

Голаров Андрей Викторович
Семинар 1

02.09

Зачёт по прак. с оценкой

Экзамен по курсу ОСи

Голаров "Введение в профессии", 3 тома

<http://stolyarov.info/books>

3-й том — 1 курс по ОСи

Побочный эффект (side effect) — вычисляется
ариф. выражение и происходит что-то ещё (в функции)

В ассемблере нет побочных эффектов

В паскале побоч. эффект мб в функции

$\text{fun } (a+b \rightarrow f(x))$ — $f(x)$ не только вычисляет
выражение

function $f(\underline{\text{var}} x: \text{integer}): \text{integer};$

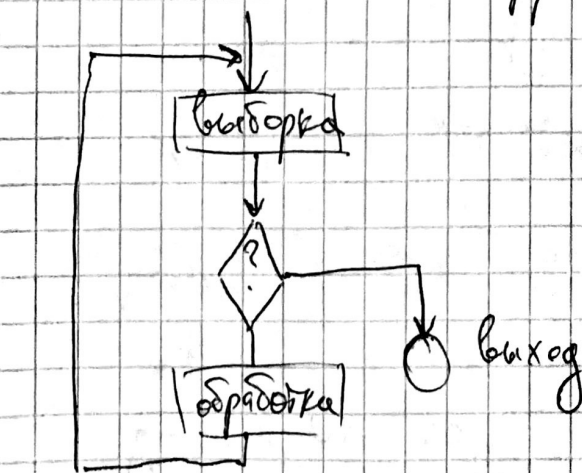
begin

$x := 15;$

— побоч. эффект.

П. эф-т — действие, связанное с вычислениями

Единственная ситуация с оправданным использо-
ванием побоч. эффекта: цикл



$\text{read}(\dots)$
while not eof do
 read
end

while get($\dots$) do

end

Побоч. эффект (лучше 2 вер., чем дублирование строк)

В Си нет процедур, только ф-и
 = - самый популярный оператор языка Си (statement)
 $t = y = x = 7;$ - оператор вычисления выражения ради подб. эффекта
 оператор - ариф. выражение.

В языке Си выполнение программы состоит из
 побочных эффектов!

Пример (Hello world)

целостн. тип `#include <stdio.h>` (вставляет вместо `#include` содержимое заголовочного файла, не Библиотека!)

`int main()` (ф-я с именем main)

`{`
 им я добавил
 катяет ф-ю
 парадигматическую
 часть; это
 новое в stdio,
 отсюда компилятор
 знает это слово
`printf("Hello, world\n");` (Си-нет главн. прогн как в пascal, только ф-я `main` - особая точка входа в программу)

`return 0;` ; превращает ариф. выражение в оператор

`}` код завершится
 процесс! всё хорошо, значение, от 0 - компилятор хочет подожать что-то не так

задает значение,
 возвращаемое
 ф-ей

GCC - компилятор, может использовать внешние ф-и, но так делать не нужно!

Ассемблер можно заменить Си почти всегда

03.09

Семинар 2

Редакторы: joe, nano, vim

! команды vim

Компилятор: gcc

Флаги: -Wall проверка

-g

позволяет смотреть, где заканчивается/упала программа

$if (a == b)$

сравнение

gcc -Wall -g

hello.c - имя исходного файла

gcc -Wall -g hello.c // компилировать

-o hello // куда положить результат компиляции

Резервные копии: ^{или} ^{some} ^{у нас} ^{написать} hello.c в другую директорию

Компилятор ничего не сказал - всё хорошо

./hello - запуск

- директория, / - сдвинуть файл?

Пример. Квадратное уравнение

double - длинное число

float - длинное, но занимает меньше памяти

sqrt - квадратный корень // хранится не в стандартной библиотеке! её нужно подключать

```
#include <stdio.h>
```

```
#include <math.h>
```

```
double det (double a, double b, double c)
```

```
{  
    return b*b - 4*a*c;  
}
```

// скобки: ; отделяет один оператор от другого
или: является частью синтаксиса оператора (если неки: ;)

```
int main()
```

```
{  
    double p, q, r, d;
```

показ. пер. можно сделать в начале блока

```
if (3 != scanf ("%lf %lf %lf", &p, &q, &r)) {
```

^{long float = double} ^{адрес пер. p}

передача параметров в p - только по значению

```
printf ("Data format error");  
return 1; // указываем ошибку, чтобы программа
```

выход: передать адрес

```
}  
if (p == 0) {  
    printf ("Not a quadratic equation (n");  
    return 2;  
}
```

p - scanf вернет: 0 - не введено ни одного числа, 1 - одно, 2 - два, 3 - три


```

d = det(p, q, r);
if (d < 0) {
    printf("No roots\n");
    return 0;
}

```

scanf: float or double

считается

printf: float = double $\Rightarrow$ %f

```

d = sqrt(d);
printf("%f %f\n", (-q-d)/(2*p), (-q+d)/(2*p));
return 0;
}

```

Запуск:

```

gcc -Wall -g ge.c //компиляция
-lm -o ge
Библиотека

```

Место хранения библиотеки: /usr/lib/libm.so

градусы

float 4 байта

double 8 $\geq$ float

long double 16 не $\leq$ double, только $\geq$

целые

char - целое число разрядности, достаточной для хранения символов
 - минимальное адресуемое, но не меньше 8 байт
 - 8 байт $\equiv$ 1 байт

Разные трактовки

0...255
 -128...127

зависит от реализации компилятора

$\Rightarrow$ 2 типа: signed char
 unsigned char

signed используется только в сочетании с char,
 не от. типа но увеличивает знаковые

short int = short 2 байта, знак

неб. константы

unsigned short (int) 2 б., 8/34
int 4

unsigned int

(unsigned) long (int) 4 / 8 байт

(unsigned) long long (int) 8 байт

char c;

c = '2';

'1' == 49

16-ричная

25 == 031 == 0x19

восьмеричная система счисления

758u - 8/34. целое

758L - long

758ul - long long

7.5

0.75e1

" " - строка - массив символов

"abracadabra" const char *

const char *s;

s = "abracadabra";

Операции

+ - * / %

остаток от деления

деление определяется типом операндов
оба операнда должны быть одного типа.
не надо разнотип

Сравнение: < > <= >= == !=

возвращает 0 или 1

(a < 3) + (b > 17) + (c != 3)

- компилятор считает, но не надо

Логические связи: $\&\&$ и $\|\$ или $!$ не

Битовые операции: $\&$ $|$ $\sim$ побитовая инверсия

$$15 \& 9 = 9$$

$$15 \&\& 9 = 1$$

Битовые сдвиги: $\ll$ $\gg$

знак. шифр $\Rightarrow$ арифметические сдвиги, б. ш. $\Rightarrow$ простые

$$1 \ll 4 = 16$$

$$0_01 \leftarrow 4$$

$$0_010000$$

Присваивание: $=$

$+=$ $-=$ $*=$ $/=$ $\&=$ $|=$ $\ll=$ $\gg=$ $\%=$

$$x += 4; \quad !=$$

$$x = x + 4;$$

$$m[f(x)] += 4;$$

$$m[f(x)] = m[f(x)] + 4;$$

m - массив

$$++ \quad +1$$

$$-- \quad -1$$

$$a = 7;$$

$$x = a++;$$

$$x = ++a;$$

$$\Rightarrow a = 8;$$

$$x = 7$$

$$x = 8$$

$$m[k++] = 7;$$

$$==$$

$$m[k] = 7;$$

$$k++;$$

Индексирование: $a[b]$

арифм. операция!

$$f(x, y, z, \dots)$$

- вызов ф-и

field

поле

$\& \quad *$ - указатели

$$a \rightarrow b \quad == \quad (*a).b$$

$$a ? b : c$$

a - истина $\Rightarrow$ вычисляется выражение b
и становится результатом операции,
 a - ложь $\Rightarrow c$

$$(a, b)$$

- вычисляется a потом b , одно из
выражений (b) - рез. операции

Семинар 3

09.09.

$\text{printf}(\text{" -- \%f -- "}, \boxed{\text{аргумент (число)}}$

лезет в параметр printf и извлекает double
(всё, что меньше $\text{int} \rightarrow \text{int}$; меньше $\text{double} \rightarrow \text{double}$
при передаче параметров в ф-ю)

$\text{scanf}(\text{" -- "}, \text{---})$

куда положить то, что прочитал?

$\text{scanf}(\text{" -- \%f -- "}, \boxed{\text{---}})$

заполняет 4 байта,

начиная с переданного адреса

$\text{float } f$

$\text{double } d$

$\&f$

$\&d$

$\text{scanf}(\text{" -- \%lf -- "}, \text{---})$

8 байт (double)

В C 16 уровней приоритета ариф. операций

, - низкий приоритет

?: - второй с конца

умнож выше слож

унарные выше бинарных

постфиксные выше префиксных

логич связи ниже сравнений

$a < b$ && $a > c$ - не нужны скобки

$a[b]$ - унар!

$\overline{a++}$

сначала

Операторы

либо пустое

1) $\langle \text{выражение} \rangle ;$ - оп-р вычисления выр. ради побоч. эф-та

2) $\{ \}$ - составной оператор
либо пустой; описание переменных, типов, набор операторов

$\{ \}$ | пустые операторы; $\{ \}$ - всё побоч. эф-ты делаются
цикл $\Rightarrow$ тело пустое

3) if

if (<условие>)

конечн условие

условие / адрес
<оператор>

[else

<оператор>]

; это, если предусмотрены операторы, независимо от того, есть ли else

Оформление:

if (_) {

{ else {

}

if (_)

{

{ else

}

4, 5, 6) while
while (<условие>)
<оператор>

do-while :

do

<оператор>

while (<условие>);
усл. продолжения цикла

(for с условием)

Оформление:

do {

<оператор>

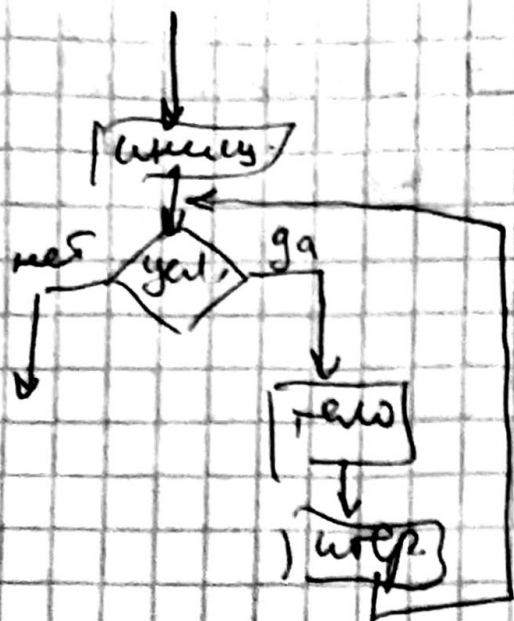
} while (<условие>);

for

for (<инициализация>; <условие>; <итерация>)
<оператор>

for (i=0; i<10; i++)
printf("Hello \n");

$\text{for}(; a < b;) \equiv \text{while}(a < b)$



$\text{for}(i=0 \text{ @ } j=10; i < j; i++ \text{ @ } j--)$

ради чего делаем эту операцию

7, 8) `break;` - досрочно завершает цикл

`continue;` - досрочно завершает выполнение тела цикла

9) `return [<варам.>];` - завершает ф-ю

```
void f()
{
  :
}
```

- в конце ф-и не нужен `return;`!

10) `goto` ровно 2 ситуации:

I. Выход из (вложенного) цикла (циклов)

```
while ( ) {
  while ( ) {
```

```
    while ( ) {
```

```
      f ( ) {
```

```
        goto escape;
```

```
      }
```

```
    }
```

```
  }
```

```
}
```

escape:

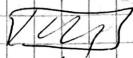
II. Ф-я открывает файлы, если нужно закрывает /
большая динамич. память — освобождает

int f()

{



if () {

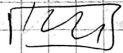


return;

goto quit;

/*переходит к quit*/

quit: }



}

1) switch () {

(несколько вариантов!)

