Поток

Самостоятельная цепочка последовательно выполняемых операторов программы, соответствующих некоторой подзадаче. В адресном пространстве одного процесса может выполняться несколько потоков.

Все ***глобальные*** данные являются общими для различных потоков, выполняющихся в рамках одной программы

локальные данные индивидуальны

Глобальные и локальные данные

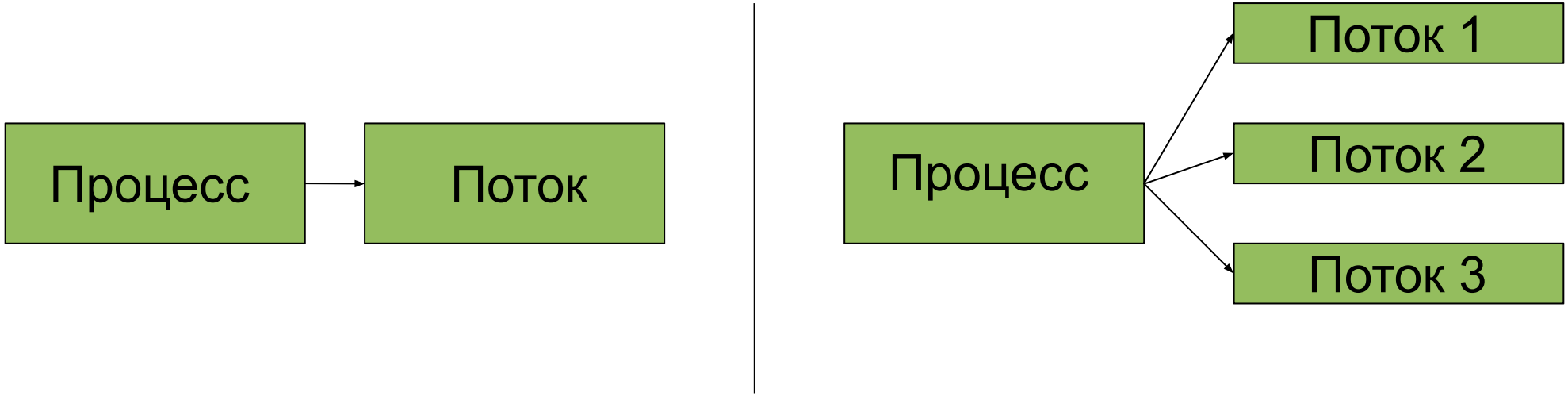
глобальные данные - это данные, расположенные в статической памяти и области динамической памяти, выделенные с помощью функций управления динамической памятью

локальные данные это автоматические переменные, т.е. переменные, объявленные внутри функций или составных блоков

```
1: int a;
 2: main()
 3: {
 4:    double b;
 5:    static int k;
 6:    char* p;
 7:    p=(char*)malloc(10);
 8: }
```

В данном примере переменные ***a*** и ***k***, а также область памяти, выделенная в строке 7, являются глобальными данными. Переменные ***b*** и ***p*** представляют собой пример локальных данных.

Прирост в производительности при использовании потоков



При создании двух потоков в рамках одного процесса для получения ускорения необходимо, чтобы оба потока выполнялись **разными процессорными устройствами**, что невозможно, если весь процесс рассматривается планировщиком как **единое целое.**

Функция создания потока

```
int pthread_create(pthread_t * iaddr,
pthread_attr_t * attr,
void *(*start_routine)(void*),
void *arg);
```

iaddr *- идентификатор создаваемого потока (выходной параметр);*

attr *- атрибуты создаваемого потока;*

start_routine *- адрес функции с единственным аргументом* ***arg****, которая будет выполняться создаваемым потоком.*

Создает поток с атрибутами `attr`*. Если в качестве* `attr` *передается* `NULL`*, то применяются атрибуты по умолчанию. Выполнение потока начинается с функции* `start_routine`*. Функция возвращает 0 в случае успешного завершения и код ошибки в противном случае*

Функция ожидания завершения потока

```
int pthread_join(pthread_t thread,
                 void **value_ptr)
```

Приостанавливает выполнение вызывающего потока до тех пор, пока поток thread не будет завершен.

thread **- *идентификатор потока;***

value_ptr **- *указатель на значение, возвращаемое функцией потока; функция возвращает 0 в случае успешного завершение и код ошибки в противном случае***

Функция завершения потока

```
void pthread_exit(void *value_ptr)
```

***value_ptr* - указатель на область памяти, содержащую возвращаемое значение;**

Пример использования функций работы с потоками

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

void* hello(void* arg) {
    printf("Hi from thread %s \n", (char*)arg);
    return NULL;
}

main(int argc, char* argv[]){
    int i, rc, *status;
    pthread_t *threads;
    if(argc > 1) {
        threads = (pthread_t*) malloc((argc - 1) * sizeof(pthread_t));
        for(i = 1; i < argc; i ++) {
            rc = pthread_create(&(threads[i-1]), NULL, hello, argv[i]);
            if(rc != 0) {
                fprintf(stderr, "Error creating thread: code %d\n", rc);
exit(-1);
            }
        }
        for(i = 1; i < argc; i ++) {
            rc = pthread_join(threads[i - 1], NULL);
            if(rc != 0) {
                fprintf(stderr, "Error joining thread: code %d\n" ,rc);
exit(-1);
            }
        }
    }
}
```

Компиляция, запуск и результаты программы

Пусть программа с предыдущего слайда называлась ***hw.c***

<u>Компиляция</u> происходит следующей командой:

```
cc -o hw hw.c -lpthread
```

Обратим внимание на флаг компиляции `-lpthread` ***который символизирует подключение библиотеки потоков.***

<u>Запуск:</u>

```
$./hw sun moon
```

где `sun` ***и*** `moon` ***агрументы программы***

<u>Вывод</u> зависит от аргументов, переданных с командной строки:

```
Hi from thread sun
Hi from thread moon
```

Компиляция, запуск и результаты программы