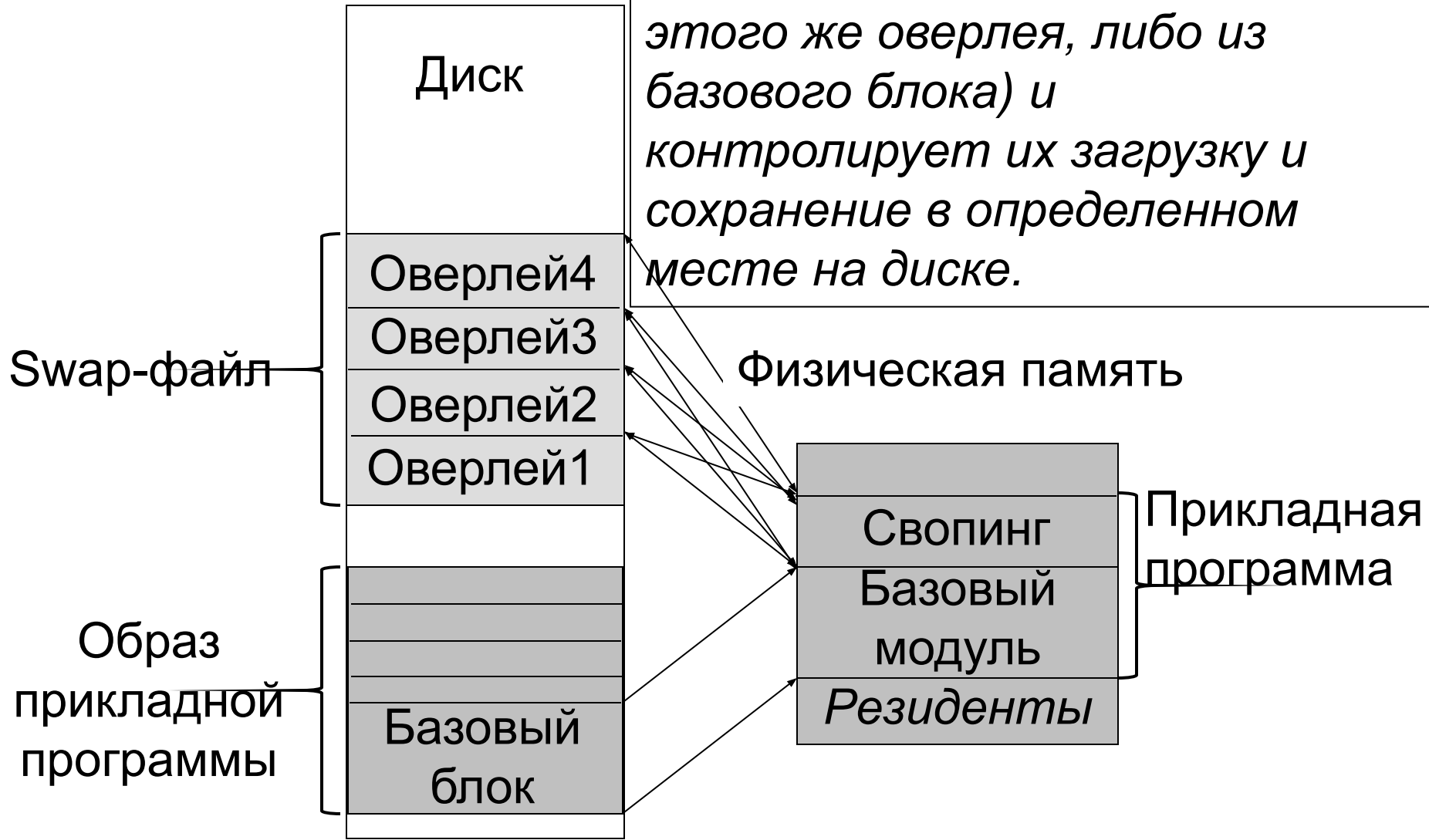


Организация памяти

Оверлейная модель



Виртуальная память.

[Виртуальное] адресное пространство – некоторая последовательность чисел. Код программы может ссылаться на адреса этого числового диапазона.

Существует некоторая схема отображения виртуальных адресов на адреса физической памяти.

Технология страничной организации памяти

Пример: машинный код позволяет адресовать 64К байт памяти, физическая память составляет 4К.

