Лабораторная работа №4 (Практикум 2): Разработка и отладка программ в вычислительном комплексе Тераграф
Студент: Равашдех Ф.Х.
Группа: ИУ7-55Б

common_structs.h

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

struct students {
      using vertex_t = uint32_t;
    int struct_number;
    constexpr students(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      idx       :idx_bits;    //Поле 0:
          uint32_t      student :32;            //Поле 1*
      };

      //Запись для формирования значений
      STRUCT(val)
      {
          uint32_t      name  :32;        //Поле 0:
          time_t        age     :32;            //Поле 1*
      };
      //Обязательная типизация
      #ifdef __riscv64__
    DEFINE_DEFAULT_KEYVAL(key, val)
      #endif
};

constexpr students STUDENTS(Structures::students_pnum);

struct courses {
      using vertex_t = uint32_t;
    int struct_number;
    constexpr courses(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      idx       :idx_bits;    //Поле 0:
          uint32_t      course  :32;            //Поле 1*
      };

      //Запись для формирования значений
      STRUCT(val)
      {
          uint32_t      student_id  :32;        //Поле 0:
          time_t        title       :32;        //Поле 1*
      };
      //Обязательная типизация
      #ifdef __riscv64__
    DEFINE_DEFAULT_KEYVAL(key, val)
      #endif
};

constexpr courses COURSES(Structures::courses_pnum);
```

sw_kernel_main.cpp

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

void update_students() {

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

void update_courses() {

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


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

void select() {
        while(1){
                uint32_t qcourse = mq_receive();

                auto ccourse =
COURSES.nsm(courses::key{.idx=courses::idx_min,.course=qcourse});
                while (ccourse && ccourse.key().course == qcourse) {
                        mq_send(ccourse.value());
                        ccourse = COURSES.nsm(ccourse.key());
                }

                mq_send(-1ull);
        }
}
```

host_main.cpp

```
int main(int argc, char** argv)
{
      ofstream log("pract1.log"); //поток вывода сообщений

      std::vector<std::string> names_students;
      std::vector<std::string> names_courses;
    names_students.reserve(25);
      names_courses.reserve(25);
    for(size_t i = 0; i < 25; ++i)
    {
        names_students.emplace_back(uniqueName());
            names_courses.emplace_back(uniqueName());
    }

      unsigned long long offs=0ull;
      gpc *gpc64_inst; //указатель на класс gpc
      regex select_regex_query("select +(.*?) +from +(.*?) +where +(.*?)=(.*?)
+and +(.*?)>(.*);", //запрос
            std::regex_constants::ECMAScript | std::regex_constants::icase);

      //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(argv[2]);
      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;
      }

      // //Инициализация таблицы для вложенного запроса
      gpc64_inst->start(__event__(update_students));
      for (uint32_t user=0;user<TEST_STUD_COUNT;user++)
    {
            gpc64_inst->mq_send(students::key{.idx=user, .student=user});
        gpc64_inst->mq_send(students::val{.name=user, .age=18+user%4});
      }
      gpc64_inst->mq_send(-1ull);

      gpc64_inst->start(__event__(update_courses));
    uint32_t counter=0;
      for (uint32_t course=0;course<TEST_COURSE_COUNT;course++)
    {
        for (uint32_t st_id=course*5; st_id < course*5 + 5;st_id++)
        {
                  cout << "course_idx: " << counter << ", course_id: " << course
<< ", student_id: " << st_id << " - " << names_students[st_id] << ", title_id: "
<< course << " - " << names_courses[course] <<endl;
            gpc64_inst->mq_send(courses::key{.idx=counter,.course=course});
            gpc64_inst->mq_send(courses::val{.student_id=st_id,.title=course});
            counter++;
        }
      }
    gpc64_inst->mq_send(-1ull);
```

```
	gpc64_inst->start(__event__(select)); //обработчик запроса поиска
	while(1) {
		std::string course_index;
		smatch match_query1;
	  cout << "Введите ID курса:" << endl;
		getline(cin, course_index);
		if (!course_index.compare("exit"))
	  {
			gpc64_inst->mq_send(-1ull);
			break;
		}

	  if (stoi(course_index) >= 5)
	  {
		cout << "Ошибка, такого курса нет" << endl;
		continue;
	  }
	  cout<<"Выбранный курс: " << names_courses[stoi(course_index)]<<endl;
	  gpc64_inst->mq_send(stoi(course_index));


		while (1) {
			uint64_t result = gpc64_inst->mq_receive();
			if (result!=-1ull) {
				auto st_id = courses::val::from_int(result).student_id;
				cout << "ID студента: " << st_id << " - ";
				cout << "Имя студента: " << names_students[st_id] <<
endl;
			} else {
				break;
			}
		}
	}
	log << "Выход!" << endl;
	return 0;
}
```