выбирает → переход
на соответ. case →
выполняется до
конца тела

case _:

;

case _:

;

default:

* не по умолчанию *

case _:

break;

}

/* really no break here! — комментарий
(если не нужен break)

if (ptr → x)

проверка, не нулевой ли указатель, прежде чем
лезть в память

Переменные типа

целочисл. константы:

enum colors { red, green, blue, yellow, ... };
0 1 2 3
название типа!
enum colors c;

enum colors {

black = 0,

green = 0x00ff00,

red = 0xff0000,

};

Описание констант:

enum { string_length_max = 1024 },

Семинар 4

10.09

Указатель — переменная, хранящая адрес другой переменной.
а указ. на b ⇔ a содержит адрес b либо явного или

int *p;

double *q;

double *u, *v, *w;

double x;

q = &x;

// q = адрес x

*q = 35;

// x = 35

Массивы

int m[20];

sizeof(m)

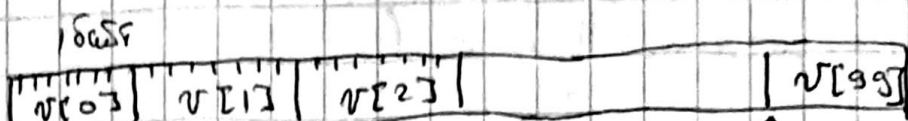
— кол-во ячеек памяти (длина)

Имя массива — адрес 1-го эл-та массива

$n[0] \equiv *n$

$\text{sizeof}(n) / \text{sizeof}(*n)$ — кол-во эл-тов

`double v[100];`



$* (v+38) \equiv v[38] \equiv 38[v]$

компилятор примет, но не кэп!

$a[b] \equiv *(a+b)$

`double *p;`

`p = v;`

`void *t;`

— абстрактный указатель: ему и его можно присваивать и указатель(ю) непомыслио, как сдвигать

Передача массива в ф-ию

`int sum(const int *a, int len)`

нельзя менять область памяти, на кот. указатель (сам указ. можно)

`int s, i;`

`s = 0;`

`for (i = 0; i < len; i++)`

`s += a[i];`

`return s;`

}

~~`int sum(const int *a, int len)`~~

~~`int s, i;`~~

~~`s = 0;`~~

~~`for (i = 0; i < len; i++)`~~

Короче:

```

(I) int sum(const int *a, int len)
{
    int s = 0;
    for (; len > 0; len--) {
        s = *a++;
    }
    return s;
}

```

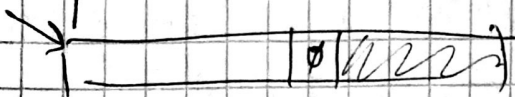
```

(II) int sum(const int *a, int len)
{
    return (len == 0) ? 0 : *a + sum(a++, len - 1);
}

```

Строки

Строка - массив char - ~~с~~

→ 

↑ ноль, не символ нуля

"abc" - корректная стр, 4 байта ('a', 'b', 'c', 0)

↑
const char * (адрес)

```

int main()
{
    char *p = "abcdef",
    p[2] = 'c' ←
    printf("%s\n", p);
    return 0;
}

```

секция кода
во время комп.
не может измениться

⇒ стр. не выполняется

```

int len(const char *s)
{
    int i = 0;
    while (s[i]) {
        i++;
    }
    return i;
}

```

"abcd"[k] - работа с

Динамическая память - выделяется во время работы progr.

`void * malloc (int size);` - выделяет память и возвр
адрес (ук. на void)

`void free (void *p);` - освобождает память

Если памяти не хватает, malloc обращается к ОЗ, она выдает
10 МБ памяти, malloc выделяет нужную часть. free
не возвращает память ОС!

```
double *p;
```

```
int n;
```

```
p = malloc (n * sizeof(double));  
                *p
```

```
free(p);
```

Структурный тип данных

/6.09

пример 5

Структур данных - то-то, что лежит в памяти.

В С (аналог записи в Паскале) - переменная, сост.
из гр. переменных разных типов (поля).

```
struct mystz {
```

```
    int count;
```

// 4 байт

```
    double average;
```

// 8 байт

```
    char name[100];
```

// 100 байт

```
};
```

Можно описать пер., но обратно не делаем:

```
struct mystz {
```

```
}; s1, s2;
```


struct mystz s1; // имя типа из 2 слов!

Выделяется память, чтобы поместиться
поле `count` и поле `average` нулевой
памяти, чтобы работать быстрее

`s1.count`

`s1.average`

`s1.name`

// адрес начала поля `name`

Присваивание структур - копирование данных
область памяти. Обязательно работать с адресом

struct mystz *p1;

`p1 = malloc(sizeof(struct mystz));` | вариант

`p1 = malloc(sizeof(*p1));`

`p1 = (struct mystz *) malloc(sizeof(st...)` | ~~неверно~~

Писать в C++ на чистом C нельзя!

`(*p1).count`

поскольку унарная операция, её можно писать `*p1`

`p1 -> count` - то же самое

Списки (с помощью структур)

struct item {

int count;

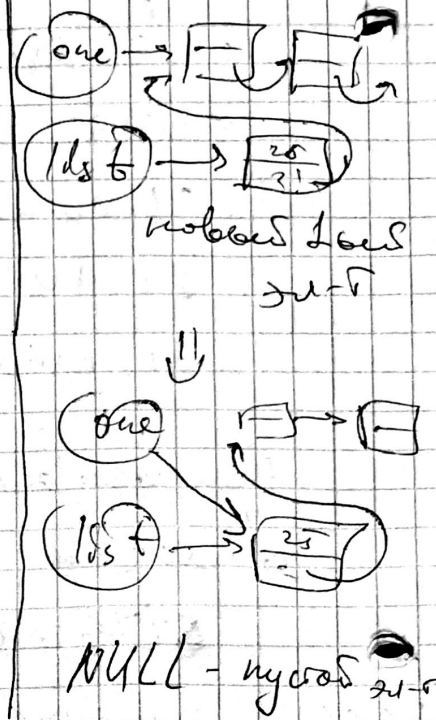
struct item *next;

};


```

struct item *vec2list (const int *v, int sz)
{
    int i;
    struct item *list, *one;
    one = NULL;
    for (i = sz-1; i >= 0; i--)
    {
        list = malloc (sizeof(*list));
        (*list).count = v[i];
        (*list).next = one;
        one = list;
    }
    return one;
}

```



↑ godabuenno & naran
 ↓ godabuenno & koners

```

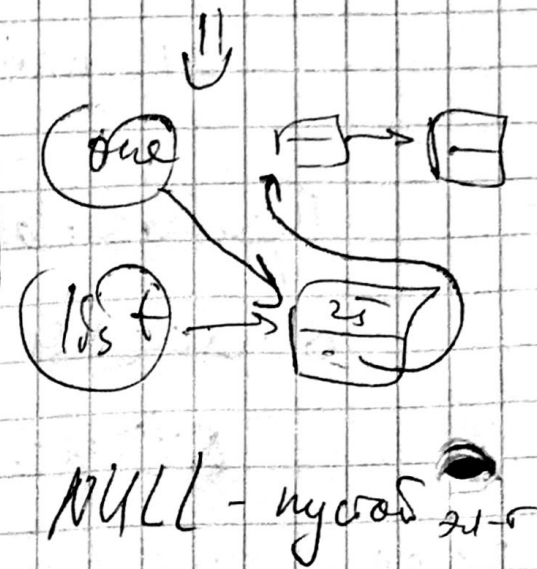
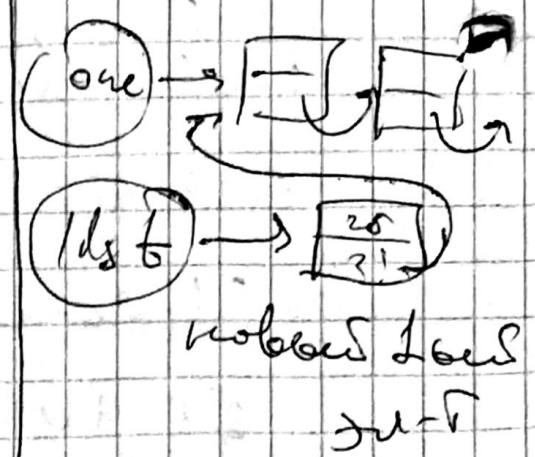
struct item *vec2list (const int *v, int sz)
{
    int i;
    struct item *tmp, *first, *last;
    first = NULL;
    last = first;
    for (i = 0; i < sz; i++)
    {
        tmp = malloc (sizeof(*tmp));
        tmp->count = v[i];
        tmp->next = NULL;
        if (first == last)
        {
            first = tmp;
            last = tmp;
        }
    }
}

```

```

struct item *vec2list (const int *v, int sz)
{
    int i;
    struct item *list, *one;
    one = NULL;
    for (i = sz-1; i >= 0; i--)
    {
        list = malloc (sizeof(*list));
        (*list).count = v[i];
        (*list).next = one;
        one = list;
    }
    return one;
}

```



↑ godaburene & naran
 ↓ godaburene & koner

```

struct item *vec2listb (const int *v, int sz)
{
    int i;
    struct item *tmp, *first, *last;
    first = NULL;
    last = first;
    for (i = 0; i < sz; i++)
    {
        tmp = malloc (sizeof(*tmp));
        tmp->count = v[i];
        tmp->next = NULL;
        if (first == last)
        {
            first = tmp;
            last = tmp;
        }
    }
}

```



```
else
{
```

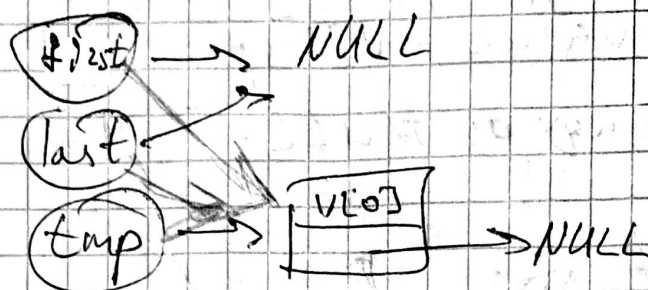
```
    last → next = tmp;
    last = tmp;
}
```

```
last = NULL;
return first;
```

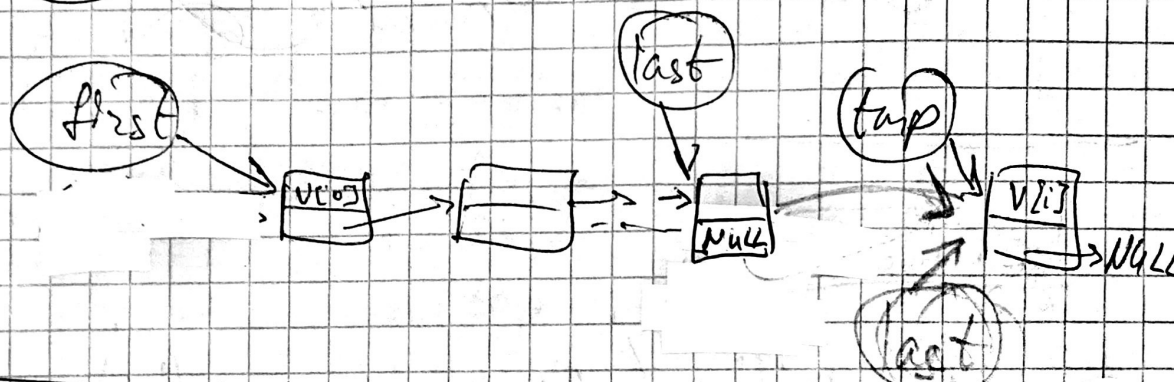
```
}
```

```
struct item *vec2list (const int v, int sz)
{
    struct item *first, *last, *tmp;
}
```

ИЗМ-Т:



Описание:



```
struct item *vec2list (const int v, int sz)
{
```

```
    struct item *list;
```

```
    if (sz == 0)
    {
        list = NULL;
    }
```

```
    else
```

```
    {
        list = malloc (sizeof (*list));
```

```
        list → count = v [ 0 ];
```

```
        list → next = vec2list (v+1, sz-1);
```

```
    }
    return list;
```

```
}
```

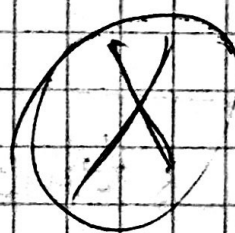
Самунар.

