# АРХИТЕКТУРА СОВРЕМЕННЫХ КОМПЬЮТЕРОВ

## Лекция 6:
## *Уровень операционной системы. Уровень ассемблера.*

ВМК МГУ им. М.В. Ломоносова, Кафедра АСВК
Доцент, к.ф.-м.н. Волканов Д.Ю.

# План лекции

- Уровень операционной системы
- Уровень ассемблера

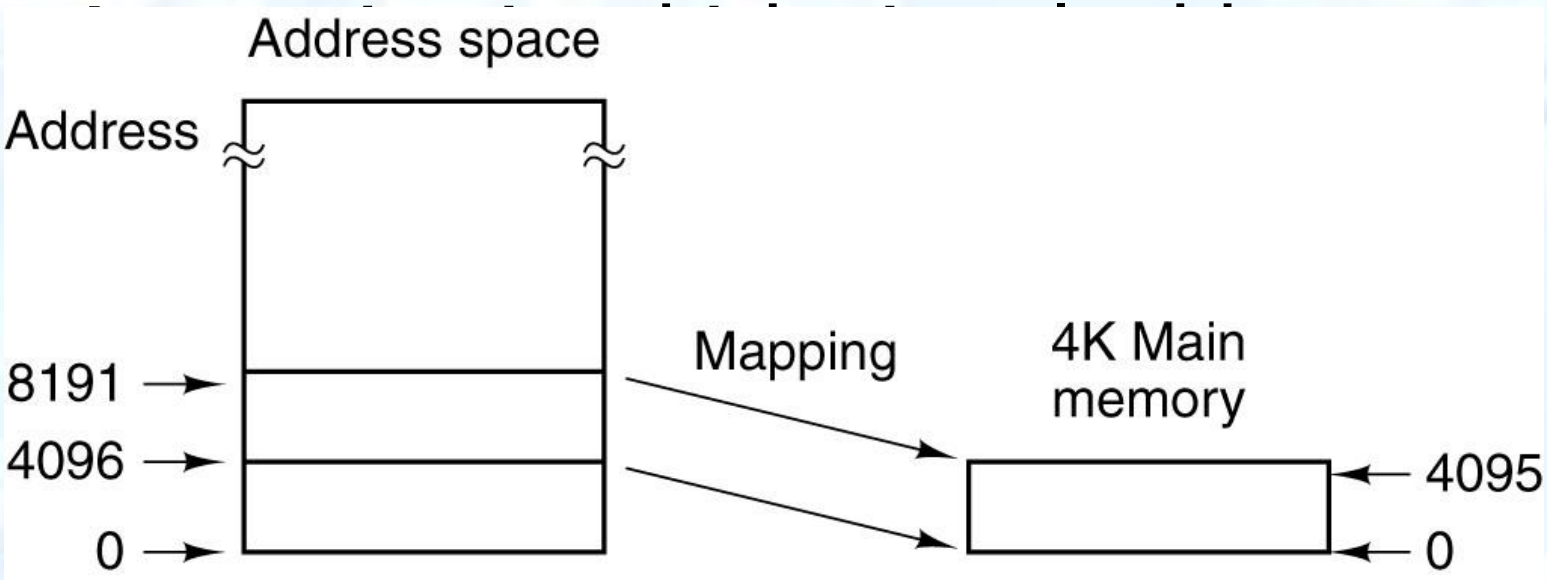
# Уровни архитектуры

- Цифровой логический уровень
- Уровень микроархитектуры
- Уровень архитектуры набора команд
- **Уровень операционной системы**
- Уровень ассемблера

