**Лабораторная работа №3,** «Создание механизма виртуальной памяти»  
Группа: Б23-534  
ФИО: Калашников Владимир Алексеевич  
Номер в журнале: 5  
Год в Москве: 2025

**Описание лабораторной работы**

[Ссылка на GitHub](https://github.com/SD5th/Informatika_Labs_MIFI/tree/main/4Sem/LW_4sem-3)

Работа выполнена на языке Си с использованием и включает в себя два файла:   
1) *virtual\_memory.h*, содержит в себе реализацию механизма виртуальной памяти   
2) *main.c*, предоставлен по условию лабораторной, тестирует созданную библиотеку. Немного отредактировал, об этом дальше

Также проект оснащён небольшим скриптиком *BuildAndRun.sh* для сборки с помощью CMake и последующего запуска оболочки.

**«Класс» Virtual\_Memory**

typedef struct

{

  int\*   physical\_memory;

  bool\*  physical\_memory\_isfree;

  size\_t physical\_memory\_size;

  int\*\*  virtual\_pages;

  bool\*\* virtual\_pages\_isused;

  size\_t max\_workers;

  size\_t max\_virtual\_pages\_per\_worker;

} Virtual\_Memory;

Virtual\_Memory\* vm\_construct(size\_t physical\_memory\_size, size\_t

max\_workers, size\_t max\_virtual\_pages\_per\_worker);

void vm\_destruct(Virtual\_Memory\* vm);

int vm\_alloc(Virtual\_Memory\* vm, size\_t worker\_id);

void vm\_free(Virtual\_Memory\* vm, size\_t worker\_id, size\_t page);

void vm\_write(Virtual\_Memory\* vm, size\_t worker\_id, size\_t page, int value);

int vm\_read(Virtual\_Memory\* vm, size\_t worker\_id, size\_t page);

Структура, «класс» *Virtual\_Memory* хранит в себе массивы значений ячеек физической памяти и доступности ячеек, количество ячеек. Содержит таблицы виртуальных страниц и их доступности, доступ по номеру воркера + номеру страницы. Дальше, соответственно, размеры.

Её создание оправдано тем, что хранение указателя на физическую памяти в формате глобальной переменной неудобно и непрактично. В свою очередь, оборачивая данные в структуру, можно создавать разные экземпляры *Virtual\_Memory* и, возможно, создавать некоторые обособленные контейнеры.

Интерфейс включает функции выделения+освобождения, чтения+записи физической памяти. Каждая из функций теперь является «методом» и дополнительно принимает объект класса *Virtual\_Memory*. Также добавлены функции создания и удаления объекта виртуальной памяти.

**«Конструктор объекта» vm\_construct**

Virtual\_Memory\* vm\_construct(size\_t physical\_memory\_size, size\_t

max\_workers, size\_t max\_virtual\_pages\_per\_worker) {

  if (physical\_memory\_size == 0 || max\_workers == 0 || max\_virtual\_pages\_per\_worker == 0) {

    return NULL;

  }

  Virtual\_Memory\* vm = malloc(sizeof(Virtual\_Memory));

  if (vm == NULL) {

    return NULL;

  }

  vm->physical\_memory = malloc(sizeof(int) \* physical\_memory\_size);

  if (vm->physical\_memory == NULL) {

    free(vm);

    return NULL;

  }

. . .

for (size\_t i = 0; i < physical\_memory\_size; i++) {

    vm->physical\_memory\_isfree[i] = true;

  }

  for (size\_t i = 0; i < max\_workers; i++) {

    for (size\_t j = 0; j < max\_virtual\_pages\_per\_worker; j++) {

      vm->virtual\_pages\_isused[i][j] = false;

    }

  }

  vm->physical\_memory\_size = physical\_memory\_size;

  vm->max\_workers = max\_workers;

  vm->max\_virtual\_pages\_per\_worker = max\_virtual\_pages\_per\_worker;

  return vm;

}

vm\_construct очень длинная и представлена не целиком. В целом, она выделяет память под все поля структуры и каждый раз проверяет, правильно ли прошло выделение. Если после одного из выделений оказывается, что указатель равен NULL, то все предыдущие указатели освобождаются, а функция возвращает NULL. Если всё прошло успешно, то заполняются массивы «свободности» и «выделенности» ячеек, заполняются размеры, функция возвращает экземпляр Virtual\_Memory.