Поделим адресное пространство на 16 областей (страниц) по 4К и установим следующее соответствие:
физический адрес = виртуальный адрес % 4К;
номер области (страницы) = виртуальный адрес / 4К.

При ссылке по виртуальному адресу A

- содержимое физической памяти сохраняется на диске;
- область с номером $A/4K$ загружается в память;
- произойдет обращение по адресу физической памяти $A\%4K$.

Современные реализации страничной организации памяти

Каждому процессу выделяется адресное пространство (например в Windows числа от нуля до $0xFFFFFFFF$).

Адресное пространство разбивается на страницы размером, обычно (в зависимости от ОС) от 512 байт до 64К.

Физическая память разбивается на области (*страничные кадры* (фреймы, блоки, слоты)) размером в страницу.

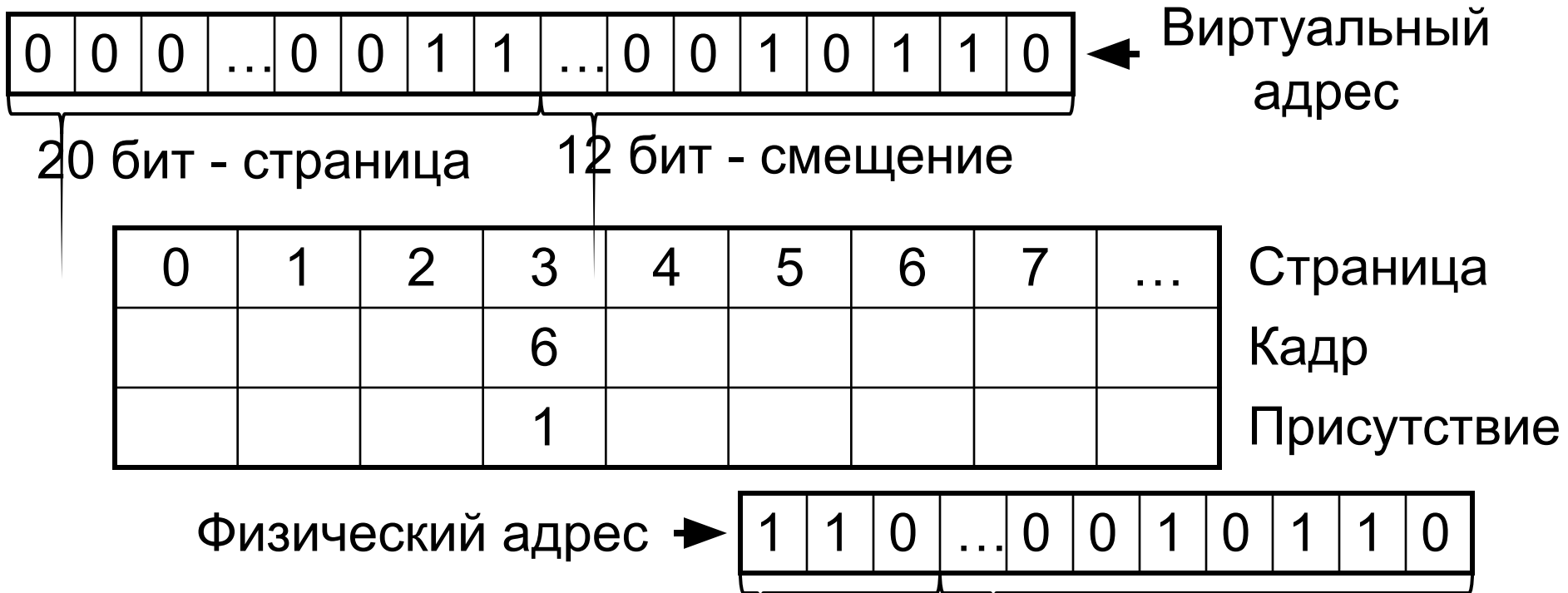
Таблица страниц устанавливает соответствие между страницами и страничными кадрами.

Виртуальная страница	Страничный кадр	Бит присутствия
...
7	0	0
6	3	1
5	4	1
4	0	0
3	2	1
2	0	0
1	0	1
0	1	1

Физическая память	Кадр
Виртуальная страница 5	4
Виртуальная страница 6	3
Виртуальная страница 3	2
Виртуальная страница 0	1
Виртуальная страница 1	0

Отображением виртуальной памяти на физические адреса занимается диспетчер виртуальной памяти – *VMM (Virtual Memory Management)*.