12.09

```
struct item *vec2list (const int *v, int sz)
{
    struct item *first, *last;
    int i;
    first = NULL;
    for (i = 0; i < sz; i++)
    {
        *((i == 0) ? &first : &(last->next)) =
            malloc (sizeof (struct item));
        last = (i == 0) ? first : last->next;
        last->value = v[i];
        last->next = NULL;
    }
    return first;
}
```

Удаление списка

```
void dellist (struct item *list)
{
    struct item *tmp;
    if (list->next == NULL)
    {
        free (*list);
    }
    else
    {
        tmp = list->next;
        free (*list);
        list = tmp;
        dellist (*list);
    }
}
```



struct item *tmp, *first

while (first != NULL) {

tmp = (*first).next;

free (first);

first = tmp;

}

tmp = first;

first = first->next;

free (tmp);

Queue operations in C

struct item *first, *tmp1, *tmp2;

tmp1 = first;

tmp2 = first;

while (tmp1 != NULL)

{ if (tmp1->count < 0)

{ if (tmp1 == first)

{ first = first->next;

tmp2 = first;

free (tmp1);

tmp1 = first;

continue;

} else

tmp2->next = tmp1->next;

free (tmp1);

tmp1 = tmp2->next;

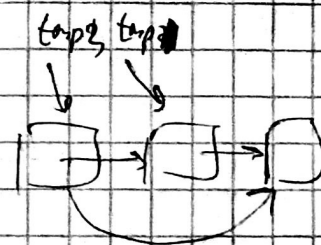
continue;

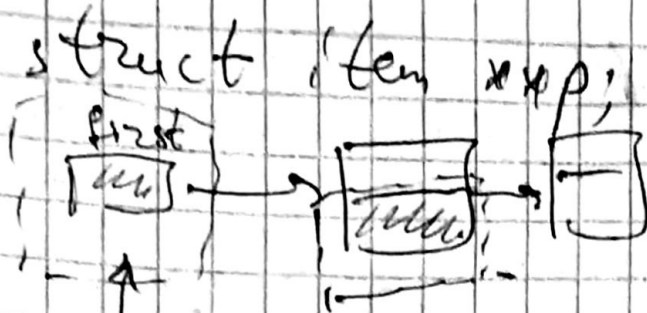
}

tmp2 = tmp1;

tmp1 = tmp1->next;

}





указатель на указатель

while (first && first->count < 0)

{ struct item *p;

p = first;

first = first->next;

free(p);

}

if (first)

{

struct item *q, *p;

p = first;

while (p->next)

{

if (p->next->count < 0)

{

q = p->next;

p->next = q->next;

free(q);

}

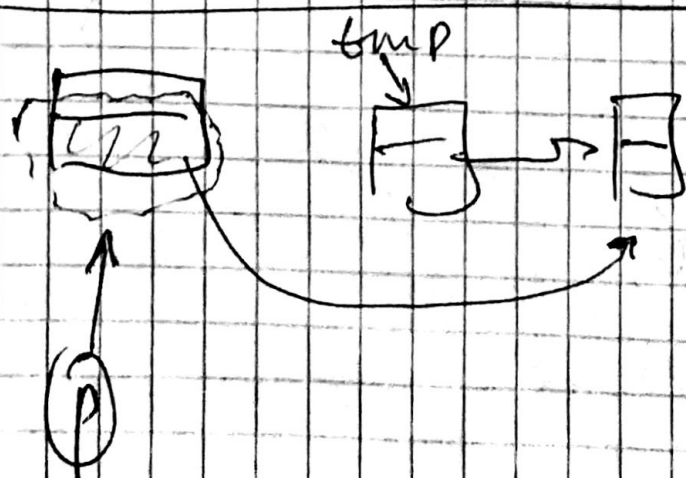
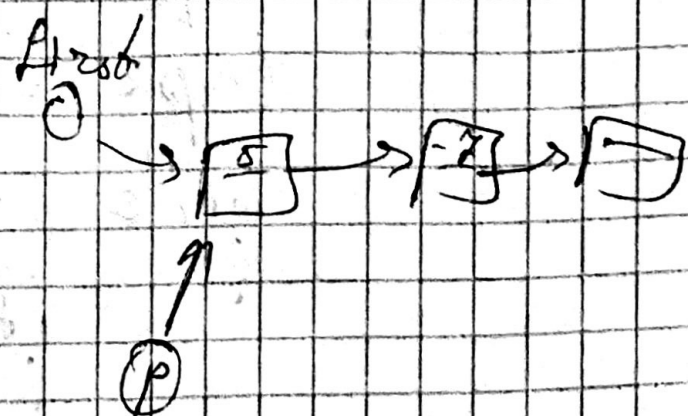
else

{

p = p->next;

}

}




```

struct item *xp, *tmp;
p = &first;
while (*p)
{
    if ((*p) -> count < 0)
    {
        tmp = *p;
        *p = (*p) -> next;
        free(tmp);
    }
    else
    {
        p = &((*p) -> next);
    }
}

```

Командная строка

vim /file.c {параметр}

vim написан на c

gcc -Wall -g prog.c -o prog -параметры

gcc написан на c

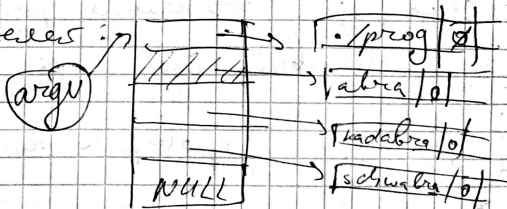
```

int main(int argc, const char* const argv)
    char** argu

```

/prog абра кадра schwaбра

Массив указателей:

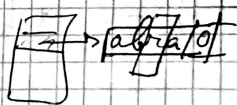


Работа с командной строкой как с массивом:

argv[1]



argv[1][2]



Семинар 4

Ф-и в графе stdlib.h

int getchar(); - ф-я с набор

из стандартного потока ввода 1 символ

#include <stdlib.h>

- для malloc

man malloc - в командной строке - вызов мануала

man 3 printf - страница по ф-и printf

серия библиотечных ф-и

1 - команды, 2 - систем. ядро

q - выйти из мануала (quit)

на самом деле 1 байт из потока

возвращает в беззнак. виде ^{код} байта 0...255

(если ф-я удалось прочитать символ)

Штатная ситуация: "конец графа"

./prog < file.txt

- прога будет читать символы ф-и из графа

В конце графа нет маркера конца графа! Нельзя прочитать конец графа!

getchar возвращает число EOF == -1 (не символ!)

unsigned char c;

c = getchar();

⇒ нельзя разместить конец графа и 255

⇒ нельзя присваивать char-у результат getchar-а

int c;

c = getchar();

{ проверка на -1 ⇒ можно работать

int putchar(int c);

- выводит на терминал

(поток стандартного вывода)

./prog > file.x - в графа

[?] puts (const char *str);

возвращает

- выводит в поток стандартного вывода строку и перевод строки.

X char *gets (char *buf); - читает заданную область памяти до перевода строки X Нельзя!

Нет ф-и для чтения строки произвольной длины!

int printf (const char *fmt, ...);

- ф-я, универсальная произвол. кол-во параметров

с т. зр. компилятора; форматная строка, имеет директивы (нач с %)

% - напечатать %

%d %i - ввести в список параметров, ввести число, напечатать в 10-м виде

%ld - long int

%lld - long long

%u - unsigned int (нб)

Форматирование: %15d - под число отведено 15 знаков, слева пробелы

%015d - слева нули, а не пробелы

%-15d - выравнивание по левому краю (пробелы справа)

%x - 16-риц. числ. систем., всегда 8/32

%X - большие лат. буквы (A, B, C, D, E, F)

%c - вывести символ с заданным кодом

%s - вывод строки: в списке параметров быть

указатель на начало стр

Можно резать строку: %10s %10s %10s

Нулевой указатель - (null) Нельзя!

`%f` - float или double

"\n" - перевод строки: читает компилятор, \n не имеет отношения к printf, она не обрабатывается ею!

табуляция

возвращает кол-во байт

scanf

`%` - прочитать данные и положить, куда указали

модификаторы: `%c` - прочитать именно этот символ

`%d` - пропустить все пробелы в потоке

"%d %d" - 10-е число, сколько угодно пробелов, 10-е число.

`int scanf(const char *fmt, ...);`

`scanf("%d %d", &, &);`

`%c`

`%c`

~~`%s` - в строке работает до первого пробельного символа~~

Пример до конца файла:

```
while ((c = getchar()) != EOF) {
```

```
    // обработка c
```

```
}
```

но без эффекта.
(оправданно!)

или:

```
c = getchar();
```

```
while (c != EOF) {
```

```
}
```

```
c = getchar();
```

```
}
```

завсегда - так себе

Задача Напиши интерпретатор командния стр

I етап: ползвателя въвежда срок, през
разделя се на слова

произв. кол-во пробелов! абреви - ка дава
и 9 - време пробелов

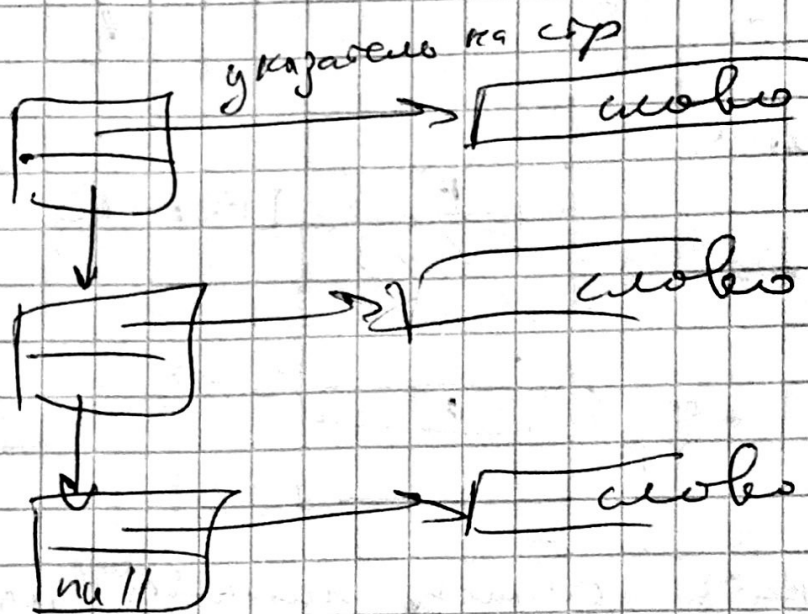
абре 'schivavra шшшш' kadavra - едно слово
с 5 пробелами

време ка въвежда не надо (по желание). " " "

време срок не завършва през, обработка
срок до конца файла

Ctrl + D

Структура:



Просто цикъл по списку, напечатать ^{срок}
очисти памет

Вход: [слово]
[слово]
?

~ 2-3 недели

Семинар 8

30.09

```
struct item {  
    ;  
};
```

```
void f(struct item *p)  
{
```

Ⓟ - значение (адрес) → []

```
}
```

```
int main()
{
```

```
    struct telen *p;
```

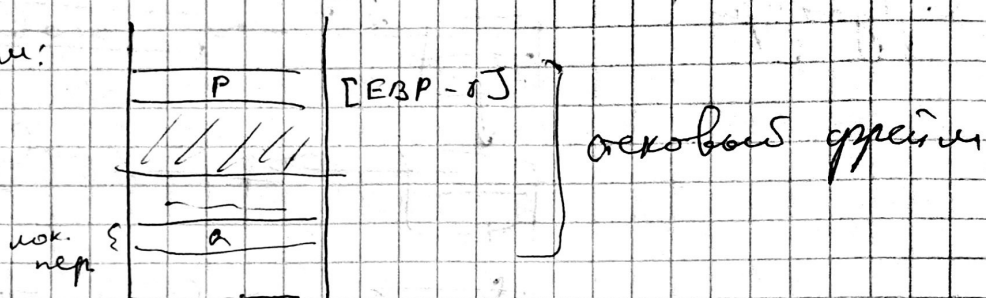
```
    f(p); // знак p!! само p не меняется
```

→ del/list(p);
p = NULL; // чтобы изменить p

```
void f(struct telen *p)
{
    ~
}
```

f(xp); - изменяется само p

основной фрейм:



чтобы изменить область памяти, надо передать адрес этой памяти в ф-ию.

изменить адрес - передать адрес указателя (запомн.)

del/list освобождает область памяти, но указатель не NULL после ф-ии.

Файлы

FILE *f - поток с т. зр. стандарт. библиотеки

(«глобальная переменная»?)

