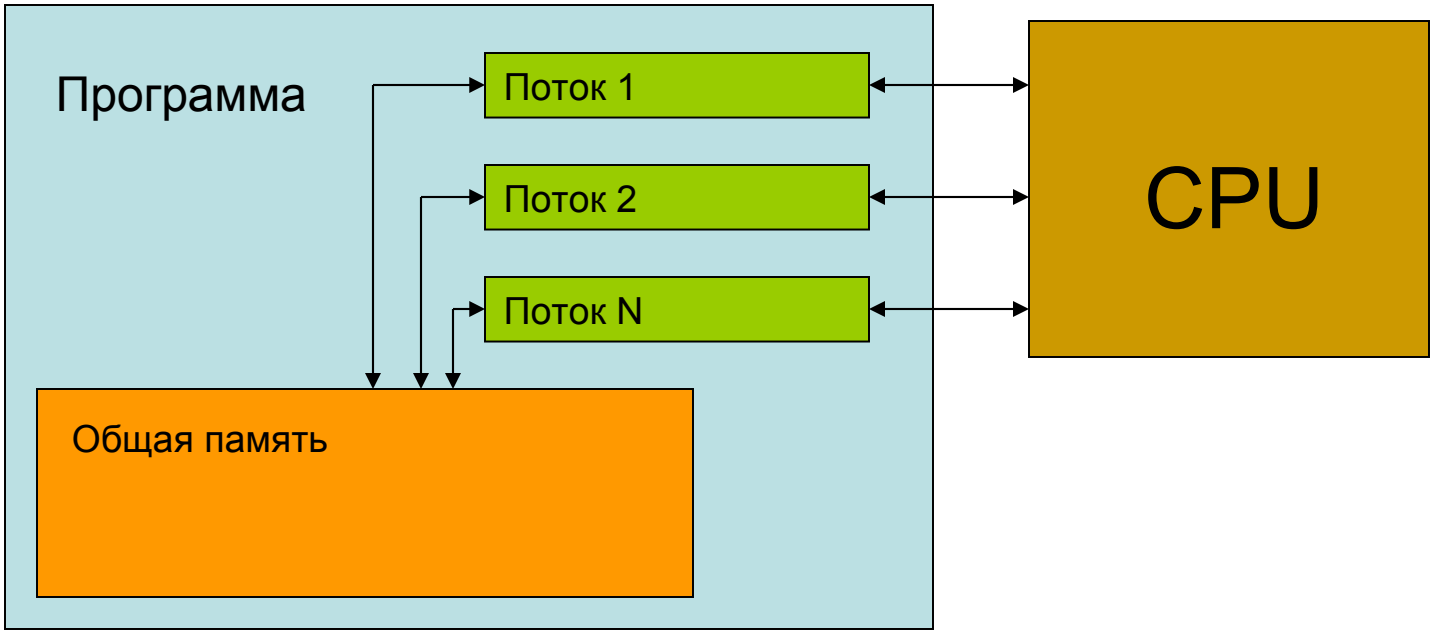
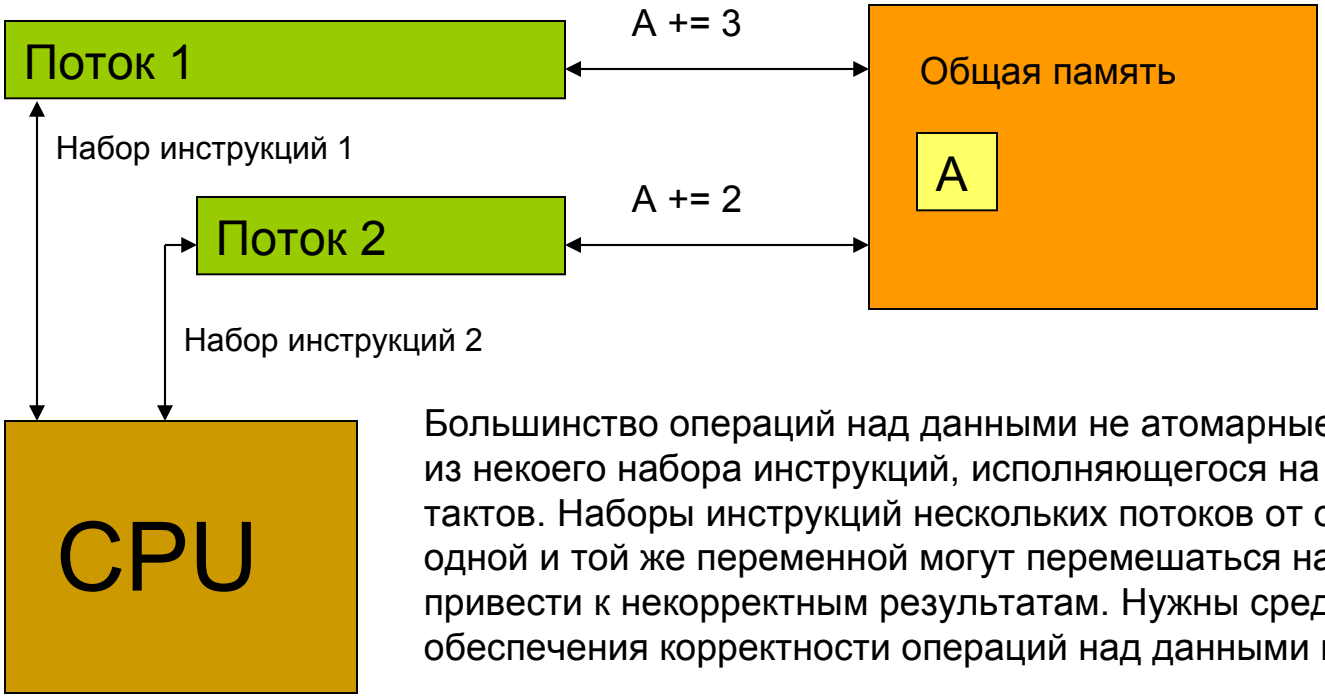
POSIX Threads