Аппаратной реализацией *VMM* является *MMU (Memory Management Unit)*, расположенный на чипе процессора.



Вызов страниц по требованию. При обращении к адресу страницы, которой нет в основной памяти (бит присутствия 0), генерируется исключение – **ошибка отсутствия страницы** (промах). Обработка этого исключения – считывается нужная страница с диска, в таблице страниц делается соответствующая запись и команда повторяется.

Политика замещения страниц. Существует множество алгоритмов удаления (как правило, с последующим сохранением на диске) страниц из физической памяти. Например: *LRU (Least Recently Used)* – удаляется дольше всего не использовавшаяся страница; *FIFO (First –in First out)* – алгоритм очереди.

Преимущества сегментированной памяти: упрощение перекомпиляции кода; индивидуальная защита сегментов («только для чтения», «выполнение» и т.д.).

Упражнение 1: в некоторой странично-сегментированной памяти виртуальный адрес содержит 2-разрядный номер сегмента, 2-разрядный номер страницы и 11-разрядное смещение внутри страницы. Память содержит 32К, разделенные на кадры по 2К. Каждый сегмент разрешается либо только читать, либо читать и выполнять, либо читать и записывать, либо читать, записывать и выполнять (таблица 1). Вычислите физический адрес для каждого случая доступа к памяти, перечисленных в таблице 2. Укажите в каких случаях происходит ошибка.

Табл.1

Сегмент 0
(Только чтение)

Сегмент 1
(Чтение и
выполнение)

Сегмент 2
(Только чтение)

Сегмент 3
(Только чтение)