чтобы
✓

глобал. переменные:
(поток):

stdin
stdout
stderr

стандарт. в/выв

- пользователь может перенаправить

поток вывода: ./prog > file.txt

./prog | pr

- конвейер (напечатать в pr)
в prog из каталога где он в pr

⇒ поток для сообщений об ошибках: stderr

Аналоги, позволяющие читать поток

`int fputc (FILE *stream, const char *ch);`
Работает как `printf`

// На самом деле `printf` вызывает `fputc`:
`fputc (stdout, ---)`

`int fscanf (---)` - аналог `scanf`

`int fgetc (FILE *stream, ---);` - получает
из заданного потока 1 байт (или EOF, если не получ.)

`[int?] fputc (int c, FILE *f);` - байт в поток

`[int?] fputs (const char *str, FILE *stream);`
размер буфера

`char * fgets (char *buf, int buf size, FILE *stream);` (**)

⊗ гарантированно не будет завершаться. Иначе
возвращает (**), либо NULL
Будет ли больше. Иначе
будет ли больше. Иначе
будет ли больше. Иначе

Пример. `char s[512];`

`char *res;`

`res = fgets (s, sizeof(s), stdin)`

Передавая 0, поток не будет => в конце не будет перевода
строки (=> можно проверить)

Можно нормально вызвать `fputc (stdout, ---)`

`FILE * fopen (const char * (fname), const char * (mode));`

Указывает - абсолютный: первый символ /
относительное: первый символ не /, а символы
от текущего каталога (с текущ. каталога)

"file.txt"

"dir1/file.txt" - в текущем каталоге есть каталог dir1,
там file.txt

"../.. / dir2 / file.txt"

на уровень выше

"d / .. / d / .. / d / .. / file.txt"

зайти выйти

mode - что собирается делать с файлом

"r" - чтение

"w" - запись

"rb"

"wb"

} windows (без \ в пропущено drive)

Корректный текстовый файл завершается переводом стр

Если файл не открылся, форен возвращает NULL

это то : туда записывается код ошибки, если форен не откр.

напр: if (errno == ENOENT)

файла нет там, где пытались открыть

ошибка много

→

strerror(errno)

- возвращает строку, соответствующую ошибке

perror (const char *)

- печатает то, что
вернул strerror

f = fopen (fname, "r");

if (!f) {

perror (fname);

}

на экр. → file.txt: no such file or directory

fclose (FILE *f);

① Буквализация : сначала программа работает с буквами, только
после завершения может к ОС. Видеояс буквенный в начале

Обращение к ОС :

1) Заполнение буфера (ноя всегда чаше)

2) Закрытие файла (поток)

3) Явный запуск

`fflush(FILE *stream);` - всегда всё, что

есть в буфере, се

4) Перевод строки (если это идет из терминального устройства)

`printf("!!!");` $\Rightarrow$ `printf("!!!\n");`

5) При операции ввода

7/10

Семинар

Основные вызовы - г-я на Си кассиберное взаимодействие

read | работа с потоком
write

Мы открыто несколько потоков вв-выв

Стандартные по соглашениям библиотеке потоки:

`stdin` 0

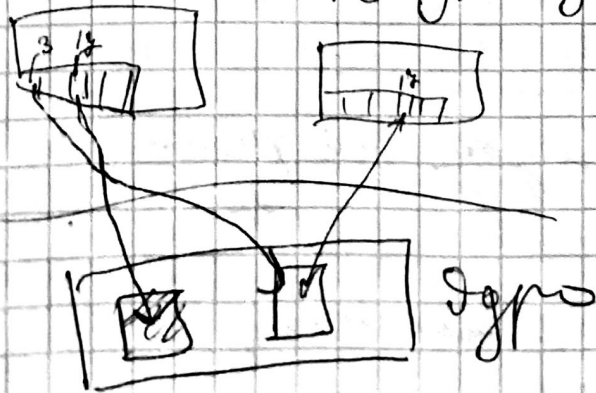
`stdout` 1

`stderr` 2

`write(1, "Hello, world\n", 13);`

1 симв. 13 символов

Для каждого процесса файловые дескрипторы хранятся
отдельно; номера могут ~~быть~~ указывать на разные
и разные структуры данных ~~одна~~



`int read (int fd, void *mem, int len);`
 fd - указатель на память, где хотим выдать прочитанные данные
 mem - ссылка на буфер

возвращает кол-во прочт. байт

file descriptor

символ

Блокировка, если недостаточно данных

Ошибка: -1 чтоно == EOF

Конец файла: возвращает 0, не трогает память!

Иногда знаком EOF бессмысленно, его не ~~н~~

`r = read(---);`



`int write (int fd, const void *data, int len);`

возвращает len

Не может вернуть 0

Ошибка: -1

может не быть

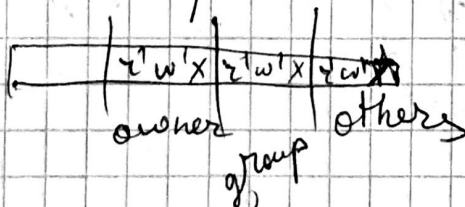
`int open (const char *fname, int mode [, int permissions]);`

mode:	O_RDONLY	константы	O_CREAT	режимы - разрешаем системе создать файл, если его нет
	O_WRONLY		O_EXCL	- требование создать новый файл (ф.с. таким именем = ошибка)
	O_RDWR		O_TRUNC	- сбросить содержимое файла, если записи

permissions - права доступа

(только на запись)

группы прав:



для польз-лей, кот. входят в группу, уяз. для файла, но не владельца

r - чтение, w - запись, x - исполнение

O_APPEND - не блокирующий режим
 O_NONBLOCK - заб `flag = 0`, UNIX игнорирует, но без него не работает

chmod - восьмеричная маска

0751 chmod -X--X

0660 0666

0600 - это основные для правки
(коп, записываем пароли)

umask - пользователь может изменить права

0660 022
644

027

077 - запретит всем все, кроме
мек, для создаваемого
графа

Проверка open на -1 !

Возвращ. наим. свободный номер потока

int close(int fd);

- обязаны закрыть поток

возвр. -1, если ошибка, 0, если успех

Запись происходит не сразу, если произошла
ошибка => close вернет ош. (пример)

lseek - посмотреть !

fcntl - с чего блокировку?

stat для информации

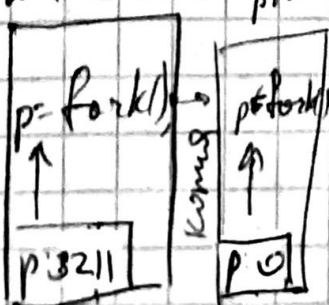
ioctl

Создание процесса

int fork();

- ОС создаёт копию процесса,
которая будет выполняться в той
же точке

ppid: 1251 ppid: 2713
pid: 2713 pid: 3211



ppid - идентификатор, который породил
этот процесс

выполнение копии после fork


```
int getpid();  
int getppid();
```

```
int main()  
{  
    fork();  
    fork();  
    fork();  
    printf("Hello\n");  
    return 0;  
}
```

- вышло 8 раз

- 1-й процесс успешен, ост. первые намерены

```
p = fork();  
if (p == 0) { /* child */  
    ...  
    exit(0);  
}
```

Лекция 10

8.10

Процесс может завершиться добровольно и по сигналу
Добровольно:

`void _exit(int code);` - обращение к ядру с
прямой записью

Ничего не возвращается, т.к. не возвр. управление
code: 0 - все хорошо

1 - не выполнили, что хотели
не -1!

Будем считать гр-д exit - возвращает буллера,
потом обращается к -exit

exit(main(_));

Процесс не завершается после завершения,
превращается в zombie. Принадлежность ему
память постепенно ~~не~~ вымывается.

Zombie нужен, чтобы сохранить игру о своем
завершении. Может сообщить её только родителю.

int wait (int * status); - возвр. ~~игру~~ №

zombie, а zombie умирает окончательно,
игра заносится в status

Если zombie неа, wait идёт, пока ~~зombie~~
завершится

Код kidного потока $\Rightarrow$ возвр -1 (ошибка)

Не интересны подробности завершения процесса:

int wait (NULL);

Unix: запуск внеш. программы: замена текущего
проц на неё

исполнение
вызов

int execve (const char * filename, const char
* const argv, const char * const envp);

исполнение
грабл
команды ср

Будем считать гр-д, содержащий окружение,
когда
execve (сохр. окружение

int execv (-u filename, -u argv);

int execvp (-u filename, argv);

имеет команду, код нужен, в системной директории
(окружении). Если есть / - не запускается, это
лишь грабл

При успехе `exec(e/p)` не возвр. управл., т.к. процесс, кол. его вызван, уже заменен другим

```

p = fork();
if (p == 0) { // child
    execvp(↓, ↓);
    perror(↓);
    exit(1); // замена на exit не увидит
              рез. perror
}
wait(NULL);
    
```

⊗ совсем правильно:

```

perror
flush(stderr); // сброс буфера
exit(1);
    
```

Как внешняя не может выполняться `cd`, т.к. она может изменить ^{текущ.} директорию

⇒ если первое слово "`cd`" и больше ничего, нельзя выполнить `cd` (`cd` без параметров)

⇒ выдать ошибку: нужны параметры (директ., куда идти)

Если `cd` с параметром:

```

int chdir(const char *dir); // см.: возвр. -1 ⇒
    
```

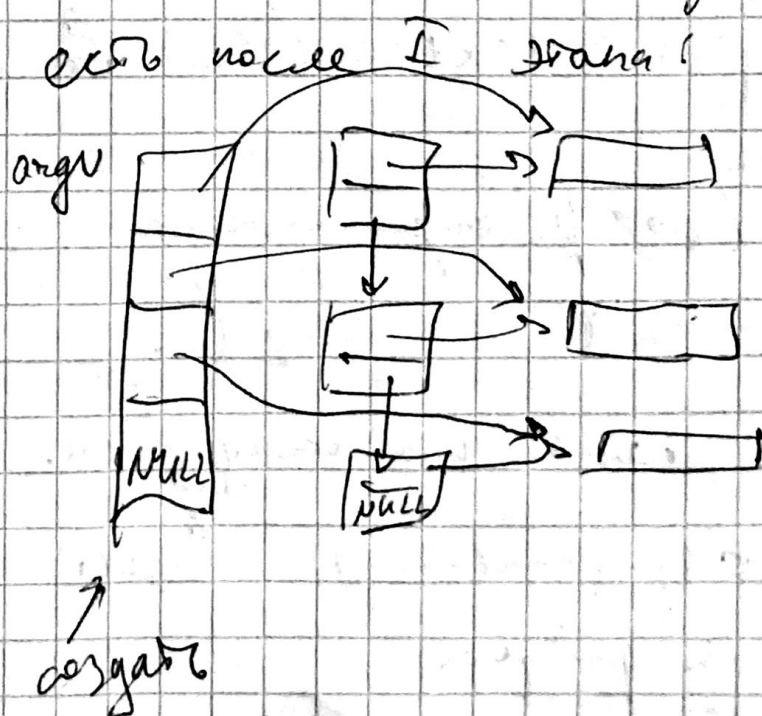
⇒ выдать сообщ. пользователю // менее текущ. директория

```

res = chdir(dir);
    
```

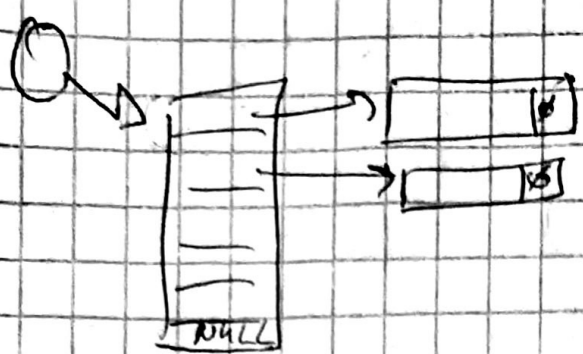
```

if (res == -1) {
    perror(dir);
}
    
```



Лекция 11

14.10



имя файла, код запуска

(1) `int execl (const char *fname, const char *arg0, ...)`

(2) `int execlp ( ... )`

сколько угодно параметров
в конце NULL

ls -l

`execlp ("ls", "ls", "-l", NULL);`

arg0 arg1

подходит только если знаем, сколько параметров

fname: (1) - абс. или отн. имя файла

(2) - поиск по системным путям

// как `execv` и `execvp`

`int wait (int *status);`

waitpid

wait3

wait4

адрес целочисл. переменной,
куда wait записал по
этому адресу код

один аргумент

`int *status;`

`int pid;`

`pid = wait(status);` // status в-мисс

сб. системная проверка, нормальный ли адрес

→

`int status;`

`int pid;`

`pid = wait(&status);`

Если процесс завершён принудительно (по сигналу),
нет кода, т.к. нет exit

его сигнал

WIFEXITED(status); (1)

if (WIFEXITED(status))

истина - заверш. сам

ложь - принудительно

WIFSIGNALED(status) (2)

истина - получил сигнал

ложь - нет

! Из одного процесса можно управлять другим
→ хакер получает контроль над 1 ⇒ может над др.

