САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Параллельные вычисления

Проектирование многопоточных программ

Михаил Моисеев

Санкт-Петербург
2012

Шаблоны проектирования параллельных программ

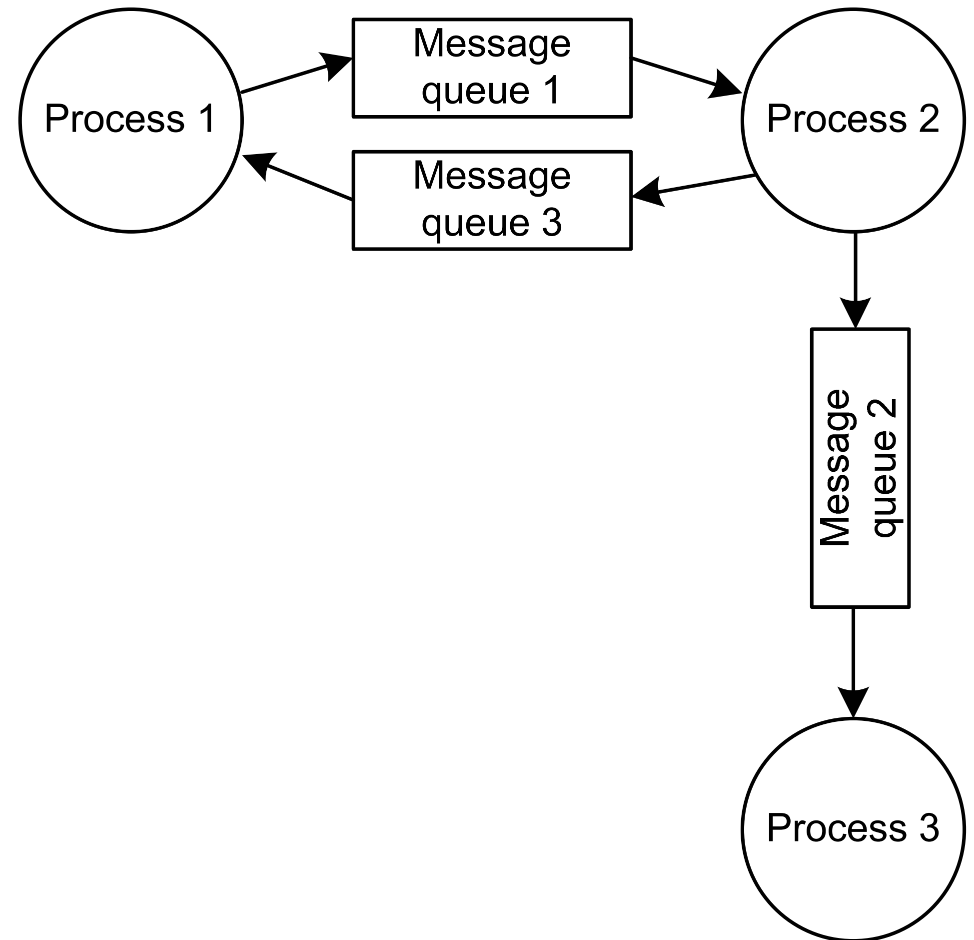
- Шаблон проектирования определяет общие правила создания параллельных программ
  - Модель параллельных вычислений – формализм описывающий параллельную программу
  - Используемые стандарты и библиотеки функций/компонентов

Модели параллельных вычислений

- Модель параллельных вычислений
  - Примитивы параллелизма: потоки, процессы, …
  - Объекты и правила синхронизации
  - Управление выполнением, переключение контекста
- Примеры моделей
  - Процессы, соединенные каналами связи
  - Процессы, обменивающиеся сообщениями
  - Процессы/потоки с общей памятью
  - …