# Основные особенности

- Виртуальная память
- Файловый ввод-вывод
- Параллельная работа

# Страница

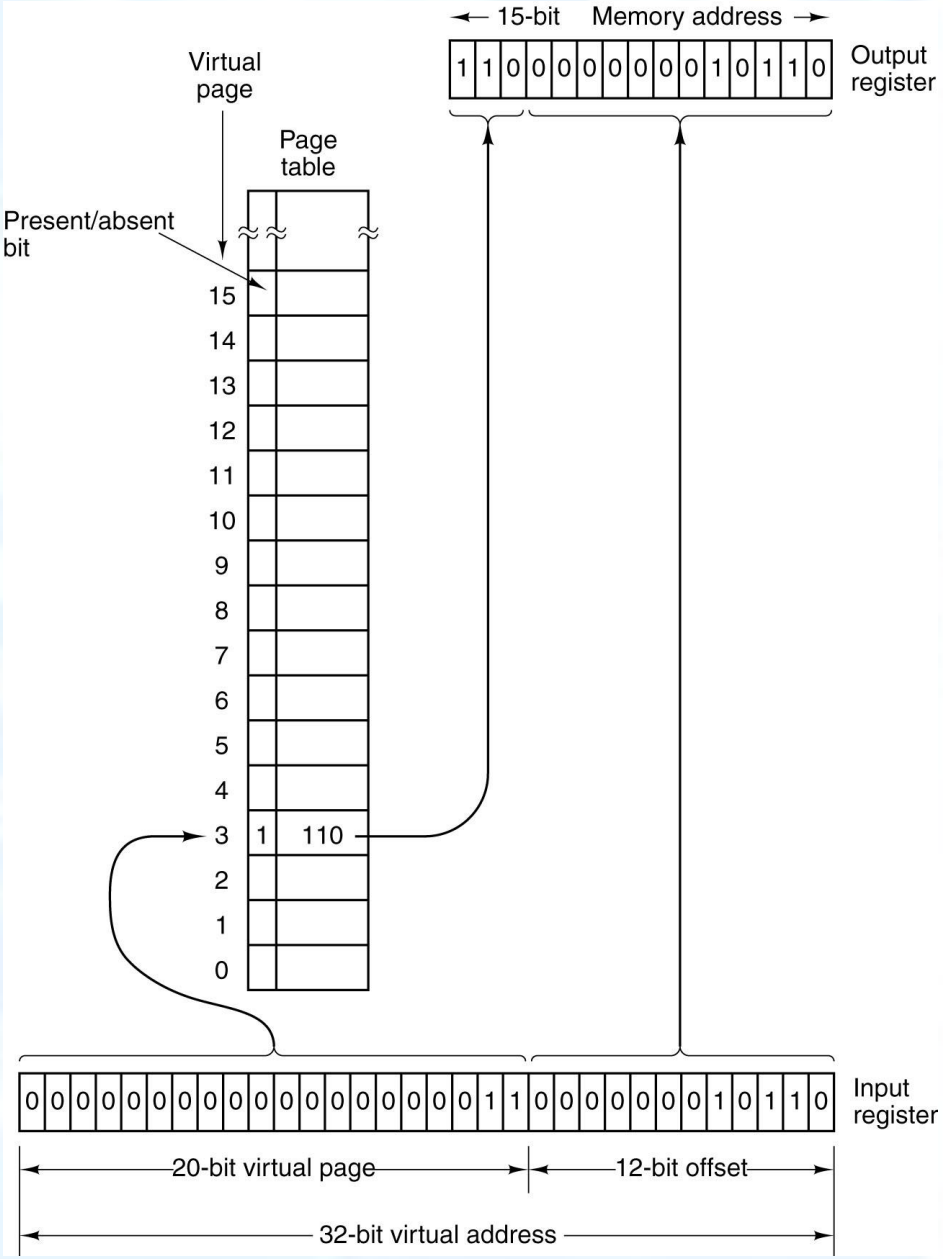

Address space
Address
8191
4096
0
Mapping
4K Main
memory
4095
0

# Реализация (1)

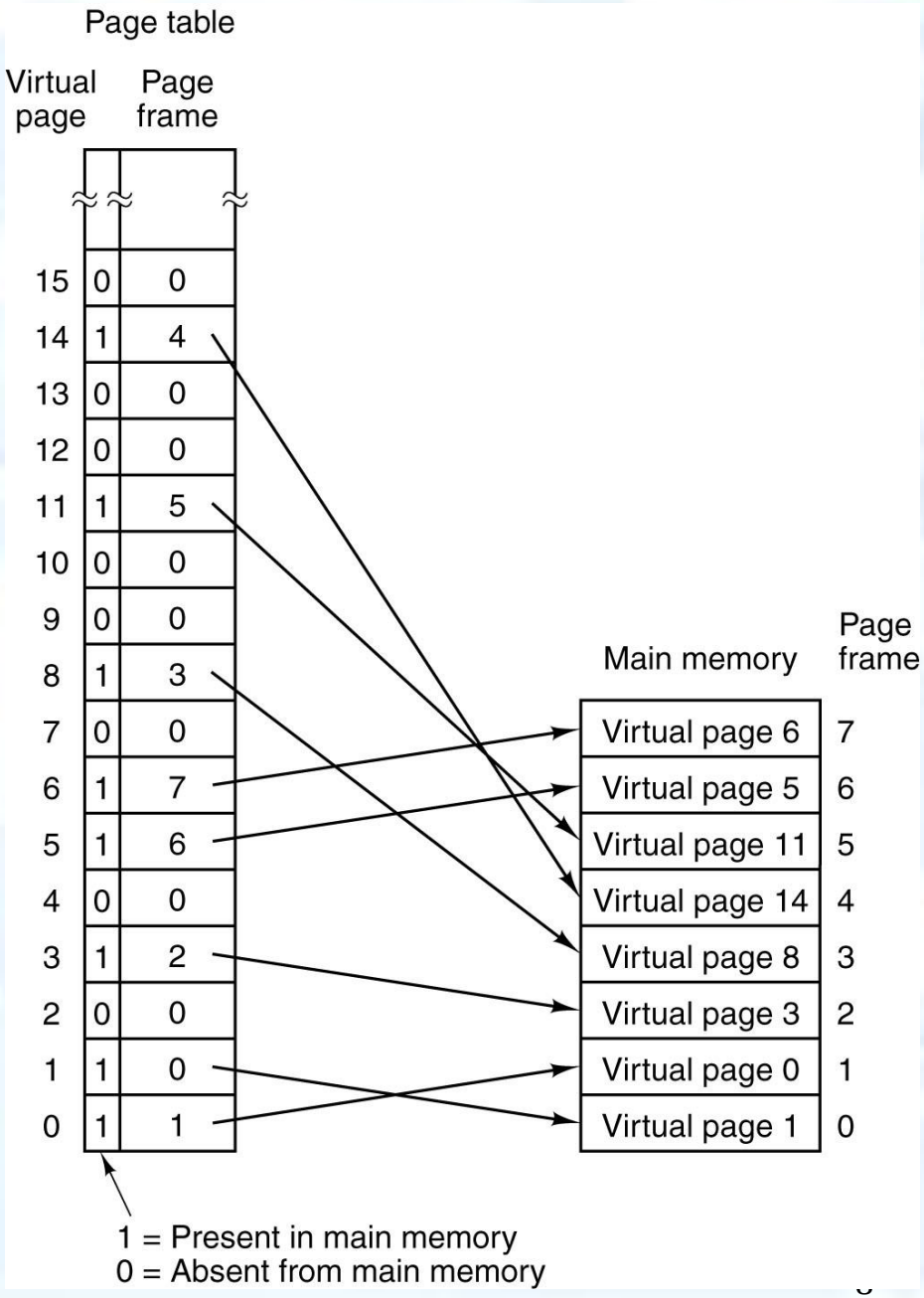
| Page | Virtual addresses |
|---|---|
| ≈ | ≈ |
| 15 | 61440 – 65535 |
| 14 | 57344 – 61439 |
| 13 | 53248 – 57343 |
| 12 | 49152 – 53247 |
| 11 | 45056 – 49151 |
| 10 | 40960 – 45055 |
| 9 | 36864 – 40959 |
| 8 | 32768 – 36863 |
| 7 | 28672 – 32767 |
| 6 | 24576 – 28671 |
| 5 | 20480 – 24575 |
| 4 | 16384 – 20479 |
| 3 | 12288 – 16383 |
| 2 | 8192 – 12287 |
| 1 | 4096 – 8191 |
| 0 | 0 – 4095 |

(a)