Общая модель



Потоки – наборы инструкций, исполняющиеся на CPU. Все потоки одной программы работают над одним (разделяемым) адресным пространством.

Для чего нужна синхронизация



Большинство операций над данными не атомарные – то есть состоят из некоего набора инструкций, исполняющегося на CPU за несколько тактов. Наборы инструкций нескольких потоков от операций над одной и той же переменной могут перемешаться на CPU, что может привести к некорректным результатам. Нужны средства для обеспечения корректности операций над данными в общей памяти.

A += 3:

```
mov ax, A
add ax, 3
mov A, ax
```

A += 2:

```
mov ax, A
add ax, 2
mov A, ax
```

→ Возможен некорректный результат (A = 2 или 3, но не 5)

Библиотека POSIX Threads

- #include <pthreads.h> - функции библиотеки
- компиляция: … –lpthreads

Создание потоков

int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void*(*start_routine)(void*), void* arg) - создание и запуск потока. В thread возвращается некий дескриптор потока для дальнейших операций с ним, attr – атрибуты потока (могут быть опущены – NULL), start_routine – указатель на функцию вида void* start_routine(void*), arg – аргумент для передачи в функцию start_routine. Исполнение потока заключается в исполнении функции start_routine.

int pthread_join(pthread_t thread, void** value) – дожидается окончания работы потока. thread – дескриптор потока, в value возвращается результат работы потока (может быть опущен – NULL).