(3) WEXITSTATUS(status) - код, которым завершился
процесс (1) - истина

(4) WTERMSIG(status) - код сигнала (2) - истина

Процесс завершён успешно: (1), (3) = 0

(1) - (4) - макросы для анализа завершившегося
процесса

int wait* (int pid, int *status, int options) ⁽⁵⁾
^{struct tusage}
^{#usage;}
pid = -1 - когда кого угодно
≥ 1 - когда процесс с данным номером
0 - создать свою группу процессов
< -1 - номер группы (группа)
= 0 - ждать как обычный wait
= WNOHANG - запретить
блокироваться (ждать, пока
есть, кого ждать ⇒
цикл, удаляющий
всех записи, кто
вернёт 0 или -1
(?))

(5) если эта игра не нужна ⇒ NIKK внешнего (5)

wait 3 - не рд, ост. как wait 4

II

3 этап - фоновые процессы

терминал: `%firefox` →  запуск браузера

терминал ждёт, пока закончится процесс загрузка браузера и не принимает ур. команды

`% Firefox &` → запуск в фоновом режиме, т.е. терминал не ждёт

не надо {
 `% Firefox`
 `Ctrl-C` - приостановка (не убивается!) активного процесса
 `% bg` - продолжает в фоновом режиме (background)
 `% fg` - ???
 `jobs` - узнать список задач

4 этап : `ls` & должен быть разбителем

`a and b` - and не разделителем

`a & b` - & - разделитель

Внутри " " & - символ

Вне " " - отдельное слово

"&" - слово из одного символа

& - отл. от "&"! запись какой-нибудь программы

[k]

Запуск: проверка на наличие [&]
если [&] последний - запуск в фоновом режиме
если не послед. - вывести ошибку

фоновый режим: всё то же, что делаем, но без wait
НО wait может снять не того, кто нужен

>> xterm &

>> xterm &

>> sleep 10000 // 10000 сек

>> sleep 100 - если предыдущие уже закончили,
wait снимет их и не дождётся до нужного

⇒ циклы, снимающие всех

МБ много фоновых задач, должно быть обычных ⇒

цикл: wait 4 (-1, NULL, WNOHANG, NULL) -

- цикл, пока возвращается что-то > 0 (т.е. рд'ы)

Куда этот цикл? Куда угодно. Напр. пока есть процесс

необходимо: К/р : 3 задачи

1) считать кажд. стр, вывести длину кажд. стр,
до EOF

Рекомендация: с 1 циклом с 1 getchar

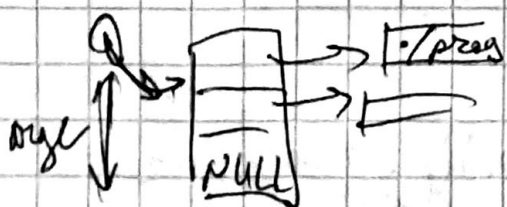
while ((c = getchar()) != EOF) // работа с EOF

2) Анализ аргументов командной строки

Работа с argc, argv // найти самый длинный
аргумент и напечатать его

3) построение списка (динамич. память только здесь!)

int main (int argc, char **argv)
≠ 0!
min 1



Перенаправление потоков в-выв

|s > list.txt - в-выв // графа перенаправляется

./prog < in.txt - в

Запуск так, что стандартный ввод заменяется на графу

./prog >> file.txt - в конец файла

./prog1 | ./prog2 - конвейер: одс прог

запускаются одновременно; то, что 1 выводит на два стандартных вывода, 2 выводит в стандарт.

4 этап: перенаправление потоков

Создание нового потока $\Rightarrow$ ему присваивается свободный номер. (дескриптор)

close(0); - закрыли поток №0.

если после открытия графы, он станет потоком №0 (стандартным)

Сд. поток №0 !!! Его можно менять, команды printf, scanf и т.д. не заметят

fd = open(...); Сд. в-выв. - №1

Графа может не открыться $\Rightarrow$ сд. поток остался закрытым $\Rightarrow$ проверка?

fd = open(...);

/ проверка fd /

close(fd);

close(1);

fd = open(...);

проверка, затем открыть как нужный поток

ПЛОХО

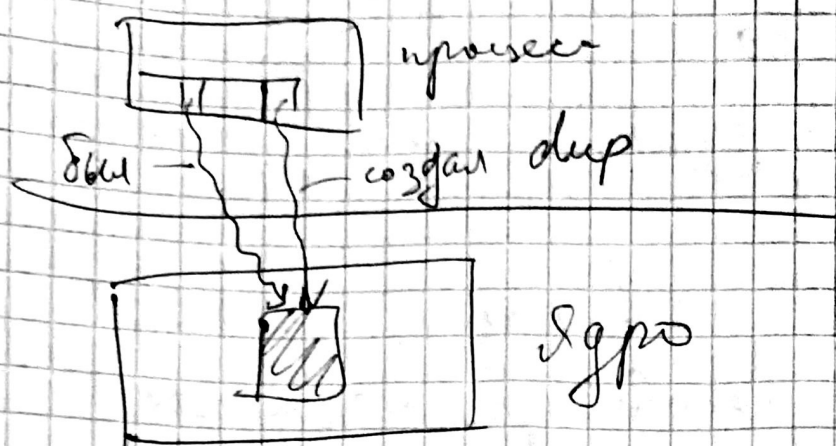
возможно в это время кто-то

удалит графу

некорректная ситуация

```
int dup (int old fd);
```

создает еще один номер для того же потока (аноним)



Поток указывает, только если дальше нет
таблицы дескрипторов, связанного с ним $\Rightarrow$
close удаляет только номер, не поток

очень редко | Ошибка: потеряны доступные номера; вместо
дескриптора подана хрень

```
fd = open(---);
```

```
if (fd == -1) {
```

```
}
```

```
close (1);
```

```
dup (fd);
```

// вернет 1

```
close (fd);
```

exec не меняет таблицу дескрипторов $\Rightarrow$ не меняются
поток 0, 1, 2

```
pid = fork();
```

```
if (pid == 0) { x/chl/dl x/
```

```
open / dup / close
```

- подготовка среды

```
exec
```

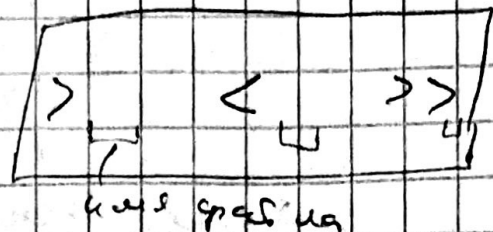
```
}
```


1nd dup2 (1nd old fd, 2nd new fd);
- сс new fd закрывается (не нужен close)
dup2(fd, 1); - ссг. буф

// dup[licate] to
// wait4() → wait for
 4 параметра

добавить разделение:

&



5.5 then

> < могут сочетаться

< >> могут

> >> не могут

после нужно дать имя файлу

< open(, O_RDONLY)

>

O_WRONLY | O_CREAT | O_TRUNC

>>

O_WRONLY | O_CREAT | O_APPEND

0666

Сигналы

SIGKILL 9

Состояние процесса D: Ядро запустило свободное состояние, процесс не может завершиться

9 удивляет наверняка: как только процесс

~~будет создан~~ получает этот сигнал, он больше не вернется к исполнению.

SIGTERM 15 - можно перехватить

kill 2735 - прислать процессу n 2735 сигнал

сигнал (по умолчанию 15)

kill -9 2735

- SIGSEGV 11 - ошибка выделения памяти?
- SIGPIPE 13 - попытка записи в канал, которого нет

SIGINT - Ctrl+C

SIGQUIT - Ctrl+Q - сбрасывает стек и core

prog.core: сегмент данных и стек
(имеет ценность, если анализировать сразу после падения проги)

- можно использовать, когда перехвачен SIGINT

Группа сигналов: ~~SIGSEGV~~ - процесс пытается

выполнить недоступное ему действие → вынужд. прерывание

SIGSEGV - нарушение защиты памяти

SIGILL - попытка выполнить код, когда не явл.

SIGBUS - ???

SIGFPE - исключение операций с плавающей точкой

SIGSTOP - остановка процесса // нельзя сопоставлять, как и 9

SIGCONT - продолжить процесс

SIGCHLD - посылается процессу, когда заверше (или в ловушке) его потомок

SIGUSR1 |
SIGUSR2 | - пользовательские сигналы

SIGKILL - активная группа процессов получает его, если закрывается терминальное окно

SIGALRM - просит систему принять 1/3 и сек; если успешно завершить вычисления отсчета

! man 7 signal - показать сигналы

Послать сигнал:

`int kill (int pid, int signo);`

(^{№ сигнала} процессу, которому посылаем сигнал)

`pid signo = SIGTERM`

`pid: ≥ 1` - номер процесса

`0` - себя группа

`-1` - заданной группе

`-2` - "все"

Семинар 13

28.10

К/р (ноябрь - декабрь):

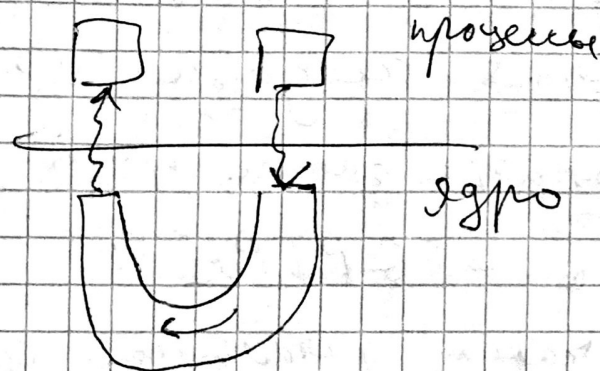
1) Сигналы (попробовать! самая простая задача)

2) Каналы

3) Управление процессами

Канал - объект ядра

1 поток ввода и 1 поток вывода



У канала есть собств. буфер (ограничен!)

Каналы → ^{лекс.} именования - `fd[0]`
→ ^{лекс.} неименов. `pipes`

`mkfifo` - открыть канал

`pipes` - системный вызов!