# Реализация (2)



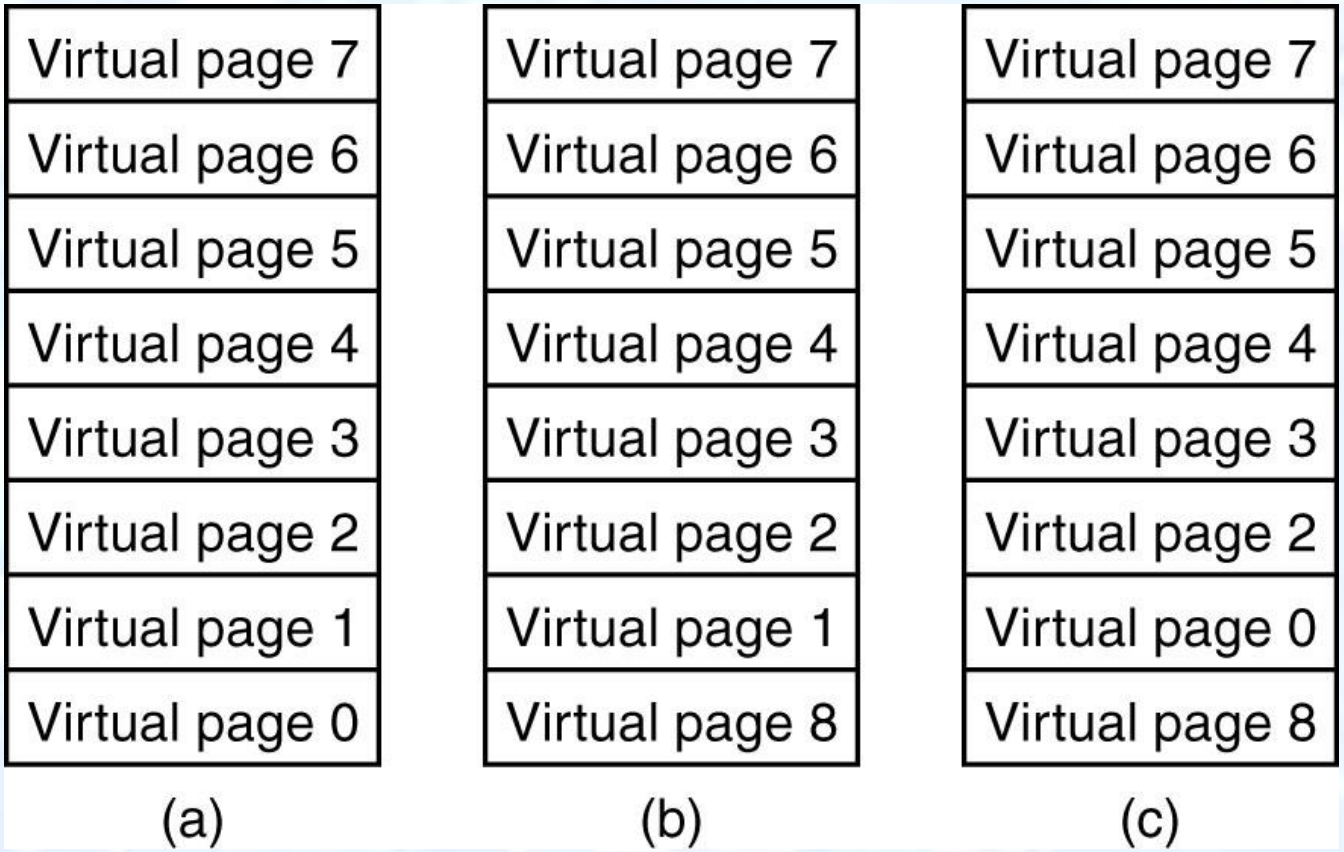
# Отображение страниц в ОП



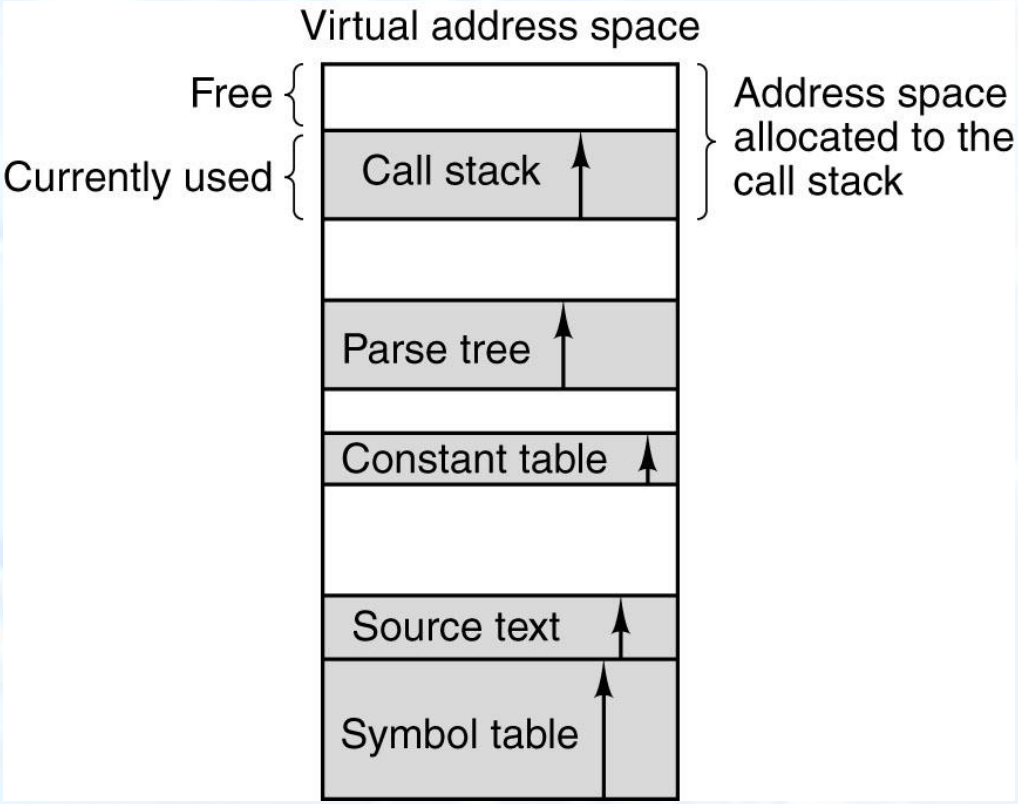
# Политика замещения страниц

- FIFO
- LRU
- Оптимальный алгоритм

# Пример пробуксовки



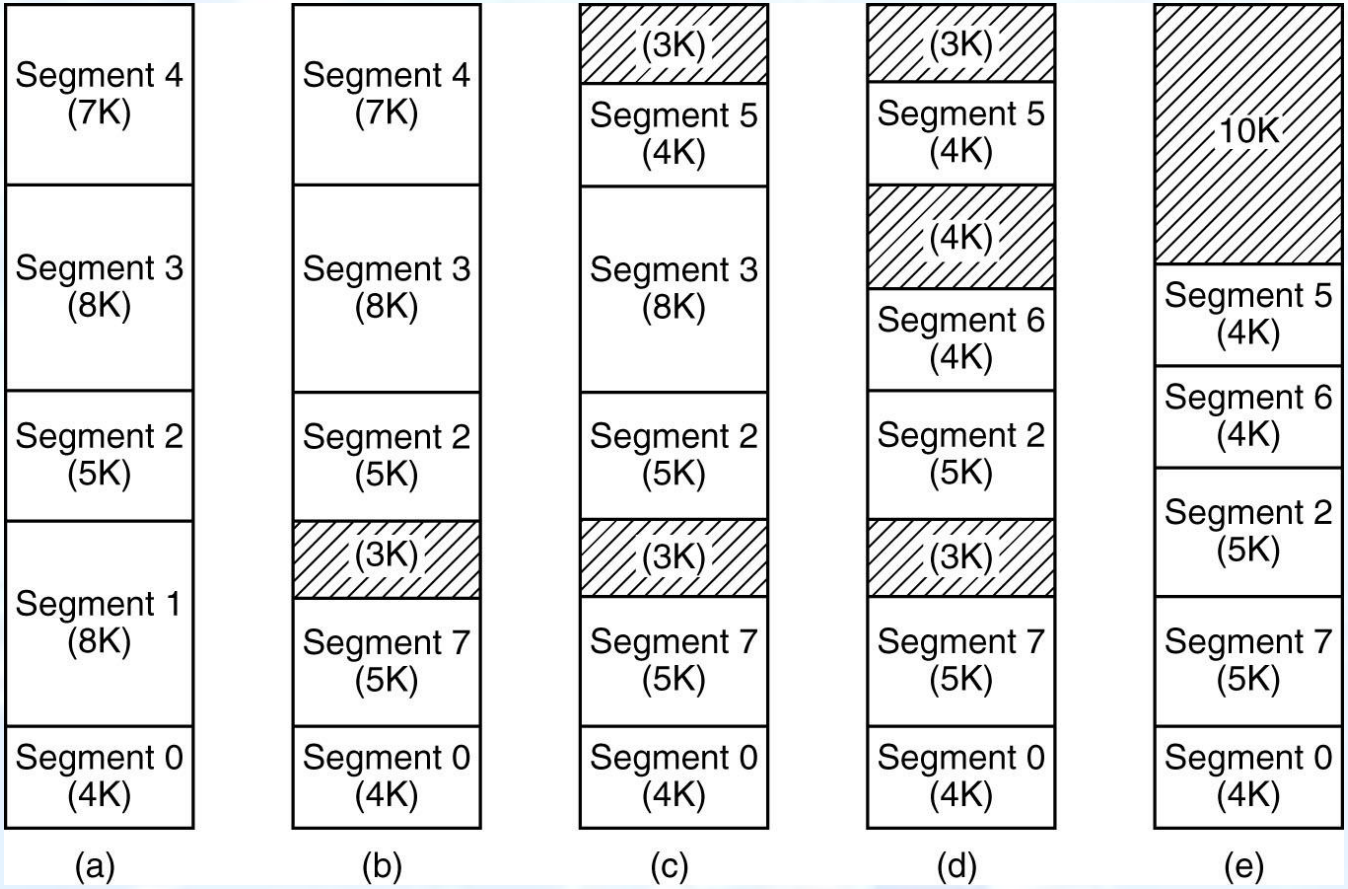
# Сегментация (1)



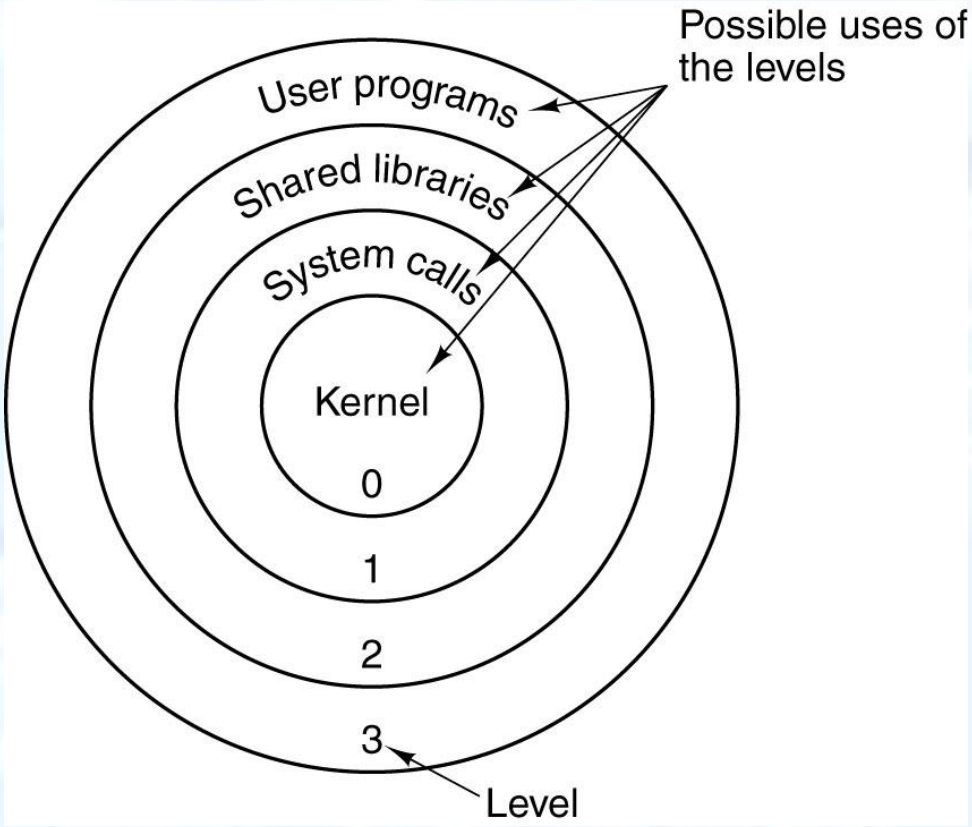
# Страницы vs. Сегменты

| Consideration | Paging | Segmentation |
|---|---|---|
| Need the programmer be aware of it? | No | Yes |
| How many linear addresses spaces are there? | 1 | Many |
| Can virtual address space exceed memory size? | Yes | Yes |
| Can variable-sized tables be handled easily? | No | Yes |
| Why was the technique invented? | To simulate large memories | To provide multiple address spaces |

