

Файлы и память

Николай Пономарев

14 февраля 2026 г.



Санкт-Петербургский
государственный университет

Формальные вопросы

- Занятия по субботам на второй и третьей паре в 3248
 - В расписании так и останется 3389
- Курс на HwProj: <https://hwproj.ru/courses/50069>
- Будут три контрольные, на 4-й паре
- Будут «Летучки»
- В конце курса будут доклады
- **Конец курса в начале апреля!**
 - Зачёт будет прямо посреди семестра!
- Каждую неделю будут выбираться несколько **счастливицков**, которые будут сдавать домашние работы вживую
 - Подозрительный на творчество БЯМ код увеличивает шансы быть избранным

Критерии оценивания

- Баллы:
 - За домашки (их будет мало, но объёмные)
 - За контрольные — выбирается две лучшие попытки из трёх
 - Плюс переписывания на зачёте, пересдаче, комиссии
- Итоговый балл за домашки: $\text{MAX}(0, (\frac{n}{N} - 0.6)) * 2.5 * 100$
 - Примерно 50 баллов максимум за обязательные домашние работы
- Летучки дают дополнительные баллы к домашке (максимум где-то три балла за каждую), которые не считаются в максимум
- Доклады также дают дополнительные баллы (примерно 5), но они достанутся только наиболее обречённым
- Есть дедлайны (минус балл к максимуму за каждую неделю просрочки, но не больше половины баллов)
- В качестве итогового берётся **минимум** из баллов за домашние работы и контрольные

Примерные баллы

Балл за домашку	Балл за контрольные	Оценка ECTS
48-50	18-20	A
46-47	16-17	B
44-45	14-15	C
43-44	12-13	D
40-42	10-11	E
0-39	0-9	на пересдачу

Что будет в курсе

- Алгоритмы и структуры данных
 - Деревья, деревья поиска, самобалансирующиеся деревья
 - Графы
 - Формальные языки, автоматы и лексический анализ
 - Немного больше про сортировки
- Системное программирование
 - Немного про то, что, судя по зачёту, не очень зашло на «Информатике»: файлы, память, функции ОС и как этим пользоваться в программах
 - Профиляторы, Perf

Сначала будет просто, потом тяжело, потом снова просто

Внезапная летучка

Доставайте листочки и ручки

Внезапная летучка

- 1 Напишите команду для того, чтобы получить исполняемый файл «executable» из программы в файле «program.c»
 - 2 Что должно и чего не должно быть в файле с расширением .h? Зачем они нужны?
 - 3 Напишите функцию, которая принимает массив чисел и его размер и возвращает новый массив, состоящий из квадратов чисел исходного массива
-
- Время: 10 минут
 - Писать и сдавать ручкой на листочке
 - Не забудьте подписать
 - Максимум 3 балла в плюс к домашкам

- Последовательность байтов на диске
 - Бывают «сырые» и «текстовые»
 - Самому файлу всё равно, это лишь способы интерпретации его содержимого
 - Режимы доступа: `r`, `w`, `a`, `r+`, `w+`, `a+`
 - Курсор
 - EOF
- Функции для работы с файлами:
 - `fopen`, `fclose`, `fprintf`, `fscanf`, `fseek`, `ftell`, `fgets`
- Файлы надо не забывать закрывать

Пример, как писать в файл

```
int main() {  
    FILE* out = fopen("ololo.txt", "w");  
    if (out == NULL) {  
        return -1;  
    }  
    fwrite("Ololo\n", sizeof(char), 6, out);  
    fprintf(out, "%s", "Ololo");  
    fclose(out);  
    return 0;  
}
```

- stdin/stdout — это тоже файлы

Пример, как читать из файла

```
#include <stdio.h>

int main() {
    FILE *file = fopen("test.txt", "r");
    if (file == NULL) {
        printf("file not found!");
        return 1;
    }
    char *data[100] = {0};
    int linesRead = 0;
    while (!feof(file)) {
        char *buffer = malloc(sizeof(char) * 100);
        const int readBytes = fscanf(file, "%s", buffer);
        if (readBytes < 0) {
            break;
        }
        data[linesRead] = buffer;
        ++linesRead;
    }
    fclose(file);
    ...
}
```

- Чтение строки целиком: `fscanf(file, "%[^\n]", buffer);`
- Или: `fgets(buffer, sizeof(buffer), file);`
- Working directory
 - Обычно Ваша текущая директория в терминале
- Работать с файлом как с массивом элементов некоторого типа может быть удобно
 - Можно отобразить файл в память
 - В Linux читать `map 2 mmap` (или `info mmap`)