стр. кадр

0	9
1	3
2	-
3	12

стр. кадр

0	-
1	0
2	15
3	8

Таблица
страниц
отсутствует
в памяти

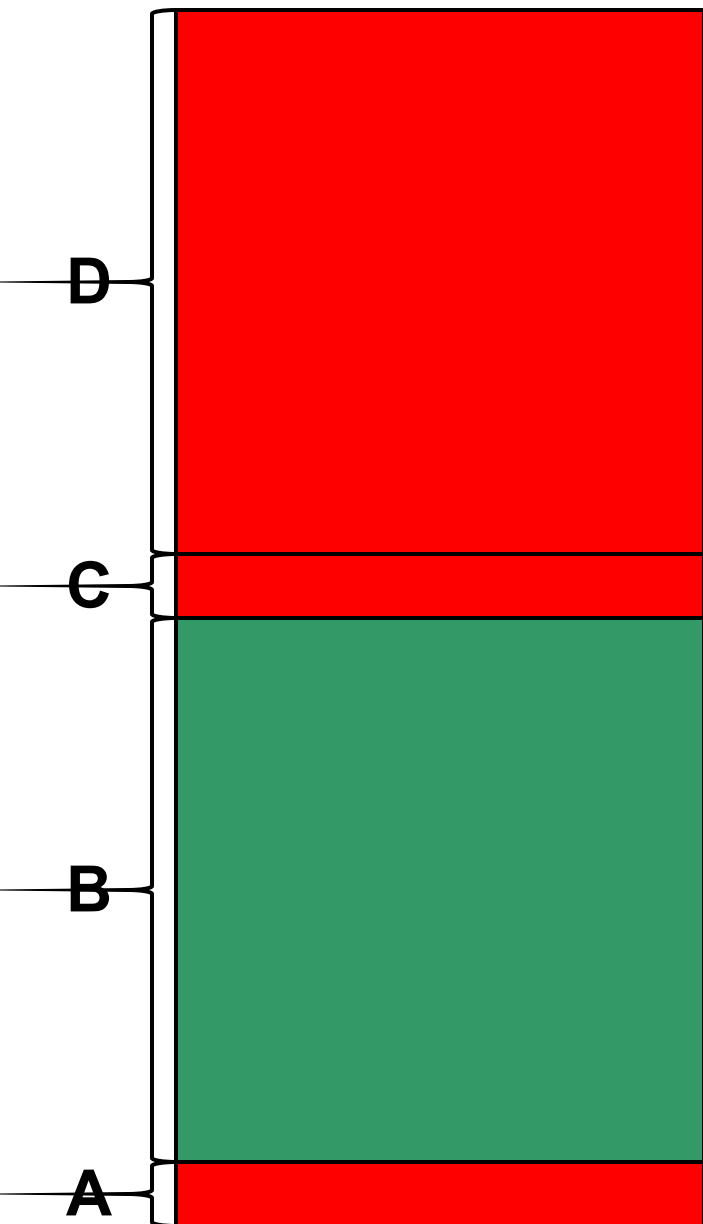
стр. кадр

0	14
1	1
2	6
3	-

Табл.2

Доступ	Сегмент	Страница	Смещение
1. Загрузка	0	1	1
2. Загрузка	1	1	10
3. Загрузка	3	3	2047
4. Сохранение	0	1	4
5. Сохранение	1	1	2
6. Сохранение	0	0	14
7. Переход	3	3	100
8. Загрузка	2	2	50
9. Загрузка	0	0	5
10. Переход	0	0	60

Особенности реализации управления памятью в MS Windows. Использование адресного пространства.



- A. 0x00000000 – 0x0000FFFF;
используется для
неинициализированных указателей;
недоступно в пользовательском
режиме.
- B. 0x0010000 – 0x7FFEFFFF; адресное
пространство процессов, содержит
прикладные модули .exe и .dll, win32
(kernel32.dll, user32.dll и т.д.), файлы,
отображаемые в память; **доступно** в
пользовательском режиме.
- C. 0x7FFF0000 – 0x7FFFFFFF;
используется для некорректно
инициализированных указателей;
недоступно в пользовательском
режиме.
- D. 0x80000000 – 0xFFFFFFFF;
зарезервировано ОС Windows для
исполнительной системы ядра и

Функции Win32 API для управления виртуальной памятью

```
#include <windows.h>
int main(){
    SYSTEM_INFO si;
    GetSystemInfo(&si);
    printf("Number of processors=
    %u\n",si.dwNumberOfProcessors);
    printf("Processor Architecture:
    %u\n",si.wProcessorArchitecture);
    printf("Page Size=%u\n",si.dwPageSize);
    printf("Low boundary of user space=
    %lx\n",si.lpMinimumApplicationAddress);
    printf("Upper boundary of user space=
    %lx\n",si.lpMaximumApplicationAddress);
    return 0;
}
```

```
typedef struct _SYSTEM_INFO {
    union {
        DWORD dwOemId;
        struct {
            WORD wProcessorArchitecture;
            WORD wReserved;
        };
    };
};
DWORD dwPageSize;
LPVOID lpMinimumApplicationAddress;
LPVOID lpMaximumApplicationAddress;
DWORD_PTR dwActiveProcessorMask;
DWORD dwNumberOfProcessors;
DWORD dwProcessorType;
DWORD dwAllocationGranularity;
WORD wProcessorLevel;
WORD wProcessorRevision;
} SYSTEM_INFO;
```

Вывод: Number of processors=4
Processor Architecture: 0
Page Size=4096
Low boundary of user space=10000
Upper boundary of user space=7ffefff
Allocation Granularity: 65536 //выравнивание

Величина константы	Значение константы
PROCESSOR_ARCHITECTURE_AMD64 9	x64 (AMD or Intel)
PROCESSOR_ARCHITECTURE_IA64 6	Intel Itanium-based
PROCESSOR_ARCHITECTURE_INTEL 0	x86
PROCESSOR_ARCHITECTURE_UNKNOWN 0xffff	Unknown architecture

Упражнение 2: используя функцию `GetSystemInfo` определите размер страницы виртуальной памяти, нижнюю и верхнюю границы адресного пространства приложения, величину выравнивания адреса выделяемой памяти и параметры процессора.

Упражнение 3: используя функцию `GlobalMemoryStatus` (см. *MSDN*) определите процент используемой памяти, количество байтов физической памяти, количество свободных байтов физической памяти, размер файла подкачки, количество свободных байтов в файле подкачки, количество байтов адресного пространства, доступного пользователю.

Упражнение 4: используя функцию `GetModuleInformation` (см. *MSDN*) и материал лекции 4 определите распределение модулей приложения в адресном пространстве процесса (текущего, дочернего и по выбору из списка активных).