`int pipe (int fd[2]);`

`int *fd`

показ., что опис. массив из 2 эл-тов

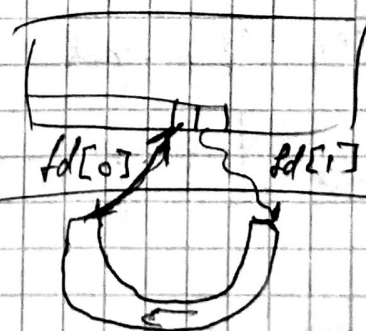
- с т. зр. компилятора тоже верно

два дескриптора

`fd[0]` - чтение

`fd[1]` - запись

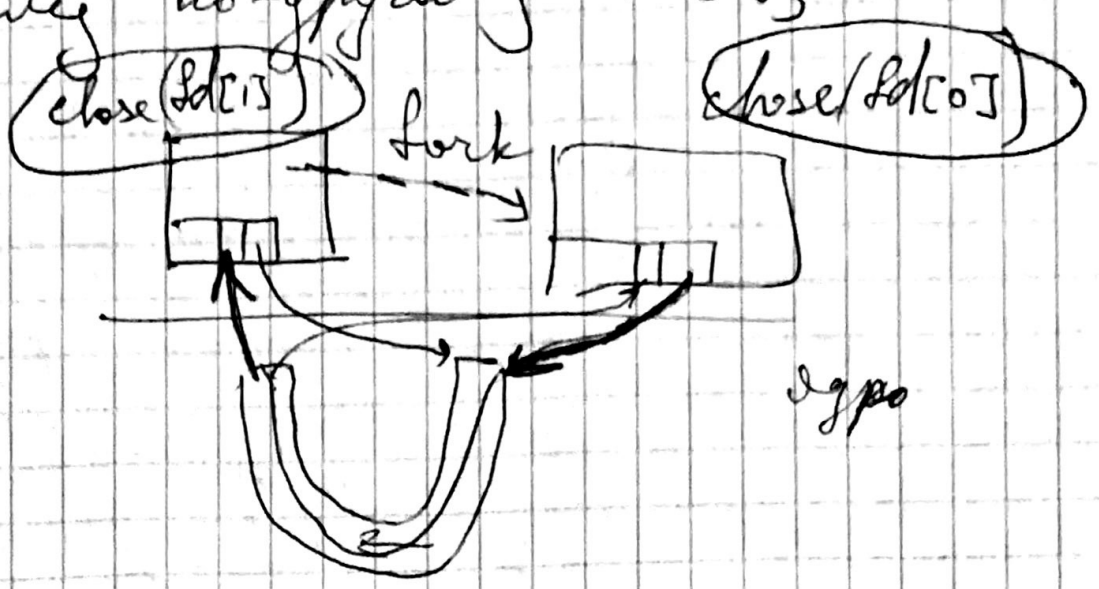
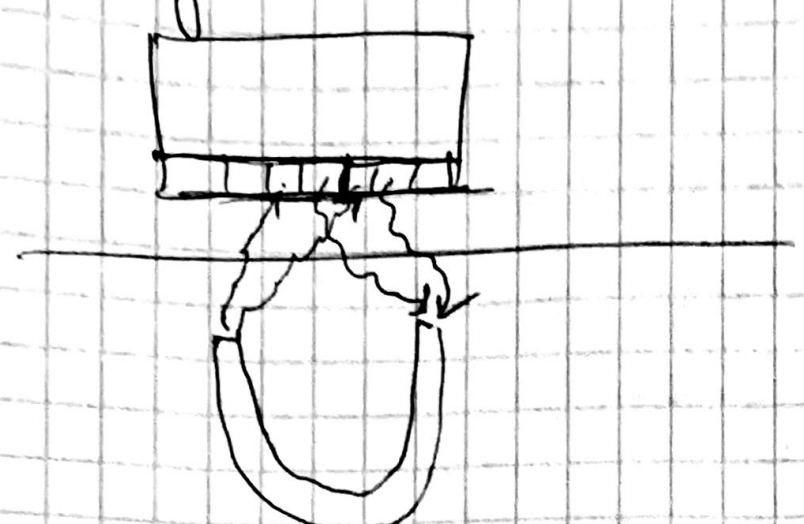
(ам с собой процесс не общается)



pipe - средство односторонней связи процессов:

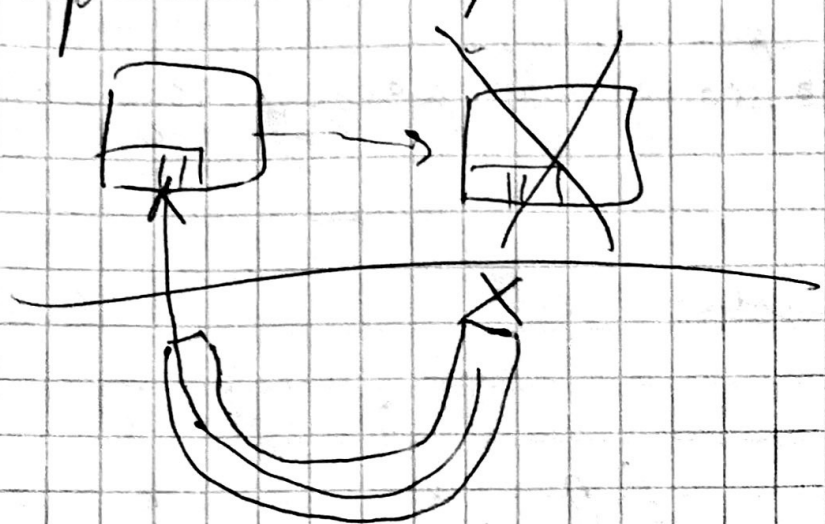
fork() - связь с тем же каналом

Подключиться к каналу по-другому нельзя

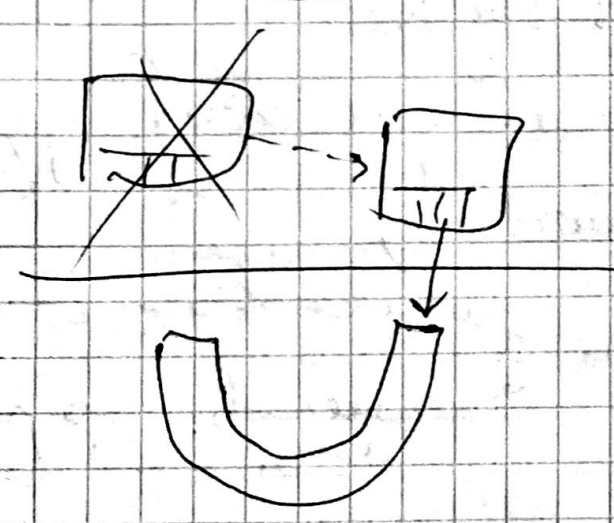


Закрывать лишние дескрипторы!

Процесс завершается -> дескриптор исчезает



ситуация конца графа



некорректная ситуация ->

При попытке записи придет

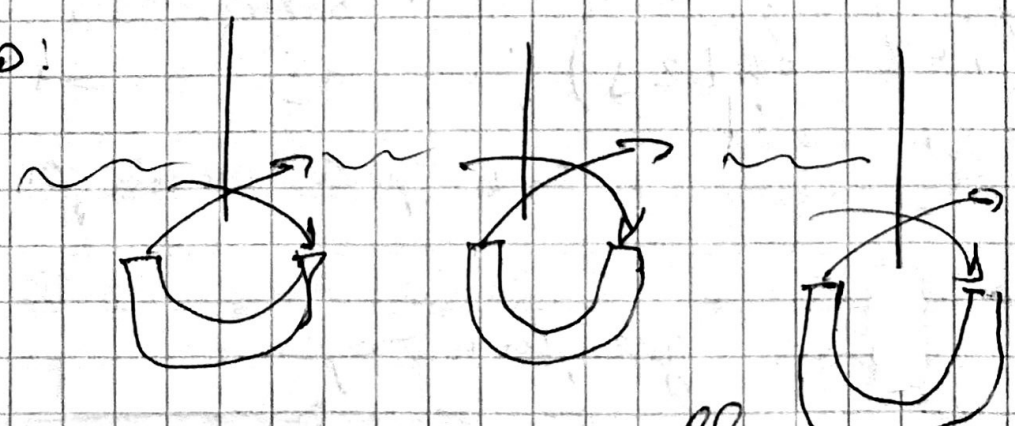
сигнал SIGPIPE!

по умолчанию fatalный сигнал ->

судает процесс, если не перехватить

Перехват: можно игнорировать

Конвертер:



cat без параметров - стандарт. вв

yes | cat | cat | cat | we

yes | cat | cat | cat | cat > /dev/null

ос. эта =>
задание закрыть на
тесты

сохранились
эта => не закрыл деск. на записи

не вернуть потоки

Не фоновый конкурент: нужно заполнить все pld's

Массив pld'ов: wait() = pld => нашли pld

в массиве => заменили на 0

Во все 0 => конкурент завершён

Можно сделать n | n => по ндтк, как
сделать цикл

① Указатели на ф-и и перехват сигналов - почитать

Семинар 14

29.10

Указатели на ф-и

char f(int x, double y)

ук: char (*p)(int, double); - описание переменной

параметр должен быть 1 => не указ.

* должна применяться раньше () после

typedef char (*funptr)(int, double);

имя типа

funptr p;

p = &f; ≡ p = f;

c = (*p)(3, 2.5); - базов ф-и ≡ c = p(3, 2.5);

Демонстрации: (что-то делать, когда увидит сигнал)

SIG_IGN

- игнорировать

SIG_DFL

- по умолчанию (напр., неслы и хотим
вернуть обработку сигнала)

базов ф-и

```
void hdl(int n)
{
    // номер сигнала
}
```

Нужно передать указатель на ф-ию в sigp

```
typedef void (*sighdl)(int);
// sighdl p = hdl;
sighdl signal(int signo, sighdl dsp);
                SIG-INT      SIG-DFL
                SIG-IGN      hdl
```

signal(SIG-INT, hdl); - обращение к ОС: если придет SIG-INT, вызвать hdl

hdl работает с глобальными переменными!

Многоразовый однократный

```
void hdl(int n)
{
    signal(n, hdl);
}
```

В реальности с ф-ей signal лучше не работать, лучше использовать $\pi/3$ signal

signal возвращает предопределенное значение

SIG-DFL - 0, приведенный к типу sighdl

IGN - 1

ERR - -1

sig_atomic_t - тип переменных, изменение хв. атомарно, т.е. либо выполнится, либо нет, за 1 действие (нет коллизии)


```
void hdl(int n)
{
    // номер сигнала
}
```

Нужно передать указатель на ф-ию в сигн

```
typedef void (*sighdl)(int);
// sighdl p = hdl;
sighdl signal(int signo, sighdl dsp);
                SIG-INT      SIG-DFL
                SIG-IGN
                hdl
```

signal(SIG-INT, hdl); - обращение к ОС: если придет SIG-INT, запустить hdl

hdl работает с глобальными переменными!

Многократная обработка

```
void hdl(int n)
{
    signal(n, hdl);
}
```

В реальности с ф-ей signal лучше не работать, вместо использовать $\pi/3$ sigaction!

signal возвращается предыдущую функцию

SIG-DFL - 0, приведённый к нулю sighdl

IGN - 1

ERR - -1

sig_atomic_t - тип переменной, приваивание к ней атомарно, т.е. либо выполнится, либо нет, за 1 действие (нет коллизии)

prints в обработчике сигнал использовать
printf, т.к он обрабатывается и буферизуется
→ write

```
sig_atomic_t count = 0; // сколько раз пришел сигнал  
const char msg[] = "press it again, I like it \n";
```

```
void hdl(int n)
```

```
{  
    signal(n, hdl);
```

```
    count++;
```

```
    if (count >= 25)
```

```
        exit(0);
```

```
    write(1, msg, sizeof(msg));  
}
```

```
int main()
```

```
{  
    signal(SIGINT, hdl);
```

```
    for (i=0;
```

```
        sleep(1);
```

```
    return 0;  
}
```

// Бесконечный цикл, не сильно
нагружает процессорное
время
// sleep прерывается сигналом

Ctrl + C 24 раза, на 25 прог выключается

Аналог (по сообщению 25):

```
void hdl(int n)
```

```
{
```

```
    signal(n, hdl);
```

```
    count++;
```

```
    write(1, "\n");  
}
```

```
int main()
```

```
{
```

```
    signal(SIGINT, hdl);
```

```
    while (count < 25)
```

```
        sleep(1000000);
```

```
    return 0;  
}
```


int pause(); - ждем (спать), пока не придет сигнал
// возвр. всегда - 1

alarm - прочитать

Семинар 15

11.11

Сервер: методика (шаги 1-3, 4 не надо)

Макропроцессор

Компилятор: преобразование текста в текст

2483 ^{4 симв.} компилятор → 1 число → транслит → 1 символ!

Аналог. ключевые слова, напр. while

25436

:=

dd

my_variable

"авра кадра" - строковая константа тоже 1 лексема!

#include <stdio.h> - <> для заголовков систем. файлов

#include "math.h" - " для своих ^(для всего внешнего по отношению к проге)

спасибо ищите в той директории, в к-ой подключает компилятор

#include - директива

NULL - макро вызов (макрое)

NULL == ((void *)0)

0_RDONLY превращается в ноль при транслитации (?)

0_WRONLY макросы

WNOHANG

Сигналы

Если строка начинается с #, она все - директива ^{макро} процессора

#define NULL ((void *)0)

Этика: ~~также~~ называют макросы заглавными буквами

#define x 15

— задан макрос

void f(—)

{
 int x;

}

⇒ int 15; — будет компилятор на этапе синтаксического анализа → треб. идентификатор

gcc -E | less

Макрос с параметрами

#define MAX(x,y) ((x)>(y)?(x):(y))

сравнить все скобки!!!

