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

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*.