# Динамика фрагментации



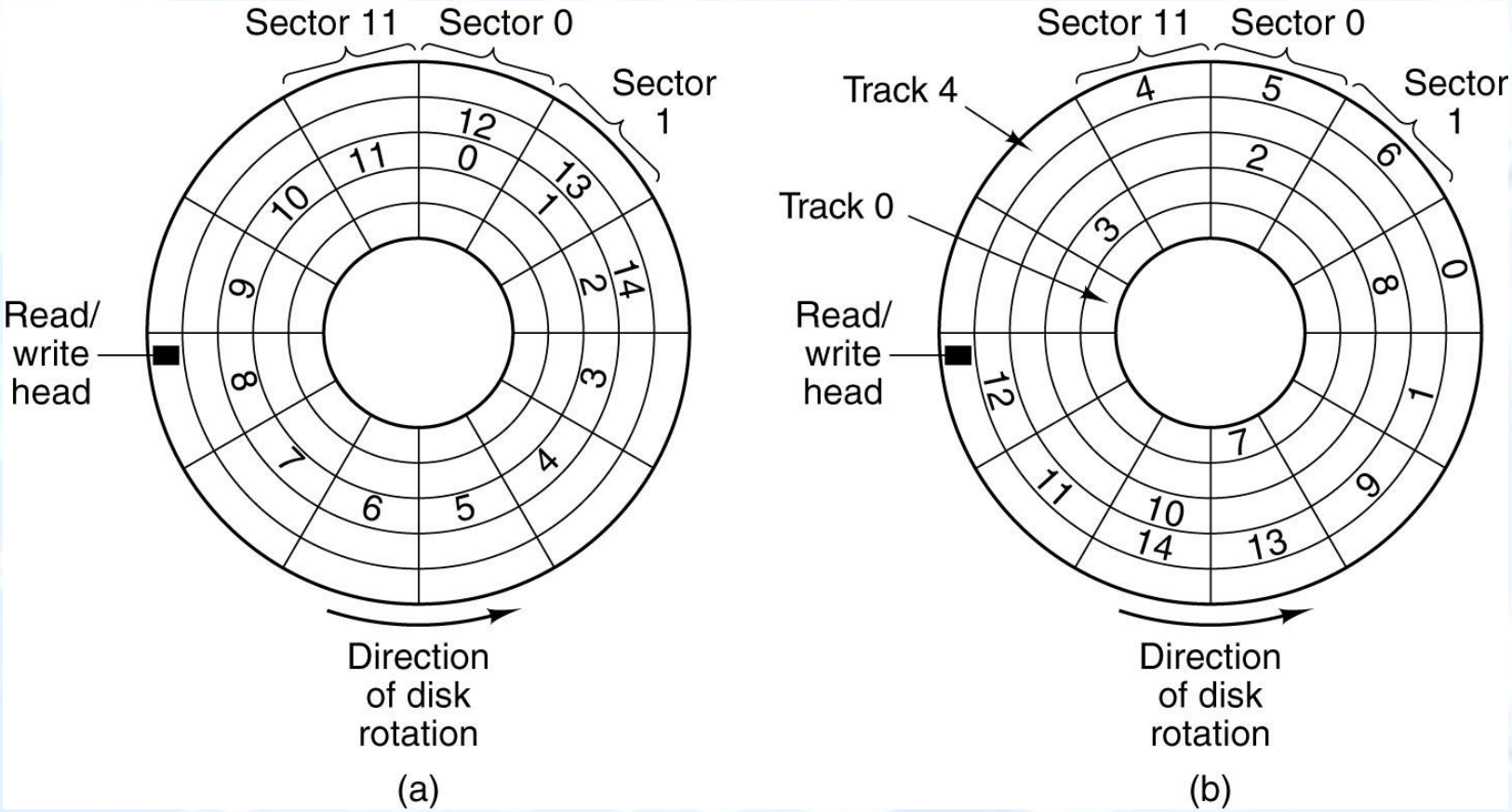
# Защита памяти в Pentium 4



# Ошибки ввода-вывода

- Аппаратные ошибки
- Процесс ввода-вывода начался до окончания предыдущего
- Ошибка синхронизации
- Несоответствие КС
- Ошибка проверки записи

# Файл на диске



# Трассировка свободных секторов

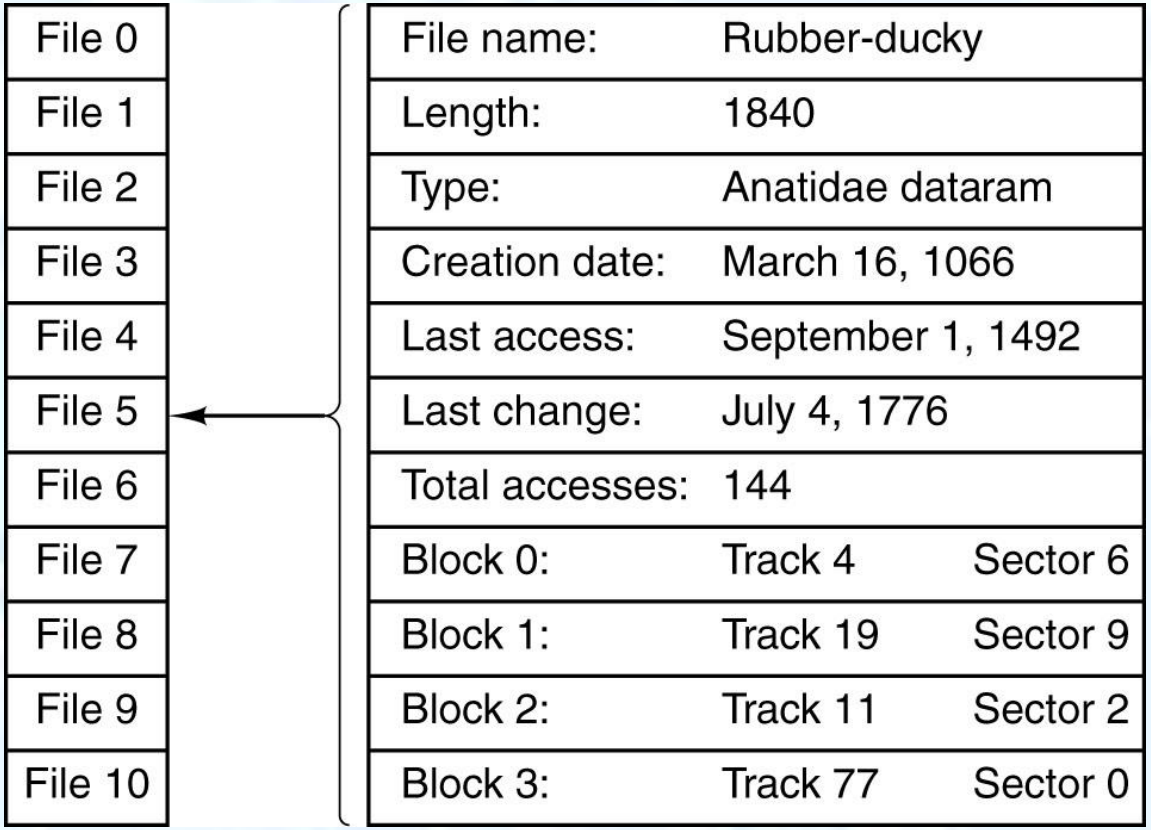
| Track | Sector | Number of sectors in hole |
|---|---|---|
| 0 | 0 | 5 |
| 0 | 6 | 6 |
| 1 | 0 | 10 |
| 1 | 11 | 1 |
| 2 | 1 | 1 |
| 2 | 3 | 3 |
| 2 | 7 | 5 |
| 3 | 0 | 3 |
| 3 | 9 | 3 |
| 4 | 3 | 8 |

(a)

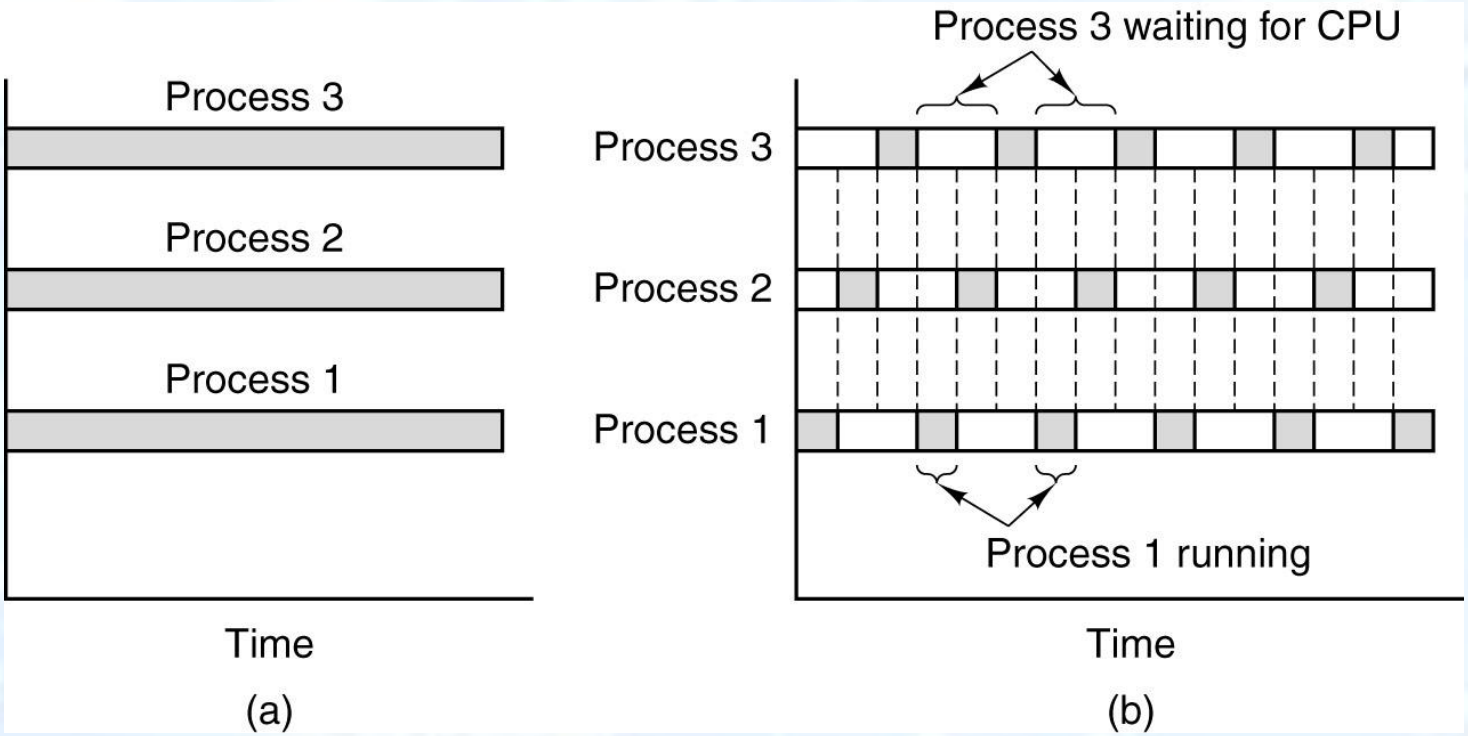
| | Sector | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Track | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 4 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