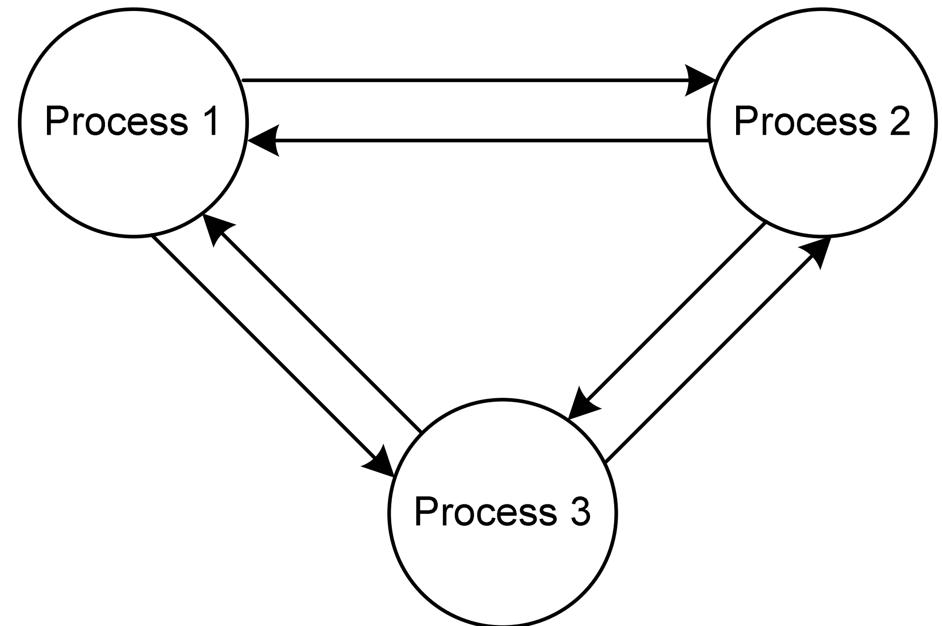
Процессы, соединенные каналами связи

- Несколько параллельно выполняющихся процессов, динамическое создание процессов
- Локальные данные процессов, общих данных нет
- Каналы – однонаправленные очереди сообщений, связывающие пары портов разных процессов, динамическое создание каналов
- Между двумя процессами может быть произвольное число каналов



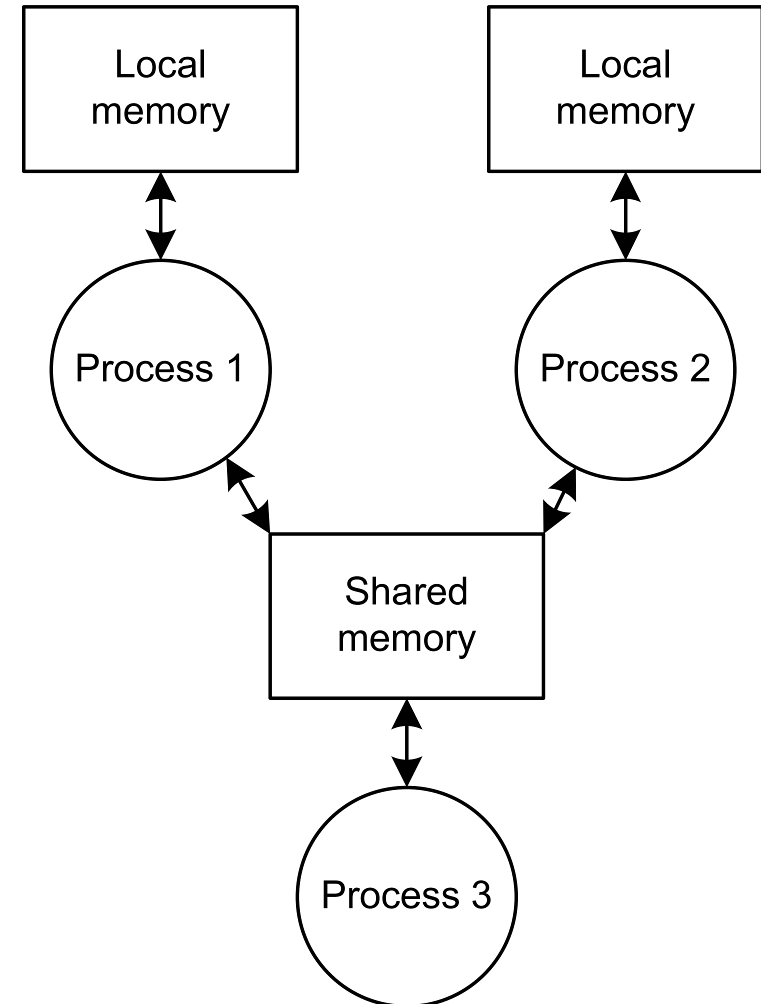
Процессы, обменивающиеся сообщениями

- Несколько параллельно выполняющихся процессов, динамическое создание процессов
- Локальные данные процессов, общих данных нет
- Передача сообщений из процесса в процесс – адресуется конкретный процесс, а не канал



Процессы/потоки с общей памятью

- Несколько параллельно выполняющихся процессов/потоков, динамическое создание процессов
- Локальные и общие данные процессов
- Использование механизмов синхронизации при работе с общей памятью



