

Семинар №9

ФАКИ 2017

Бирюков В. А.

November 10, 2017

Память под локальные переменные

```
int sum(int a, int b)
{
    int s = a + b;
    return s;
}

void print_sum(int a, int b)
{
    int p = sum(a, b);
    printf("Sum = %d\n", p);
}

int main()
{
    → int x, y;
      scanf("%d%d", &x, &y);
      print_sum(x, y);
      return 0;
}
```


main()
Переменные x и y

Память под локальные переменные

```
int sum(int a, int b)
{
    int s = a + b;
    return s;
}

void print_sum(int a, int b)
{
    int p = sum(a, b);
    printf("Sum = %d\n", p);
}

int main()
{
    int x, y;
    scanf("%d%d", &x, &y);
    print_sum(x, y);
    return 0;
}
```



main()
Переменные x и y

Память под локальные переменные

```
int sum(int a, int b)
{
    int s = a + b;
    return s;
}
void print_sum(int a, int b)
{
    int p = sum(a, b);
    printf("Sum = %d\n", p);
}
int main()
{
    int x, y;
    scanf("%d%d", &x, &y);
    print_sum(x, y);
    return 0;
}
```



print_sum()
Переменная p

main()
Переменные x и y

Память под локальные переменные

```
int sum(int a, int b)
```

```
{
```

```
    int s = a + b;
```

```
    return s;
```

```
}
```

```
void print_sum(int a, int b)
```

```
{
```

```
    int p = sum(a, b);
```

```
    printf("Sum = %d\n", p);
```

```
}
```

```
int main()
```

```
{
```

```
    int x, y;
```

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

```
    print_sum(x, y);
```

```
    return 0;
```

```
}
```

sum(int, int)

Переменная s

print_sum(int, int)

Переменная p

main()

Переменные x и y

Память под локальные переменные

```
int sum(int a, int b)
```

```
{
```

```
    int s = a + b;
```

```
    return s;
```

```
}
```

```
void print_sum(int a, int b)
```

```
{
```

```
    int p = sum(a, b);
```

```
    printf("Sum = %d\n", p);
```

```
}
```

```
int main()
```

```
{
```

```
    int x, y;
```

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

```
    print_sum(x, y);
```

```
    return 0;
```

```
}
```

sum(int, int)

Переменная s

print_sum(int, int)

Переменная p

main()

Переменные x и y

Память под локальные переменные


```
int sum(int a, int b)
{
    int s = a + b;
    return s;
}
void print_sum(int a, int b)
{
    int p = sum(a, b);
    printf("Sum = %d\n", p);
}
int main()
{
    int x, y;
    scanf("%d%d", &x, &y);
    print_sum(x, y);
    return 0;
}
```



print_sum(int, int) Переменная p
main() Переменные x и y

Сегмент памяти: Стек вызовов

```
int sum(int a, int b)
{
    int s = a + b;
    return s;
}
void print_sum(int a, int b)
{
    int p = sum(a, b);
    printf("Sum = %d\n", p);
}
int main()
{
    int x, y;
    scanf("%d%d", &x, &y);
    print_sum(x, y);
    return 0;
}
```



main()
Переменные x и y

Управление памятью

Сегменты памяти процесса

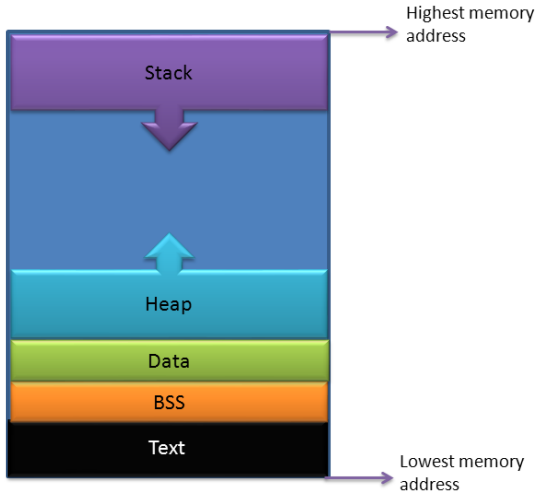


Figure : Process memory organization

Сегменты памяти: Стек вызовов (Call stack)

- Стек представляет собой обычный алгоритмический стек, применённый для управления памяти
- В нём хранятся локальные переменные
- Имеет фиксированный размер, определяется операционной системой, на порядок меньше чем Куча
- Быстрее, чем Куча

Сегменты памяти: Куча (Heap)

- Куча представляет собой обычную алгоритмическую кучу, применённую для управления памяти
- В ней можно динамически выделять память
- Размер, обычно, ограничен только доступными ресурсами
- Медленней, чем Стек

Below Your Program

High-level
language
program in C

```
void swap(int v[ ], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

Compiler

swap:

```
mul    $2, $5, 4
add    $2, $4, $2
lw     $15, 0($2)
lw     $16, 4($2)
sw     $16, 0($2)
sw     $15, 4($2)
jr     $31
```

Assembler

Assembly
language
Program
(for MIPS)

```
000000001010000100000000000011000
00000000100011100001100000100001
10001100011000100000000000000000
100011001111001000000000000000100
10101100111100100000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```

Binary
machine
language
program
(for MIPS)

Алгоритмы

- Алгоритм – это формально описанная вычислительная процедура, получающая исходные данные, и выдающая результат вычислений на выход (Кормен и др. "Алгоритмы: построение и анализ")

- Простейшие:
 - Сортировка вставками
 - Сортировка выбором
 - Сортировка пузырьком
- Чуть сложнее:
 - Сортировка слиянием
 - Быстрая сортировка
 - Цифровая сортировка

- Обычно изучают зависимость времени работы от размера входа
- Размер входа – зависит от конкретной задачи
- Для сортировки, размер входа – это количество элементов, которые нужно отсортировать
- Время работы – число элементарных шагов, которые выполняет алгоритм


```
for (int j = 0; j < length-1; j++)  
    for (int i = 0; i < length-1; i++)  
        if (a[i] > a[i+1])  
            swap(a[i], a[i+1]);
```

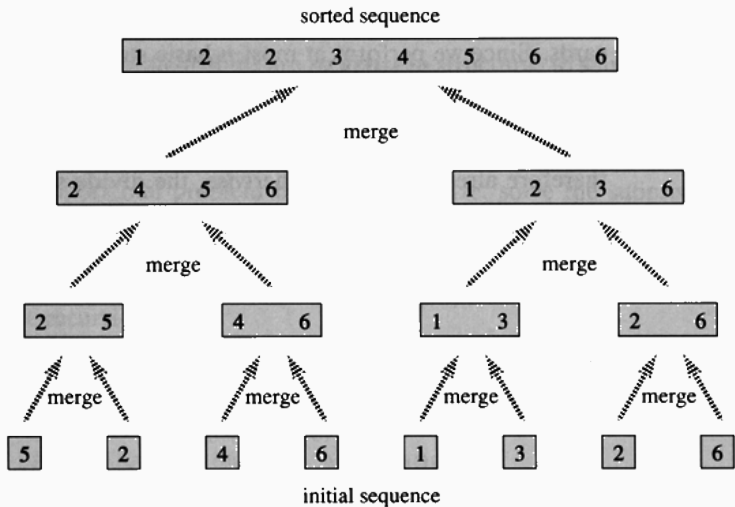
- Число операций, требуемых на один проход: $a * n$
- Число проходов: n
- Значит, время работы $\sim n^2$

Принцип "разделяй и
властвуй". Сортировка
слиянием и быстрая
сортировка

Принцип "разделяй и властвуй"

- Задача разбивается на несколько подзадач меньшего размера
- Эти задачи решаются (обычно с помощью рекурсивного вызова)
- Решения этих задач комбинируются и получается решение исходной задачи

Сортировка слиянием



```
void MergeSort(int * A, int p, int r)
{
    if (p < r)
    {
        int q = (p + r) / 2;
        MergeSort(A, p, q);
        MergeSort(A, q + 1, r);
        Merge(A, p, q, r);
    }
}
```

Быстрая сортировка (quicksort)

- Выбираем в массиве некоторый элемент, который будем называть опорным
- Переставляем элементы массива таким образом, чтобы все элементы со значением меньшим или равным опорному элементу, оказались слева от него, а все элементы, превышающие по значению опорный — справа от него
- Рекурсивно сортируем подмассивы, лежащие слева и справа от опорного элемента

Быстрая сортировка (quicksort)



Неотсортированный массив



Опорное значение = 7



$12 \geq 7 \geq 2$,
меняем местами 12 и 2



$26 \geq 7 \geq 7$
меняем 26 и 7

Быстрая сортировка (quicksort)



...



Время работы сортировок

- Время работы сортировки пузырьком, выбором и вставками $O(n^2)$
- Время работы сортировки слиянием и быстрой сортировки в среднем $O(n\log(n))$

- Пусть мы хотим отсортировать массив из 1 миллиарда чисел
- Сортировка пузырьком написана аккуратно и требует $2n^2$ операций и выполняется на суперкомпьютере состоящем из 1000 CPU
- Сортировка слиянием написана неэффективно и требует $50n\log(n)$ операций и выполняется на домашнем компьютере (1 CPU)
- Сортировка пузырьком на суперкомпьютере выполнится за 10 дней
- Сортировка слиянием на домашнем ПК выполнится за 10 минут

Стандартная сортировка

qsort()

Стандартная сортировка qsort()

```
#include <stdlib.h>
int values[] = { 88, 56, 100, 2, 25 };

int cmp(const void * a, const void * b)
{
    return ( *(int*)a - *(int*)b );
}
...
qsort(values, 5, sizeof(int), cmp);
```