(b)

# Информация в ФС



# Параллелизм



# Состояние гонки

```
volatile int x;
// Поток 1:
while (!stop)
{
  x++;
  …
}
// Поток 2:
while (!stop)
{
  if (x%2 == 0)
    System.out.println("x=" + x);
  …
}
```

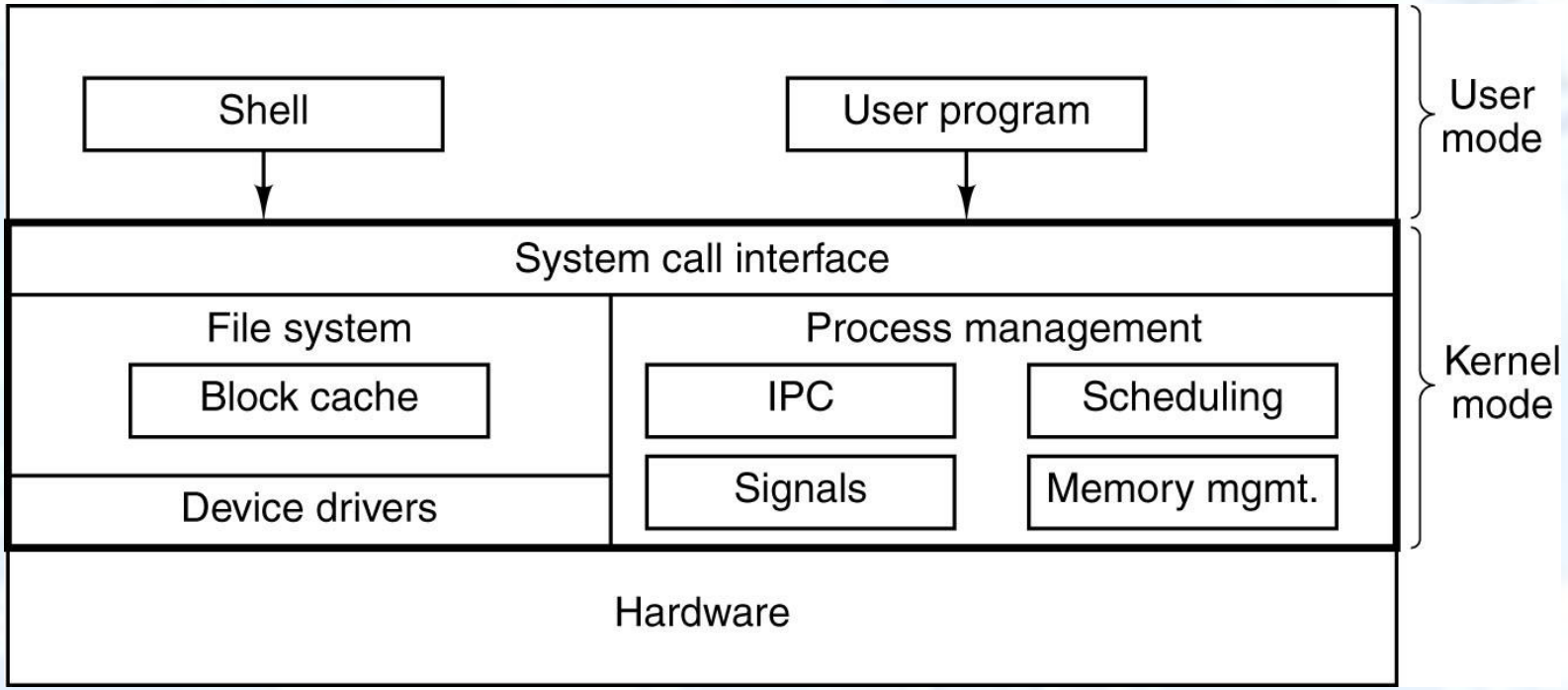
# Семафоры

| Instruction | Semaphore = 0 | Semaphore > 0 |
|---|---|---|
| Up | Semaphore = semaphore + 1;<br>if the other process was halted attempting to complete a down instruction on this semaphore, it may now complete the down and continue running | Semaphore = semaphore + 1 |
| Down | Process halts until the other process ups this semaphore | Semaphore = semaphore – 1 |

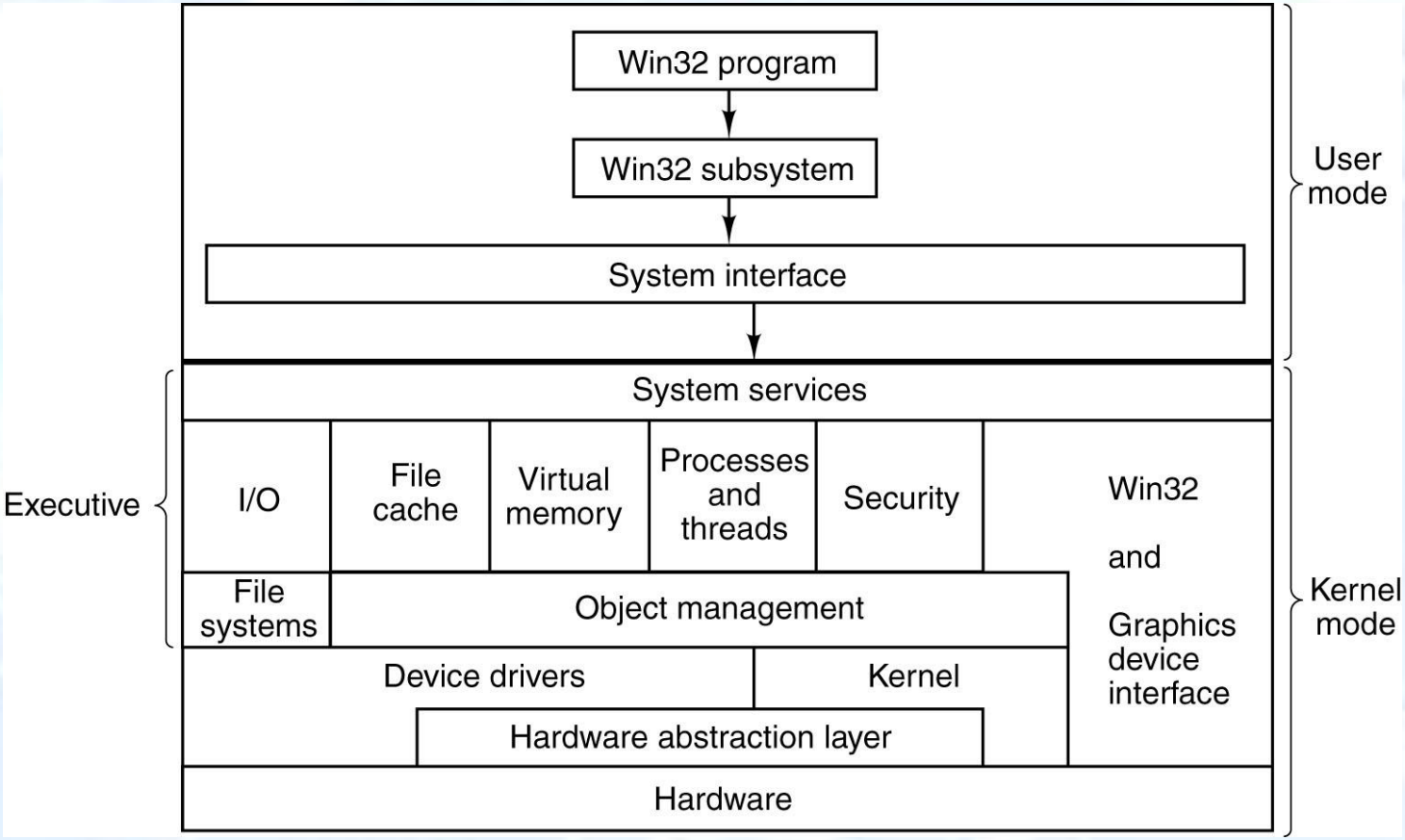
# UNIX (1)

| Category | Some examples |
|---|---|
| File management | Open, read, write, close, and lock files |
| Directory management | Create and delete directories; move files around |
| Process management | Spawn, terminate, trace, and signal processes |
| Memory management | Share memory among processes; protect pages |
| Getting/setting parameters | Get user, group, process ID; set priority |
| Dates and times | Set file access times; use interval timer; profile execution |
| Networking | Establish/accept connection; send/receive message |
| Miscellaneous | Enable accounting; manipulate disk quotas; reboot the system |