Классификация шаблонов проектирования ПП

- По уровню параллелизма
  - Уровень задач(подзадач)/команд/данных
- По используемым примитивам параллелизма
  - Процессы/потоки/пользовательские потоки
- По используемым процедурам синхронизации
- По способу управления процессами
  - Вытесняющая/невытесняющая многозадачность
- По организации доступа к данным
  - Локальные/общие данные
- По применимости для различных платформ

Проектирование многопоточных программ

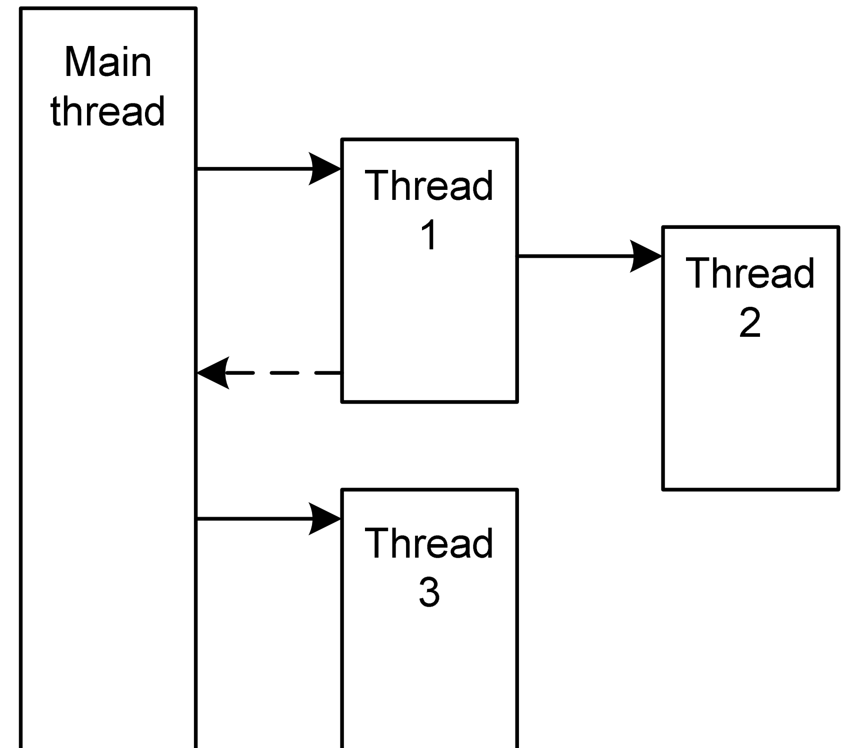
- Модель взаимодействующих потоков с общей памятью
  - Динамическое создание потоков
  - Использование различных объектов синхронизации
- Стандарты и библиотеки
  - POSIX Threads
  - OpenMP
  - Windows Threads
  - Boost Threads
  - Intel TBB

Создание многопоточных программ на основе POSIX Threads

- POSIX Threads – стандарт IEEE 1003.1, ISO/IEC 9945 (http://pubs.opengroup.org/onlinepubs/009695399/mindex.html)
- Pthreads определяет интерфейсы для языков C/C++
- Имеются реализации для ОС Linux, FreeBSD, Solaris, QNX, MS Windows