Модули

- Способ группировки кода в логически обособленные группы
- В С это реализуется с помощью заголовочных файлов и файлов с реализацией
 - .h и .c
- В отдельный модуль выносятся объявления типов данных и функции, которые делают одно дело
 - Например, разные функции сортировки
 - Или всё для работы с матрицами
- В интерфейсную часть модуля выносится только то, что может использовать другой код
 - Меньше знаешь — крепче спишь
- Функции, используемые только для реализации, пишутся только в .c-файле
 - Например, функция разделения массива для быстрой сортировки или swap

Модули

Заголовочный файл:

```
#pragma once
```

```
// Комментарий к функции 1  
int function1(int x, int y);
```

```
// Комментарий к функции 2  
void function2();
```

.c-файл:

```
#include <имя заголовочного файла.h>
```

```
#include <все остальные библиотеки>
```

```
int function1(int x, int y)  
{  
    ...  
}
```

```
void function2()  
{  
    ...  
}
```

- Реализации функций в .h-файле писать нельзя
 - Иначе будет беда, если один .h-ник подключат в два .с-шника
 - Бывают прикольные исключения — header-only library
- Комментарии обязательны
- #pragma once обязательна
- Подключать «свой» заголовочный файл в .с обязательно
- Файлы .h/.с всегда ходят парами, кроме файла с main

Немного о терминологии

- Формально в Си нет модулей
- Есть — единицы трансляции (translation unit)
 - Обычно это .с после препроцессора, именно из них компилятор делает объектные файлы
- Единица трансляции определяет область видимости
 - По-умолчанию все функции доступны извне (extern)
 - Ключевое слово `static` у функции делает её невидимой для линковщика, его стоит использовать для внутренних функций
- Осторожно: `static` многозначное ключевое слово

- Способ обратиться к ОС, обычно для некоторого взаимодействия
- Типичные системные вызовы:
 - Выделение памяти
 - Запуск приложений
 - Работа с файлами (открытие, чтение, запись, закрытие)
 - Узнать системную информацию (время, ядро, на котором исполняемся)
- Специфичны для ОС
 - Делает приложение переносимым
 - Для Linux читать `man 2 syscalls` (или `info syscalls`)

Идея виртуальной памяти

Проблема:

- Памяти мало, процессов много
- Хотим удовлетворить всех с минимальными потерями для себя

Идея:

- Дадим каждому приложению собственное адресное пространство — виртуальную память
- Отобразить виртуальную память в физическую заставим ОС

Побочные эффекты

- Каждое приложение получает свой непрерывный «массив байтов», возможно даже большего размера, чем доступно физически
- Возможность перемещать данные из сильно ограниченной RAM, например, на SSD

Возможные определения

По Я.А. Кириленко

Виртуальная организация памяти — это процесс расширения логической памяти за пределы физической

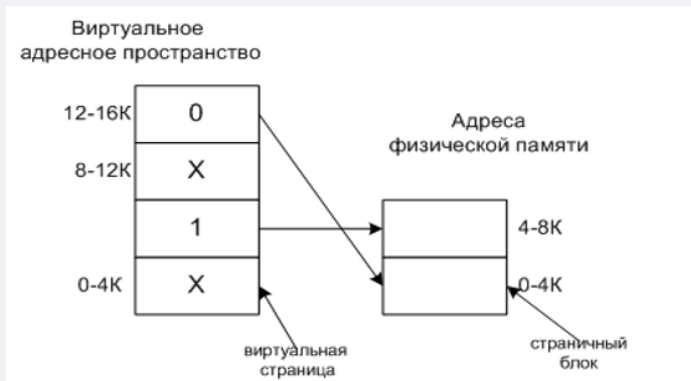
Виртуальная память — это техника, позволяющая исполнять процессы, которые могут находиться в памяти не полностью

Страничная организация (при поддержке ЦПУ)

По Я.А. Кириленко

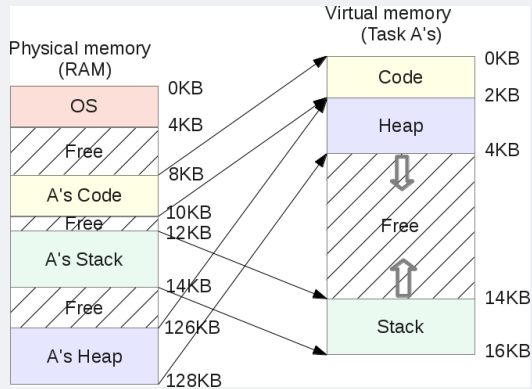
Страницы — это части, на которые разбивается пространство виртуальных адресов

Страницы всегда имеют фиксированный размер (часто 4 КиБ)



Ещё одна красивая картинка

Теперь с адресным пространством процесса. Полезно, что мы не теряем память между стеком и кучей.¹



¹Картинка из неплохой статьи: <https://web.archive.org/web/20161210164858/http://jpauli.github.io/2015/04/16/segmentation-fault.html>

Или на русском: <https://habr.com/ru/companies/nix/articles/277759/>