**vm\_alloc**

int vm\_alloc(Virtual\_Memory\* vm, size\_t worker\_id) {

  if (vm == NULL || worker\_id >= vm->max\_workers) {

    return -1;

  }

  int virtual\_page = -1;

  for (int i = 0; i < vm->max\_virtual\_pages\_per\_worker; i++) {

    if (!vm->virtual\_pages\_isused[worker\_id][i]) {

      virtual\_page = i;

      break;

    }

  }

  if (virtual\_page == -1) {

    return -1;

  }

  int physical\_page = -1;

  for (int i = 0; i < vm->physical\_memory\_size; i++) {

    if (vm->physical\_memory\_isfree[i]) {

      physical\_page = i;

      break;

    }

  }

  if (physical\_page == -1) {

    return -1;

  }

  vm->virtual\_pages[worker\_id][virtual\_page] = physical\_page;

  vm->virtual\_pages\_isused[worker\_id][virtual\_page] = true;

  vm->physical\_memory\_isfree[physical\_page] = false;

  return virtual\_page;

}

Проверяет корректность введённых данных, остались ли у воркера свободные страницы, осталось ли место в физической памяти и после этого возвращает номерок, по которому можно будет обратиться с помощью *vm\_read*.

**vm\_free, vm\_write, vm\_read**

void vm\_free(Virtual\_Memory\* vm, size\_t worker\_id, size\_t page) {

  if (vm == NULL || worker\_id >= vm->max\_workers || page >= vm->max\_virtual\_pages\_per\_worker) {

    return;

  }

  if (vm->virtual\_pages\_isused[worker\_id][page]) {

    vm->physical\_memory\_isfree[vm->virtual\_pages[worker\_id][page]] = true;

    vm->virtual\_pages\_isused[worker\_id][page] = false;

  }

}

void vm\_write(Virtual\_Memory\* vm, size\_t worker\_id, size\_t page, int value) {

  if (vm == NULL || worker\_id >= vm->max\_workers || page >= vm->max\_virtual\_pages\_per\_worker) {

    return;

  }

  if (vm->virtual\_pages\_isused[worker\_id][page]) {

    vm->physical\_memory[vm->virtual\_pages[worker\_id][page]] = value;

  }

}

int vm\_read(Virtual\_Memory\* vm, size\_t worker\_id, size\_t page) {

  if (vm == NULL || worker\_id >= vm->max\_workers || page >= vm->max\_virtual\_pages\_per\_worker) {

    return -1;

  }

  if (vm->virtual\_pages\_isused[worker\_id][page]) {

    return vm->physical\_memory[vm->virtual\_pages[worker\_id][page]];

  }

  return -1;

}

Три оставшихся «метода» для взаимодействия с данными. Проверяют корректность аргументов и делают то, что от них просят.

**«Деструктор объекта» vm\_destruct**

void vm\_destruct(Virtual\_Memory\* vm) {

  if (vm == NULL)

    return;

  for (size\_t i = 0; i < vm->max\_workers; i++) {

    free(vm->virtual\_pages\_isused[i]);

    free(vm->virtual\_pages[i]);

  }

  free(vm->virtual\_pages\_isused);

  free(vm->virtual\_pages);

  free(vm->physical\_memory\_isfree);

  free(vm->physical\_memory);

  free(vm);

}

Освобождает память.

**Изменения в main.c**

void run\_test(Virtual\_Memory\* vm) {

    perform\_writes(WRITE\_OPS, vm);

    perform\_reads(vm);

    free\_random\_pages(300, vm);

    perform\_writes(PHYSICAL\_MEMORY\_SIZE, vm);

    perform\_reads(vm);

    free\_all\_workers();

}

int main() {

    Virtual\_Memory\* vm = vm\_construct(PHYSICAL\_MEMORY\_SIZE, MAX\_WORKERS, MAX\_VIRTUAL\_PAGES\_PER\_WORKER);

    srand(time(NULL));

    run\_test(vm);

    vm\_destruct(vm);

    return 0;

}

Пример изменений, внесённых в *main.c*. В функции main создаётся экземпляр *Virtual\_Memory*, который позже передаётся во все тестовые функции. Соответственно, каждая из тестовых функций обзавелась новым аргументом *Virtual\_Memory\* vm*.