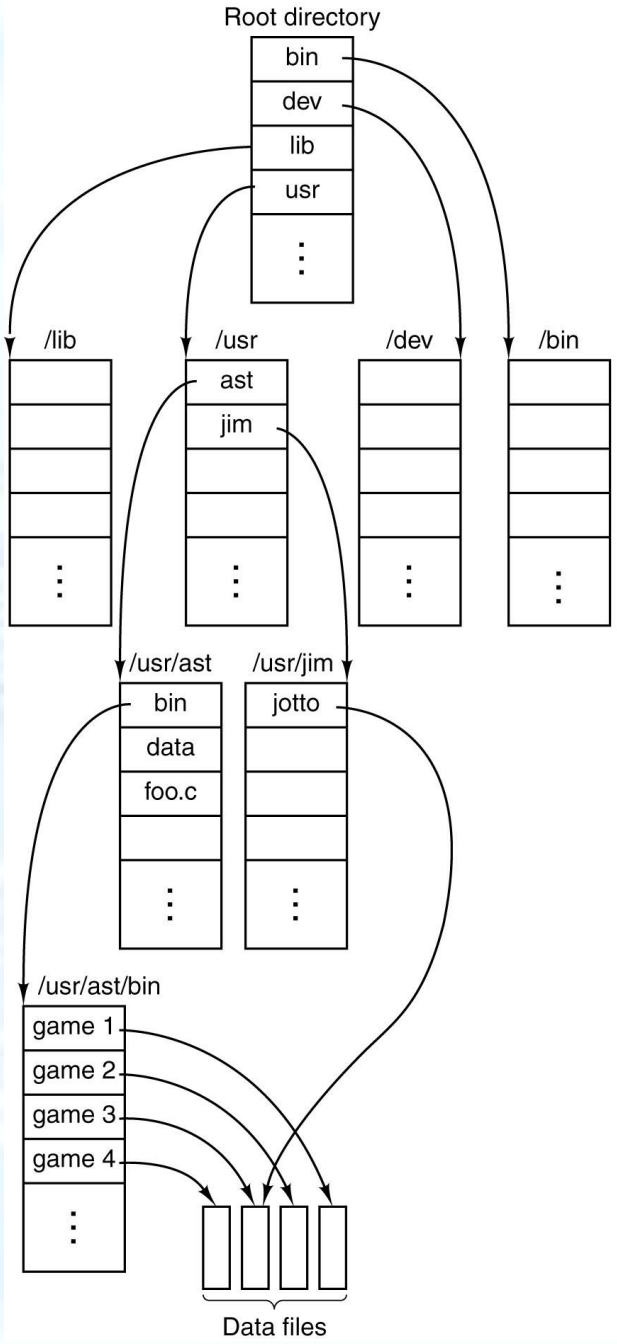
# UNIX (2)


Shell
User program
User mode
System call interface
File system
Block cache
Process management
IPC
Scheduling
Signals
Memory mgmt.
Device drivers
Kernel mode
Hardware

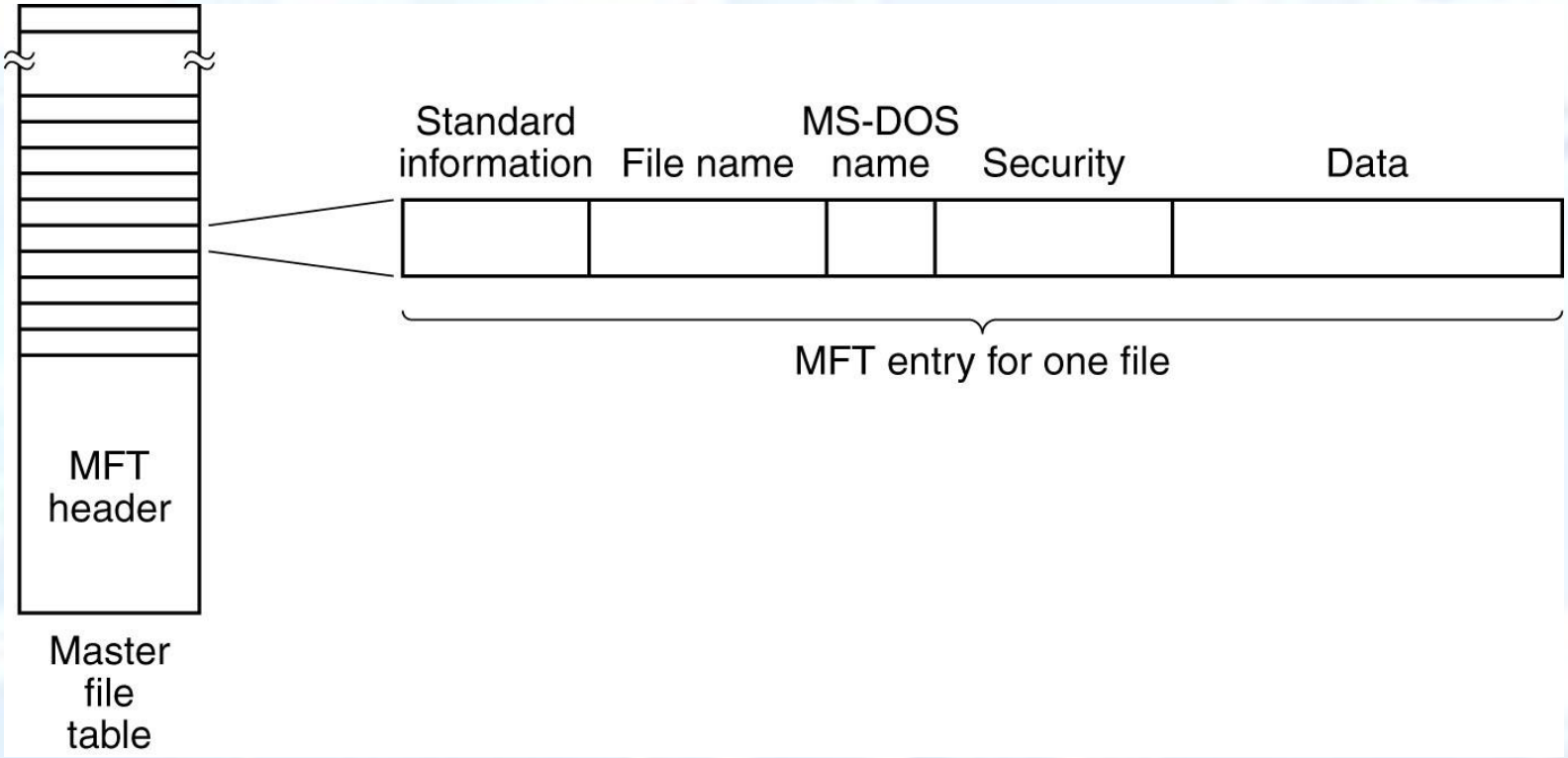
# Windows XP


Win32 program
Win32 subsystem
System interface
User mode
System services
Executive
I/O
File cache
Virtual memory
Processes and threads
Security
Win32 and Graphics device interface
File systems
Object management
Device drivers
Kernel
Hardware abstraction layer
Hardware
Kernel mode

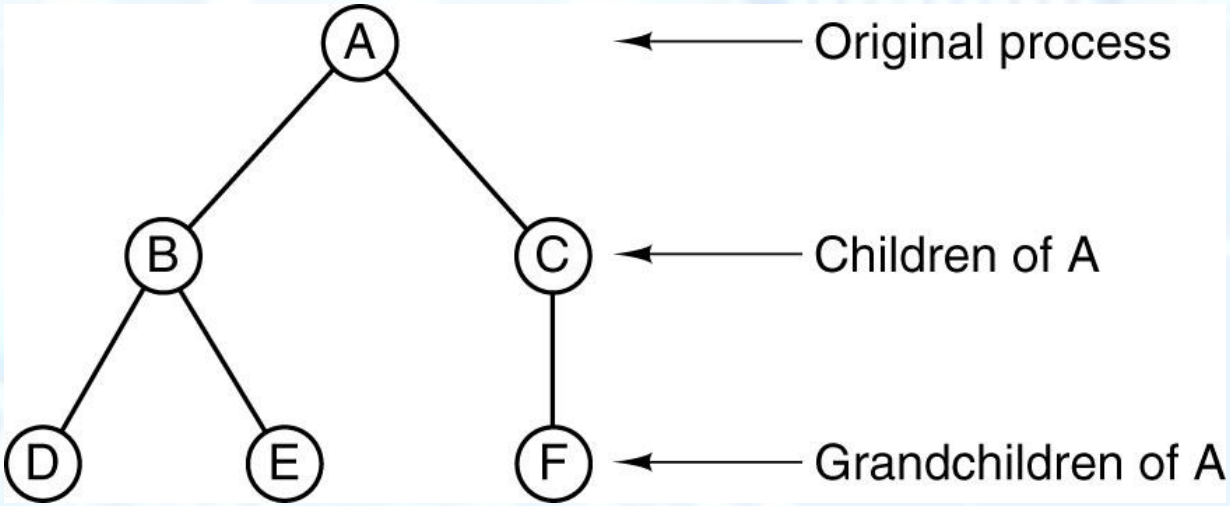
# UNIX Каталоги



# Главная ФТ в Windows XP



# Дерево процессов в UNIX



# Управление процессами в UNIX

| Thread call | Meaning |
|---|---|
| pthread_create | Create a new thread in the caller's address space |
| pthread_exit | Terminate the calling thread |
| pthread_join | Wait for a thread to terminate |
| pthread_mutex_init | Create a new mutex |
| pthread_mutex_destroy | Destroy a mutex |
| pthread_mutex_lock | Lock a mutex |
| pthread_mutex_unlock | Unlock a mutex |
| pthread_cond_init | Create a condition variable |
| pthread_cond_destroy | Destroy a condition variable |
| pthread_cond_wait | Wait on a condition variable |
| pthread_cond_signal | Release one thread waiting on a condition variable |