Пример создания потоков:

```
void* func1(void* arg) {… return 0;} // функции потоков
void* func2(void* arg) {… return 0;}

int main()
{
pthread_ thread1[2], thread2; // дескрипторы потоков
int a[2] = {0,1};
pthread_create(&thread1[0], NULL, &func1, &a[0]); // создаем и запускаем потоки
pthread_create(&thread1[1], NULL, &func1, &a[1]);
pthread_create(&thread2, NULL, &func2, NULL);
… // можно еще что-то поделать
pthread_join(thread1[0], NULL); // дожидаемся завершения потоков
pthread_join(thread1[1], NULL);
pthread_join(thread2, NULL);
return 0;
}
```

Mutex

Mutex – некий объект в системе. Может быть в двух состояниях: “занят”/”незанят”.

pthread_mutex_t mutex; // mutex в POSIX Threads

int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr) – инициализация mutexa. Параметр attr – атрибуты mutexa, в общем случае может быть опущен (равен NULL)

int pthread_mutex_destroy(pthread_mutex_t *mutex) – удаление mutexa

Пример:

```
pthread_mutex_t mutex;

pthread_mutex_init(&mutex, NULL); // инициализация mutexa

…// используем mutex

pthread_mutex_destroy(&mutex); // удаляем ненужный mutex
```

Использование mutex'a

int pthread_mutex_lock(pthread_mutex_t *mutex) – занять мутекс. Поток, занявший свободный мутекс, исполняется дальше. Поток, попытавшийся занять уже занятый мутекс, приостанавливается до его освобождения.

int pthread_mutex_unlock(pthread_mutex_t *mutex) - освободить мутекс.

В каждый момент времени мутексом может владеть только один поток. Порядок, в котором потоки получают власть над мутексом, в общем случае не детерминирован. Операционной системой гарантируется, что все потоки, ожидающие мутекс, когда либо его получат.

Пример ограничения доступа к разделяемой переменной с помощью mutex'a:

```
// Для каждого потока:

pthread_mutex_lock(&mutex); // занимаем мутекс

A += 2;   // в каждый момент времени операцию над A будет исполнять только один поток,
          // поэтому итоговый результат будет корректным

pthread_mutex_unlock(&mutex); // освобождаем мутекс
```

Условные переменные

Условная переменная – некий объект в системе, который может использоваться для оповещения потоков о наступлении некоего условия.

pthread cond_t cond; // условная переменная в POSIX Threads

int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr) – инициализация условной переменной. Параметр attr – атрибуты переменной, в общем случае может быть опущен (равен NULL)

int pthread_cond_destroy(pthread_cond_t *cond) – удаление условной переменной

Пример:

```
pthread_cond_t cond;

pthread_cond_init(&cond, NULL); // инициализация переменной

…// используем переменную

pthread_cond_destroy(&cond); // удаляем ненужную переменную
```

Работа с условными переменными

int pthread_cond_wait(pthread_cond_t* cond, pthread_mutex_t *mutex) – приостанавливает поток до наступления события cond. Мутекс mutex освобождается на время ожидания и снова занимается при выходе из функции.

int pthread_cond_signal(pthread_cond_t* cond) – сигнализирует о наступлении события cond. При этом только один из потоков, ожидающих событие (если есть такой), выйдет из ожидания.

int pthread_cond_broadcast(pthread_cond_t* cond) - сигнализирует о наступлении события cond. При этом все потоки, ожидающие событие (если есть такие), выходят из ожидания.

Пример использования условных переменных:

```
pthread_cond_t cond, pthread_mutex_t mutex;

// Поток 1:
pthread_mutex_lock(&mutex); // занимаем мутекс
if(!condition) // condition зависит от переменных, защищаемых мутексом
            pthread_cond_wait(&cond, &mutex); // ждем наступления события (condition = true)
pthread_mutex_unlock(&mutex); // освобождаем вновь занятый мутекс

// Поток 2:
pthread_mutex_lock(&mutex); // занимаем мутекс
… // что-то делаем – чтобы condition = true
pthread_cond_signal(&cond); // посылаем сигнал о событии – Поток 1 должен проснуться (если ждет события)
pthread_mutex_unlock(&mutex); // освобождаем мутекс
```