$x > y ? x : y$

MAX(10+5, 20+7) * 3

⇓
10+5 > 20+7 ? 10+5 : (20+7*3)

Условная компиляция

на уровне макропроцессора

на директивах

#if 0

:

#endif

] кусок кода не компилируется
(когда нужно закомментировать кусок кода с комментарием, напр.)

#if 1

1

⇒ компилируется

#if 1 2 0

#else

#endif

#elif

#if defined(MYSYMBOL)

#if !defined(MYSYMBOL)

— превращ. в 1, если макрос есть, в 0, если нет

— наоборот


```
#ifndef MYSYMBOL
#define MYSYMBOL
#endif
#endif
```

Отладка (вместо printf - ob):

```
#ifdef DEBUG
printf(--)
#endif
```

В начале проги: #define DEBUG
/* */ - если нужно убрать

Менять код некрасиво => вместо *:

```
gcc -Wall -g -DDEBUG
```

Для разных заказчиков: #ifndef FOR_SIDOROV

Перенос из-под Линукса под Винду: MinGW

Макрос -- WIN32 --

компания под Виндоу => определяет
Линукс => не оур

Линкер

Модуль (obj) на ассемблере => id в явном виде
на C => gcc вызывает id, как надо

```
gcc -Wall -g -c mod1.c
```

не выполняет динамическую линковку: оставил
объектный модуль как есть

* заметка на 0 -> -o не надо

В модуль можно вынести ф-и, глобал перем,
описание типа


```

prog.c
void func1(int x, float y); { вызов }
int main()
{
    func1(-);
}

```

```

mod1.c
void func1(int x, float y)
{
}

```

Разделение трансляции! Компилятор запускается по отдельности, linker рассовывает адреса

3 в mod1 15 функций, их можно использовать и другие → они не работают и мы не можем их использовать
 ⇒ делаем заголовочный файл где mod1

```

mod1.h
void func1(int x, float y);

```

```

prog.c
#include "mod1.h"

```

```

mod1.c
#include "mod1.h"

```

Проблема перекрест:

```

mod2.c
int my_bad_global_var; - описание пер.

```

```

mod1.h

```

```

extern int my_bad_global_var; - объявление

```

Описание типа, макросы → в заголовочном mod1.h
 (все, что не занимает памяти)

```

mod1.h

```

```

struct myst {
};
// и, конечно, myst

```

```

mod2.h

```

```

#include "mod1.h"
struct best_str {
    struct myst s; // none
};

```


prog.c

```
#include "mod1.h"  
#include "mod2.h"  
;
```

→ не скомпилируется, так
mod1.h подкл. дважды (в
кажд. б.б.) → 2 описания типа

Мешается с комментариями, можно
на #include → защита от
повторного включения!

mod1.h

```
#ifndef MOD1_H_SENTRY  
#define MOD1_H_SENTRY  
{
```

//защита от повторного включения модуля

```
#endif (конец модуля)
```

↑ используемый хак, но это не знает, но он хорош!

~~gcc mod1.o prog.o -lm -o prog~~

gcc -Wall -g mod1.o prog.c -lm -o prog

Много модулей для компиляции →

правила Makefile

→ команда make

Семинар 16

К/р:

1) Переопределить потоки и что-н. сделать

2) Сигналы: а/ачт: если прошло больше 1 сек -
высветит сообщ., меньше - что-то делать

Знать: signal

kill

alarm

raise/sleep

92

// Номера сиск. знать по падо,
имена содержат в заголовках

fork

execp/execvp 81

wait/pwlohang 82

exit/-exit 84

pipe

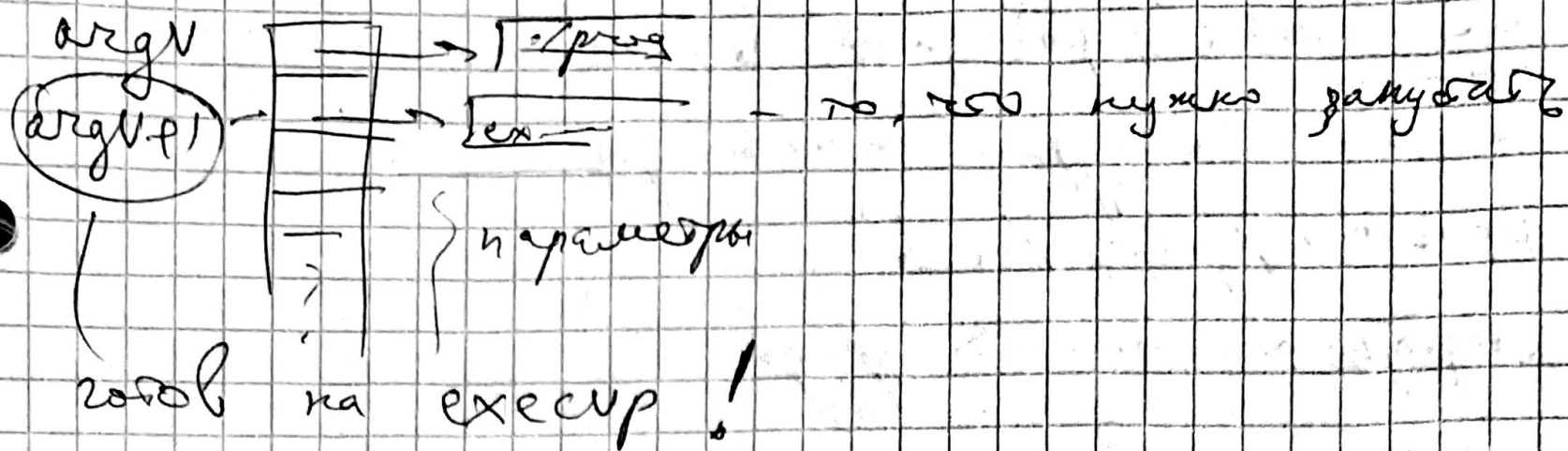
open, close

read, write

dup2

50/54

52



Сокеты

Четкого определения нет!

Объект сокета - некая структура данных

(сокеты связ. файловый дескриптор)

Всегда можно сделать с `fd: dup/dup2, close`.

Протоколы - набор соглашений

Нужна комп. сеть $\Rightarrow$ сокет

В разных протоколах адреса выглядят по-разному

IP-адрес: 192.168.218.12

195.42.160.101

- адрес машины, на к-ой стоит столукасов 1480

идентифицирует интерфейс компа

1 комп. интерфейс имеет свой IP-адрес

Идентиф. сокета + порт к IP-адресу

195.42.160.101:7738

- адрес сокета

IPV4

порты с 1 до 1023 - привилегированные (только для супер-пользователя)

IPv4: заканчиваются IP-адреса $\Rightarrow$ надо переходить

к IPv6: такие же порты, др. IP-адреса

17E8:AF0B:100F:3F07:7738

$(2^{64})^2$ адресов

порт

использования вызовов функции работают с адресами

и IPv4, и IPv6

Локальные сокеты - работают в данной системе, не
могут по сети. Их адрес - числ. графа
" /var/run/proc/sock "

Кдр. информационная среда

int socket (int family, int type, int protocol);

указ. <= | семейство протоколов / система адресации
тип | - разные понятия => 2 константы; протокол, число;
адреса

AF_INET

PF_INET

AF_INET6

PF_INET

- IPv6

AF_UNIX

PF_UNIX

AF_LOCAL

PF_LOCAL

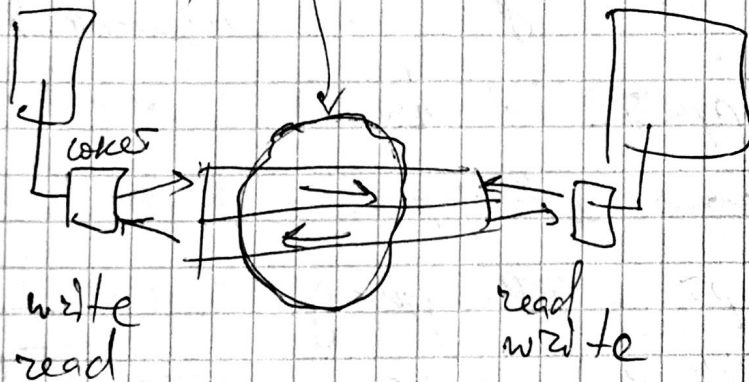
- локал. сеть

type : SOCK_STREAM

SOCK_DGRAM -> DNS

SOCK_STREAM:

машинка



двухнаправл. канал

- Будет использоваться
(в осн. использ. в машин.)

protocol : IPPROTO_TCP ≡ 0

Возвр: -1 - ош., fd - выполненная

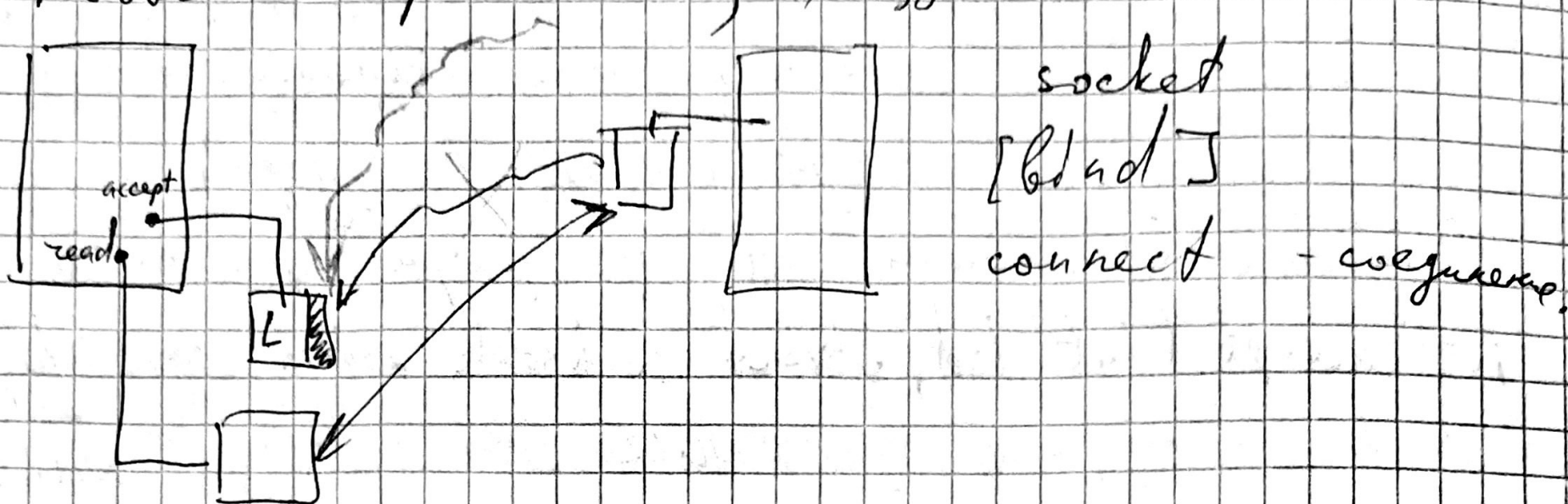
Сервер устанавливает соединение, ждёт, пока
к нему обратятся

"Клиент-сервер"

Клиент обращ. к серверу, он ~~создаёт~~ прикрепляет
адрес к созданному (socket) адресу. Создание адреса
соз. вызов bind

Приём запроса на соед: listen

Отправка запроса на соед: accept (возвр.
новый номер сокета) // создаёт ещё 1 сокет



2 сокета - проблема очередности действий

Организованный процесс на комп. подключен - 2 клиента
telnet 4664

```
int bind (int sd, const struct sockaddr *addr,  
          socklen_t addrlen);  
int
```

```
struct sockaddr_in addr;
```

```
addr.sin_family = AF_INET;
```

```
addr.sin_addr.s_addr = INADDR_ANY;
```

```
addr.sin_port = htons(4653);
```

г-я, меньше 2 байта

(номер порта)

little endian - персональные компьютеры

big endian - серверы, процессоры

```
=> bind(sd, (struct sockaddr *) &addr, sizeof(addr))
```

bind может вернуть -1!!!

```
=> if (-1 == bind(...))
```

```
{
```

return

```
}
```

Если не сделано это, остальные шаги вызовут не
будут работать

int listen (int sd, int ¹qlen);

пример непрерывных запросов на соед.?

