCT

#ifdef __riscv64__
#include "map.h"
#endif
#include "compose_keys.hxx"

//Номера структур данных в SPE
enum Structures : uint32_t {
    null            = 0,    //Нулевая структура не используется
    users_pnum      = 1,    //Таблица 1
    resources_pnum  = 2     //Таблица 2
};

#ifdef __riscv64__
//Задание даипазонов и курсоров
template<typename Range>
struct reverse {
        Range r;
        [[gnu::always_inline]] reverse(Range r) : r(r) {}
        [[gnu::always_inline]] auto begin() {return r.rbegin();}
        [[gnu::always_inline]] auto end() {return r.rend();}
};

template<typename K, typename V>
struct Handle {
        bool ret_val;
        K k{get_result_key<K>()};
        V v{get_result_value<V>()};
        [[gnu::always_inline]] Handle(bool ret_val) : ret_val(ret_val) {
        }

        [[gnu::always_inline]] operator bool() const {
                return ret_val;
        }
```

```
        [[gnu::always_inline]] K key() const {
                return k;
        }

        [[gnu::always_inline]] V value() const {
                return v;
        }
};
#endif


//////////////////////////////////////
// Описание формата ключа и значения
//////////////////////////////////////


struct users {
    using vertex_t = uint32_t;
    int struct_number;
    constexpr users(int struct_number) : struct_number(struct_number) {}
    static const uint32_t idx_bits = 32;
    static const uint32_t idx_max = (1ull << idx_bits) - 1;
    static const uint32_t idx_min = idx_max;

    //Запись для формирования ключей (* - наиболее значимые биты поля)
    STRUCT(key)
    {
        uint32_t    ip    :32;      //Поле 1*
        uint32_t    qos     :idx_bits;  //Поле 0:
    };

    //Запись для формирования значений
    STRUCT(val)
    {
        uint64_t    qos :64;        //Поле 0:
    };
    //Обязательная типизация
    #ifdef __riscv64__
    DEFINE_DEFAULT_KEYVAL(key, val)
    #endif
};

constexpr users USERS(Structures::users_pnum);

#endif //COMMON_STRUCT
```

host_main.cpp:

```
#include <iostream>
#include <iterator>
#include <string>
```

```
#include <regex>
#include <sstream>
#include <fstream>
#include <ctime>
#include "host_main.h"

using namespace std;

uint64_t MAX_IP = 256;

int main(int argc, char** argv)
{
    ofstream log("lab2.log"); //поток вывода сообщений
    gpc *gpc64_inst; //указатель на класс gpc

    //Инициализация gpc
    if (argc<2) {
        log<<"Использование: host_main <путь к файлу rawbinary>"<<endl;
        return -1;
    }

    //Захват ядра gpc и запись sw_kernel
    gpc64_inst = new gpc();
    log<<"Открывается доступ к "<<gpc64_inst->gpc_dev_path<<endl;
    if (gpc64_inst->load_swk(argv[1])==0) {
        log<<"Программное ядро загружено из файла "<<argv[1]<<endl;
    }
    else {
        log<<"Ошибка загрузки sw_kernel файла << argv[1]"<<endl;
        return -1;
    }
    gpc64_inst->start(__event__(update)); //обработчик вставки

    for (uint32_t ip = 1; ip <= 32; ip++)
    {
        for (uint32_t qos = 0; qos <= 7; qos++)
        {
            gpc64_inst->mq_send(users::key{.ip=195193200 + ip,.qos=qos});
//запись о роли #idx
            gpc64_inst->mq_send(users::val{.qos = static_cast<uint64_t>(qos)});
//роль и время доступа
        }
    }

    gpc64_inst->mq_send(-1ull);

    gpc64_inst->start(__event__(find_in_table));
    uint32_t ip[4], qos;
    cout << "Enter ip address: " << endl;
    scanf("%u.%u.%u.%u", ip, ip + 1, ip + 2, ip + 3);
    cout << "Enter QoS level: " << endl;
```

```
    scanf("%u", &qos);
    uint32_t input_data = ((ip[0] * 100 + ip[1]) * 100 + ip[2]) * 100 + ip[3];
    gpc64_inst->mq_send(users::key{.ip=input_data,.qos=qos});

    cout << "Ожидаемый интерфейс: " << qos << endl;
    uint64_t rev = gpc64_inst->mq_receive();
    cout << "Полученный от GPC: " << rev << endl;
    if (qos == rev)
        cout << "Полученный и ожидаемый совпадают" << endl;
    else
        cout << "Не совпадало :(" << endl;
    return 0;
}
}
```

sw_kernel_main.cpp:

```
#include <stdlib.h>
#include <ctime>
#include "lnh64.hxx"
#include "gpc_io_swk.h"
#include "gpc_handlers.h"
// #include "iterators.h"
#include "common_struct.h"
#include "compose_keys.hxx"

#define __fast_recall__

#define MY_TABLE 1
#define MAX_RECORD 256

extern lnh lnh_core;
volatile unsigned int event_source;

int main(void) {
    ////////////////////////////////////////////////////////
    //                  Main Event Loop
    ////////////////////////////////////////////////////////
    //Leonhard driver structure should be initialised
    lnh_init();
    for (;;) {
        //Wait for event
        event_source = wait_event();
        switch(event_source) {
            ////////////////////////////////////////////
            //  Measure GPN operation frequency
            ////////////////////////////////////////////
            case __event__(update) : update(); break;
            case __event__(find_in_table) : find_in_table(); break;
        }
        set_gpc_state(READY);
    }
```

```
}

//-------------------------------------------------------------
//      Вставка ключа и значения в структуру
//-------------------------------------------------------------

void update()
{
    while(1){
            users::key key=users::key::from_int(mq_receive());
            if (key==-1ull) break;
            users::val val=users::val::from_int(mq_receive());
            USERS.ins_async(key,val); //Вставка в таблицу с типизацией uint64_t
    }
}




//-------------------------------------------------------------
//      Передать все роли пользователя и время доступа
//-------------------------------------------------------------

void find_in_table()
{
    // uint64_t ip_request = mq_receive();
    // lnh_get_first(MY_TABLE);
    // bool flag = 0;
    // for (int i = 0; i < MAX_RECORD; i++)
    // {
    //     if (lnh_core.result.key == ip_request)
    //     {
    //         flag = 1;
    //         mq_send(ip_request);
    //     }
    //     lnh_next(MY_TABLE, lnh_core.result.key);
    // }
    // if (!flag)
    //     mq_send(-1ull);
    users::key request = users::key::from_int(mq_receive());
    uint64_t qos = USERS.search(request).value().qos;
    mq_send(qos);

}
```

Результат работы:

```
iu7049@dl580:~/prac_2023/task1$ host/host_main sw-kernel/sw_kernel.rawbinary
Enter ip address:
195.19.32.12
Enter QoS level:
2
Ожидаемый интерфейс: 2
Полученный от GPC: 2
Полученный и ожидаемый совпадают
```

Рисунок 4 – Результат запуска проекта var23

Вывод: В результате работы изучены принципы работы вычислительного комплекса Тераграф и получены практические навыки решения задач обработки множеств на основе гетерогенной вычислительной структуры.