# Уровни архитектуры

- Цифровой логический уровень
- Уровень микроархитектуры
- Уровень архитектуры набора команд
- **Уровень операционной системы**
- **Уровень ассемблера**

# Трансляция программ

- Компиляторы:
  - создание эквивалентной программы на выходном языке;
  - выполнение полученной программы.
- Интерпретаторы:
  - выполнение полученной программы;

# Использование ассемблера

| | Programmer-years to produce the program | Program execution time in seconds |
|---|---|---|
| Assembly language | 50 | 33 |
| High-level language | 10 | 100 |
| Mixed approach before tuning | | |
| Critical 10% | 1 | 90 |
| Other 90% | 9 | 10 |
| Total | 10 | 100 |
| Mixed approach after tuning | | |
| Critical 10% | 6 | 30 |
| Other 90% | 9 | 10 |
| Total | 15 | 40 |

# Форматы команд ассемблера (1)

| Label | Opcode | Operands | Comments |
|---|---|---|---|
| FORMULA: | MOV | EAX,I | ; register EAX = I |
| | ADD | EAX,J | ; register EAX = I + J |
| | MOV | N,EAX | ; N = I + J |
| I | DD | 3 | ; reserve 4 bytes initialized to 3 |
| J | DD | 4 | ; reserve 4 bytes initialized to 4 |
| N | DD | 0 | ; reserve 4 bytes initialized to 0 |

(a)

*Вычисление N = I + J.* (a) Pentium 4.

# Форматы команд ассемблера (2)

| Label | Opcode | Operands | Comments |
|---|---|---|---|
| FORMULA | MOVE.L | I, D0 | ; register D0 = I |
| | ADD.L | J, D0 | ; register D0 = I + J |
| | MOVE.L | D0, N | ; N = I + J |
| I | DC.L | 3 | ; reserve 4 bytes initialized to 3 |
| J | DC.L | 4 | ; reserve 4 bytes initialized to 4 |
| N | DC.L | 0 | ; reserve 4 bytes initialized to 0 |

(b)

Вычисление *N* = *I* + *J*. (b) Motorola 680x0.

# Форматы команд ассемблера (3)

| Label | Opcode | Operands | Comments |
|---|---|---|---|
| FORMULA: | SETHI | %HI(I),%R1 | ! R1 = high-order bits of the address of I |
| | LD | [%R1+%LO(I)],%R1 | ! R1 = I |
| | SETHI | %HI(J),%R2 | ! R2 = high-order bits of the address of J |
| | LD | [%R2+%LO(J)],%R2 | ! R2 = J |
| | NOP | | ! wait for J to arrive from memory |
| | ADD | %R1,%R2,%R2 | ! R2 = R1 + R2 |
| | SETHI | %HI(N),%R1 | ! R1 = high-order bits of the address of N |
| | ST | %R2,[%R1+%LO(N)] | |
| I: | .WORD 3 | | ! reserve 4 bytes initialized to 3 |
| J: | .WORD 4 | | ! reserve 4 bytes initialized to 4 |
| N: | .WORD 0 | | ! reserve 4 bytes initialized to 0 |

(c)

Вычисление $N = I + J$. (c) SPARC.

# Макроопределения

```
MOV     EAX,P
MOV     EBX,Q
MOV     Q,EAX
MOV     P,EBX

MOV     EAX,P
MOV     EBX,Q
MOV     Q,EAX
MOV     P,EBX
```

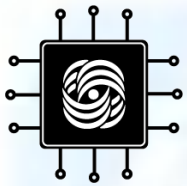
(a)

```
SWAP    MACRO
        MOV EAX,P
        MOV EBX,Q
        MOV Q,EAX
        MOV P,EBX
        ENDM

        SWAP

        SWAP
```

(b)

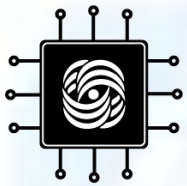
# Макросы vs. Процедуры

| Item | Macro call | Procedure call |
|---|---|---|
| When is the call made? | During assembly | During program execution |
| Is the body inserted into the object program every place the call is made? | Yes | No |
| Is a procedure call instruction inserted into the object program and later executed? | No | Yes |
| Must a return instruction be used after the call is done? | No | Yes |
| How many copies of the body appear in the object program? | One per macro call | One |



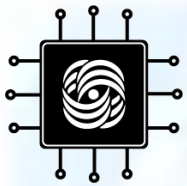
# Два прохода ассемблера (1)

| Label | Opcode | Operands | Comments | Length | ILC |
|---|---|---|---|---|---|
| MARIA: | MOV | EAX, I | EAX = I | 5 | 100 |
| | MOV | EBX, J | EBX = J | 6 | 105 |
| ROBERTA: | MOV | ECX, K | ECX = K | 6 | 111 |
| | IMUL | EAX, EAX | EAX = I * I | 2 | 117 |
| | IMUL | EBX, EBX | EBX = J * J | 3 | 119 |
| | IMUL | ECX, ECX | ECX = K * K | 3 | 122 |
| MARILYN: | ADD | EAX, EBX | EAX = I * I + J * J | 2 | 125 |
| | ADD | EAX, ECX | EAX = I * I + J * J + K * K | 2 | 127 |
| STEPHANY: | JMP | DONE | branch to DONE | 5 | 129 |



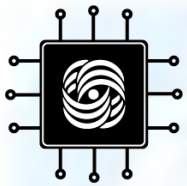
# Два прохода ассемблера (2)

| Symbol | Value | Other information |
|---|---|---|
| MARIA | 100 | |
| ROBERTA | 111 | |
| MARILYN | 125 | |
| STEPHANY | 129 | |



# Два прохода ассемблера (3)

| Opcode | First operand | Second operand | Hexadecimal opcode | Instruction length | Instruction class |
|---|---|---|---|---|---|
| AAA | — | — | 37 | 1 | 6 |
| ADD | EAX | immed32 | 05 | 5 | 4 |
| ADD | reg | reg | 01 | 2 | 19 |
| AND | EAX | immed32 | 25 | 5 | 4 |
| AND | reg | reg | 21 | 2 | 19 |