(10) - мало, сколько имеет эдро?

(если указать число больше, чем имеет эдро, ошибки не будет)

Ошибка: не написать, sd - не дескриптор сокета

int accept (int fd, struct sockaddr *¹addr, socklen_t *²addrlen);

NULL

NULL

возвр. дескриптор нового сокета, который связан с клиентом

После канг. вызова сокетов на 1 больше — на записи

int select (int max_fd, fd_set *read-set, ¹fd_set *write-set, ²fd_set *except-set, struct timeval *tv);

max дескриптор + 1
представление: подбитые сроки; канг. возможны дескриптор - 1 дит

(*) готовность по сокету = можно accept

(1) NULL

(2) узнать, если произошел таймаут?

fd_set s

FD_ZERO (&s);

FD_SET (n, &s);

- ввести др и P можно

FD_UNSET (n, &s)

FD_ISSET (n, &s)

(2) fd_set
fd_set

| узнать на ~~ка~~ время дошло сек

Мн-во останется только те дескрипторы, на
которых что-то произошло

Для комп. времени нужно помнить, что уже
принято от кого

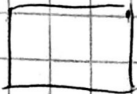
Читать книгу !

Семинар 17

25.11

select вернул $> 0 \Rightarrow$ что-то произошло, в мн-ве
дескрипторов остались только те, дескрипторы, с кем
что-то произошло

Если дескриптор готов $\Rightarrow$ read



for(;;) {

// бесконеч. цикл

выборка
select

обработка

}

(также та структура)

Реальность: графич. интерфейсы

но факту программа имеет

обработку, основное действие
(маш. язык) лежит в библиотеке

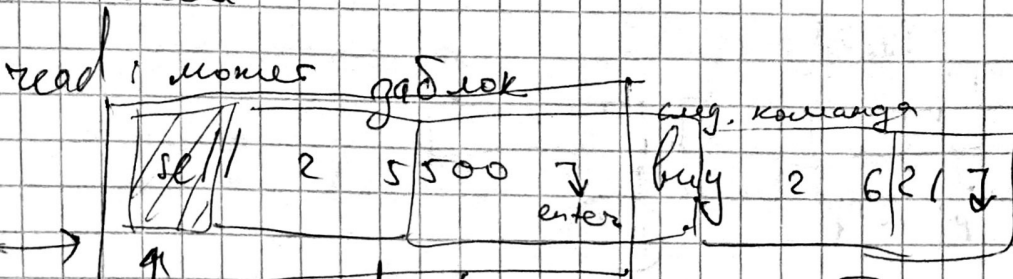
Задерживаться в обработке надолго - дурная тон!

(граф. интерфейс тормозит на 0.5 сек. $\Rightarrow$ пользоваться? хз.???)

Нельзя в обработке: сист. вызовы и др.-и, как.

могут обратиться к сист. вызову $\Rightarrow$ лб блокировка

! Нельзя допустить блокировку внутри обработки
сообщения



первый read (сообщение лб рудито как угодно)

$\Rightarrow$ не имеем права делать след. read, т.к.

находимся в обработке сообщения

работа с неблокир. каналом не зависит ни от чего,
в т.ч. от пропускной способности

Браузеры (масс программ) тормозят (напр. анимация)
→ более быстрый браузер не поможет!
Эмьюлятор

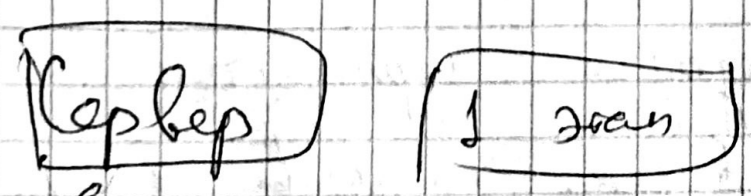
Храним команды (go ↓) - массив шагов!
полный адрес и размер / размер буфера

Читаем: buf + count, size - count Прошлого (*)
буфер / размер, сколько уже записано
(адрес начала массива)

Читаем, пока не кончатся переводы строки в буфере?
(⇒ больше нет команд)

Диалог с пользователем: нужно после key, read
возвр. управление в главный цикл ⇒ почитать,
на чём остановились

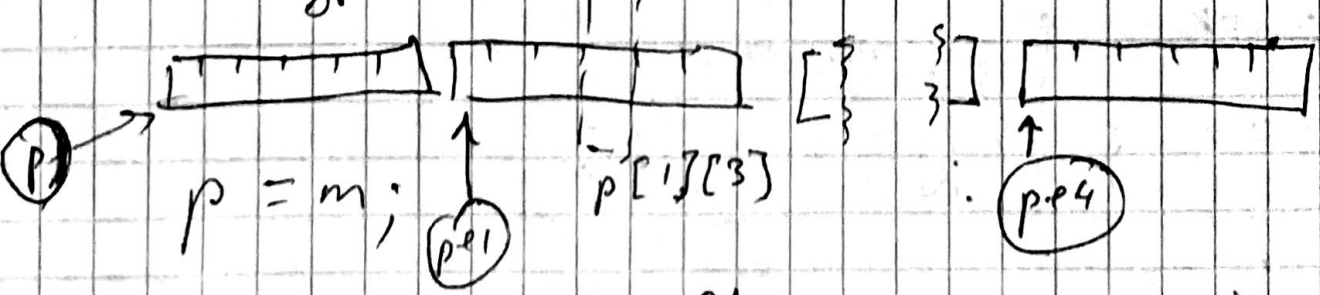
Запоминание: предусмотреть соотв. поля в структуре данных



Есть 1 целочисл. перемен., её можно ↑ и ↓ на 1 командами

double m[5][6];

m - адрес? m[1][3]



Кеверно: double *p; p имеет др. тип
если бы тип был такой, создавался бы доп. массив
указателей

Связь p на 1 приводит к связи на строку матрицы ~~тогда~~ этого нужно знать размер строки)
 $* (p+1) \equiv p[1]$

Описание типа: `double (*p)[6];`

p - указатель на массив uz в int -ов типа `double`

`double m[5][6][7];`

`double (*p)[6][7];`

3-мерный массив

Заг. Ф-я получает указатель на матрицу, ~~возвр.~~ ищет max, возвр. строку, в к-ой находится этот int -г

`double m[5][6];`

`double (*p)[6];`

$p = m;$

`double (*maxline (double (*matrix)[6]))[6];`

// ф-я применяется как параметр
указатель на массив `matrix`, возвр.
`maxline` того же типа

`typedef` $\Rightarrow$ упрощение (нормальное `typedef`)
`typedef double (*d6p)[6];` $\Rightarrow$ `d6p`, `d6p` `matrix` (?)
maxline

указатель на ф-ю: $fp = \text{maxline};$

`double (*(*fp)(double (*)[6]))[6];`

нельзя имя кроме fp

fp - указатель на массив типа `m`, возвр.
указатель на массив uz с int -ов типа `double`

`#typedef double (*double_ptr)[6];`

`double_ptr p;`

$p = m;$

`double_ptr maxline (double_ptr matrix);`

`double_ptr (*fp) (double_ptr);`

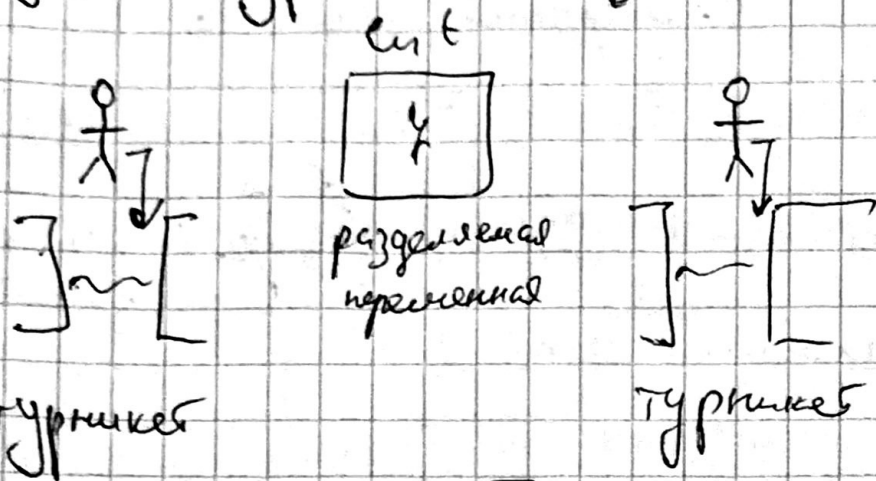
Семинар 8

2.12

Семафоры -

// Не бывает двойных!! Двойные mutex (реальность vs лекция)
- разделяемые данные, к-е могут быть модифицированы
Некая ф-я запускается // остальные прога

Задача. Турникеты: чтобы знать, сколько людей в здании



```
(1)
mov eax, [cnt]
inc eax
mov [cnt], eax
```

```
(2)
mov eax, [cnt]
inc eax
mov [cnt], eax
```

eax [8]

исполн. кода времени, т.е.
процесс первый (он об
этом не знает) ради
выполнения др. процесса
⇒ сброс eax

нет процесс успешно завершится
⇒ eax = 8

eax [8]

⇒ eax из 1-го процесса записывается
⇒ 4 + 1 + 1 = 8

Арифметика на грани безопасности © Солерв

Критическая секция: 1) фрагмент кода
2) время от начала выполнения до конца
- фрагмент времени

sem [1/0] 0 - нельзя работать
1 - можно

down(sem)

(1)

up(sem)

down(sem)

(2)

up(sem)

Корректная работа, если up и down - атомарные операции

Если это системные вызовы, то на самом деле созданы каналы $\Rightarrow$ разделяемая память теряет смысл!

⊛ передаётся 1 байт, $mb \neq 1$

Преимущество разделяемой памяти - скорость работы $\Rightarrow$ не нужно обращаться к ОС!

sem=0 $\Rightarrow$ down $\Rightarrow$ нужно заблокироваться $\Rightarrow$
 $\Rightarrow$ sleep $\Rightarrow$ обращение к ОС (+ проснуться) -
единственный случай, когда нужно обр. к ОС

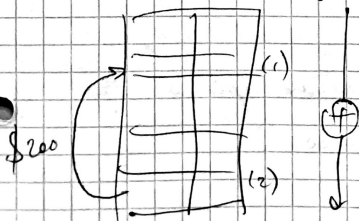
mutex

Задача о 5 обедающих философах: посадить за стол 5 человек
 $\Rightarrow$ комбинация ног (правая sinistra берет левую вишню, левая наоборот)

5 - min число, когда возникает нередукцируемость

Решение из книги Таккенбаума: нб процесс, очередь до которого не дойдёт
банковские счета

н счета кол-во денег



Кто-то запустил процесс Σ определённой категории счетов

В это время кто-то решил перевести с 1 счета на др.: когда привнесена (1),

деньги куда еще не пришли, $\pi(2)$ - деньги уже
ушли

$\Rightarrow$ мб плохо, даже когда один модифицирует
данные, остальные читают

Если есть данные, к которым нужен доступ > 1 раза,
и хотя бы 1 мб может модифицировать $\Rightarrow$ проблемы
Корректного решения нет!

IPC

Лабораторная (полезно для экзаменов)

Разделяемая память на 10 кусков

0	1	...	9
---	---	-----	---

 - как обычный массив

Осн. прога: цикл 2 сек, пробегает по массиву и
печатает сумму

2ая прога: `randseed`
`rand() % 10`

Подключаясь к раздел. памяти, выбираем 2
случайных числа от 1 до 10 (2 эл-та массива)

Цикл: если числа одинаковые $\Rightarrow$ continue

Один эл-т $\neq 1$, другой -1

0	1	...	9
---	---	-----	---

Результат: иногда вместо 0 вылезает $\neq 1$

Запустить 2ую прогу ещё раз из др. окна (т.к. работающая)

$\Rightarrow$ большие числа вместо суммы

Добавить самодур

`ipcs` - команда, показывающая графы ipcs в системе
`ipcs -t` - удалить ненужные процессы?

Семантика: senget
 senop
 senct/

Разг. нумер: shngt
 shuct/
 shmat

- возр. адр, куда присоед. нумер

$\text{ftok}(\text{lname}, \text{char})$