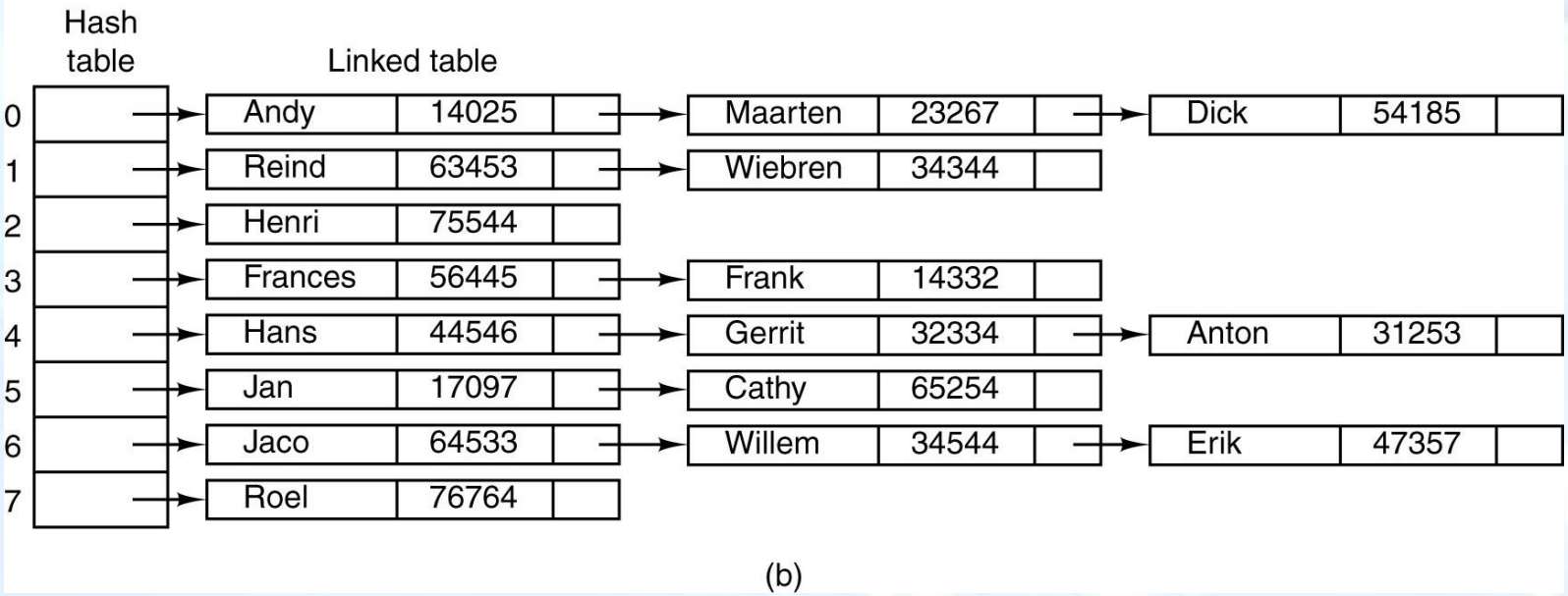


# Таблица символов (1)

| | | |
|---|---|---|
| Andy | 14025 | 0 |
| Anton | 31253 | 4 |
| Cathy | 65254 | 5 |
| Dick | 54185 | 0 |
| Erik | 47357 | 6 |
| Frances | 56445 | 3 |
| Frank | 14332 | 3 |
| Gerrit | 32334 | 4 |
| Hans | 44546 | 4 |
| Henri | 75544 | 2 |
| Jan | 17097 | 5 |
| Jaco | 64533 | 6 |
| Maarten | 23267 | 0 |
| Reind | 63453 | 1 |
| Roel | 76764 | 7 |
| Willem | 34544 | 6 |
| Wiebren | 34344 | 1 |

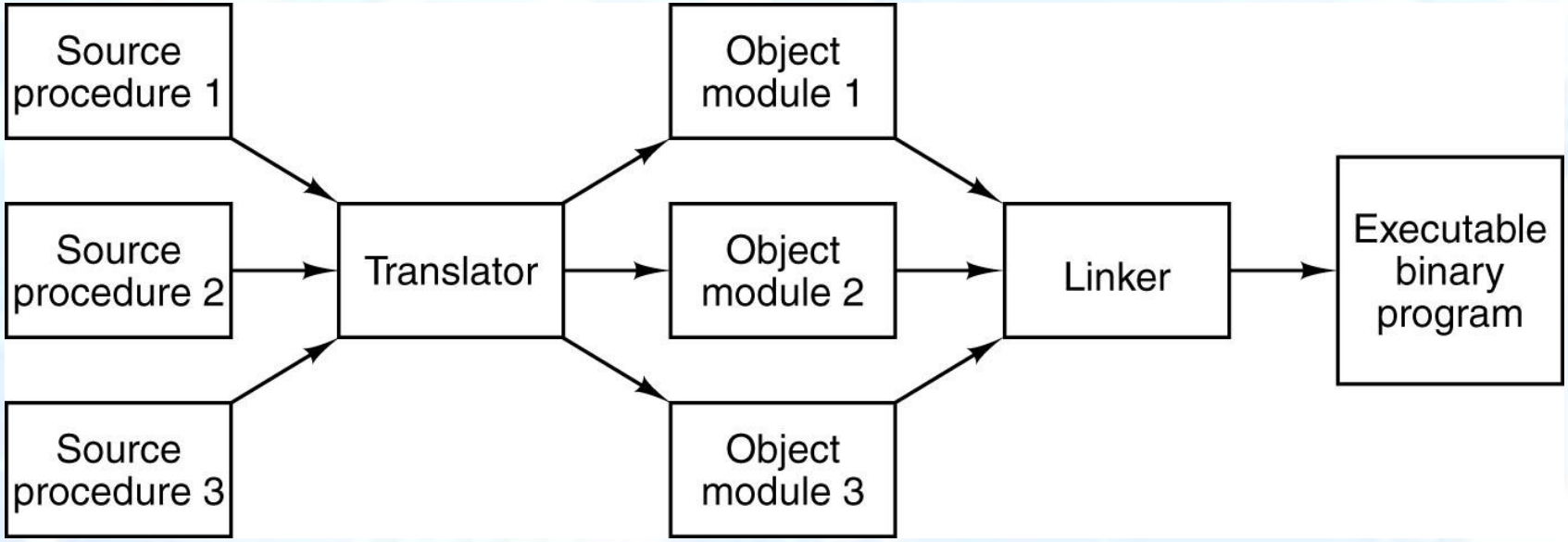
(a)

# Таблица символов (2)




(b)

# Компоновка и Загрузка



# Проблемы, решаемые компоновщиком

- Проблема перераспределения памяти
- Проблема внешней ссылки

# Действия компоновщика

- К. строит таблицу объектных модулей и их размеров.
- На её основе К. приписывает начальные адреса каждому объектному модулю.
- Ко всем командам, кот. обращаются в память прибавляется константа перераспределения.
- К командам, кот. обращаются к процедурам вставляется адрес процедур.

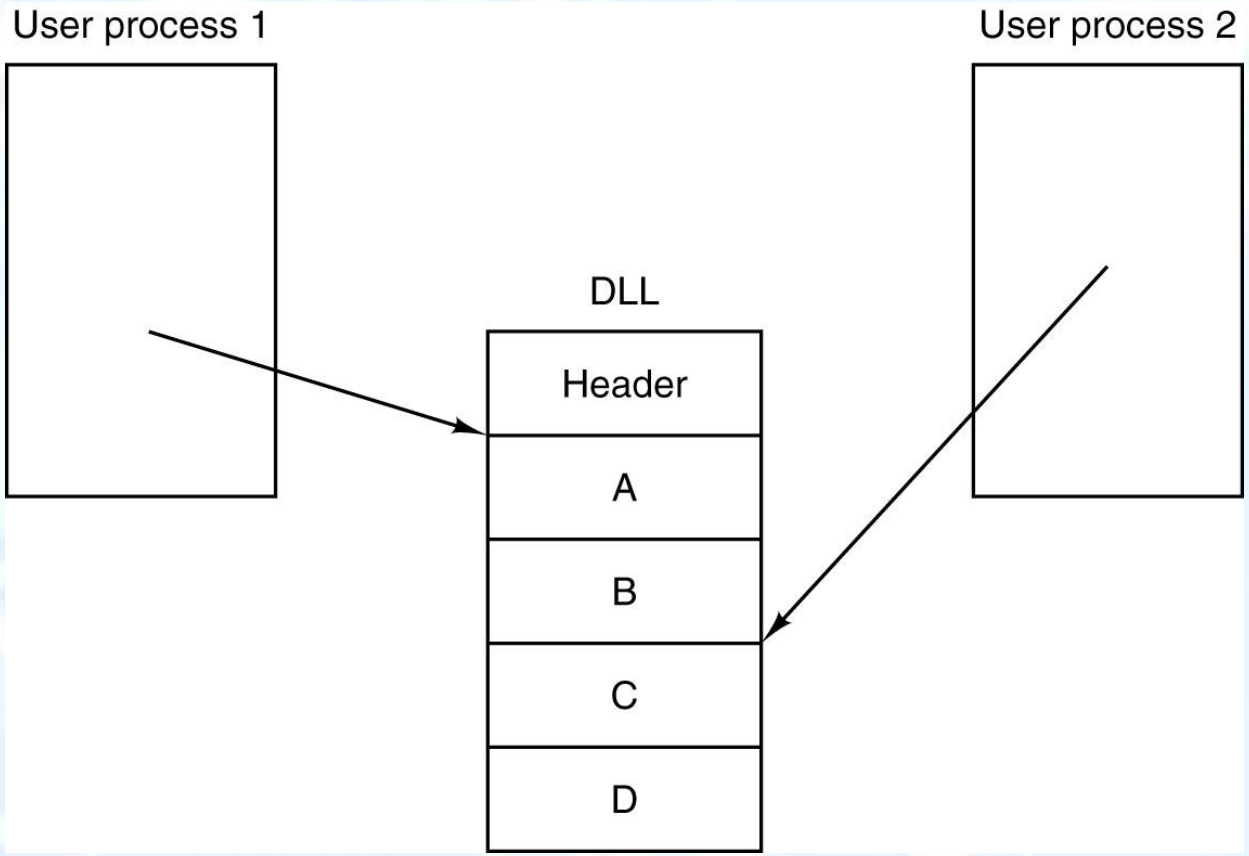
# Структура объектного модуля

- Идентификация
- Таблица точек входа
- Таблица внешних ссылок
- Машинные команды и константы
- Словарь перераспределения
- Конец модуля

# Время связывания

- Когда пишется программа
- Когда программа транслируется
- Когда программа компонуется, но до загрузки
- Когда программа загружается
- Когда загружается базовый регистр, кот. исп-ся для адресаци
- Когда выполняется команда, содержащая адрес

# Динамическая компоновка в Windows



# Спасибо за внимание!