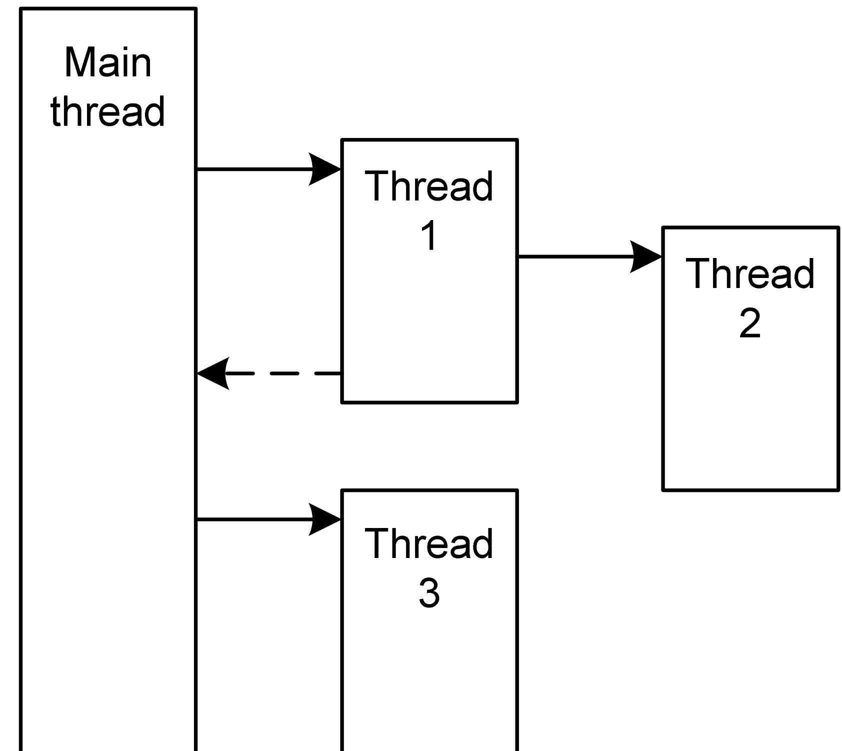
Модель параллельных вычислений PThreads

- Поток программы является основным объектом для организации параллельного выполнения функций в многопоточной программе
- При запуске программы начинает выполняться один главный поток – остальные потоки создаются динамически
- В созданном потоке запускается на выполнение функция программы
- Потоки могут создаваться в любых потоках



Модель параллельных вычислений PThreads

- Завершение потоков
  - завершение функции потока
  - завершение программы
  - вызовы специальных функций из текущего или других потоков
- Два типа потоков
  - detached-поток
  - обычный (не-detached) поток
- Ожидание завершения дочернего потока не-detached в родительском потоке



Модель памяти и синхронизация в PThreads

- Потоки программы могут обращаться ко всем видимым объектам программы
  - Локальные переменные потока – определяются только правилами языка программирования
  - Разделяемые переменные
- Для разграничения доступа к разделяемым объектам, а также для обеспечения определенных последовательностей выполнения конструкций в потоках используются объекты синхронизации

Потоки и объекты синхронизации PThreads

- pthread_t – поток программы
- pthread_mutex_t – объект синхронизации типа мьютекс
- sem_t – объект синхронизации типа семафор
- pthread_rwlock_t – объект синхронизации типа RWLock
- pthread_spinlock_t – объект синхронизации типа Spin
- pthread_cond_t – переменная состояния
- pthread_barrier_t – барьер
- pthread_key_t – ключ идентификации данных отдельных потоков
- pthread_once_t – переменная однократного выполнения

Функции управления потоками программы

- `int` **`pthread_create`** `(pthread_t * thread, const pthread_attr_t * attr, void * (*start_routine) (void *), void* arg)` – создание потока и запуск в нем указанной функции с указанным аргументом
- `int` **`pthread_join`** `(pthread_t thread, void ** value)` – ожидание завершения обычного (не-detached) потока
- `int` **`pthread_attr_init`**`(pthread_attr_t *attr)`
  - `Detach state`
  - `Stack size`
  - `...`
- `int` **`pthread_attr_destroy`**`(pthread_attr_t *attr)`

Функции управления потоками программы

- `int` **`pthread_detach`** `(pthread_t thread)` – перевод потока в состояние detached
- `void` **`pthread_exit`** `(void * retval)` – завершение выполнения текущего потока
- `int` **`pthread_cancel`** `(pthread_t thread)` – прекращение выполнения указанного потока
  - точки завершения потока PTHREAD_CANCEL_DEFERRED и PTHREAD_CANCEL_ASYNCHRONOUS
- `int` **`pthread_kill`** `(pthread_t thread, int sig)` – отправка сигнала указанному потоку программы

Пример работы с потоком программы 1

```
#include <pthread.h>
int a = 0;
void* f(void* b){
    a = a + 1;
}

int main() {
    pthread_t t;
    pthread_create(&t, NULL, f, NULL);
    pthread_join(t, NULL);
    printf("%d", a);
}
```

Пример работы с потоком программы 2

```
int a = 0;
void* f(void* b){
    int i = *(int*) b;
    a = a + i;
}
int main() {
    pthread_t t;
    int c = 1;
    pthread_create(&t, NULL, f, &c);
    pthread_join(t, NULL);
    printf("%d", a);
}
```

Пример работы с потоком программы 3

```
int a = 0;
void* f(void* b){
    a++;
    pthread_exit(&a);
}
int main() {
    pthread_t t;
    void* c;
    pthread_create(&t, NULL, f, NULL);
    pthread_join(t, &c);
    int* i = *(int*) c;
    printf("%d", *i);
}
```

Пример работы с потоком программы 4

```
int a = 0;
void* f(void* b){
    a++;
    if (a < 10) {
        pthread_t t;
        pthread_create(&t, NULL, f, NULL);
        pthread_detach(t);
    }
}
int main() {
    pthread_t t;
    pthread_create(&t, NULL, f, NULL);
    pthread_detach(t);
}
```

Функции для работы с мьютексом

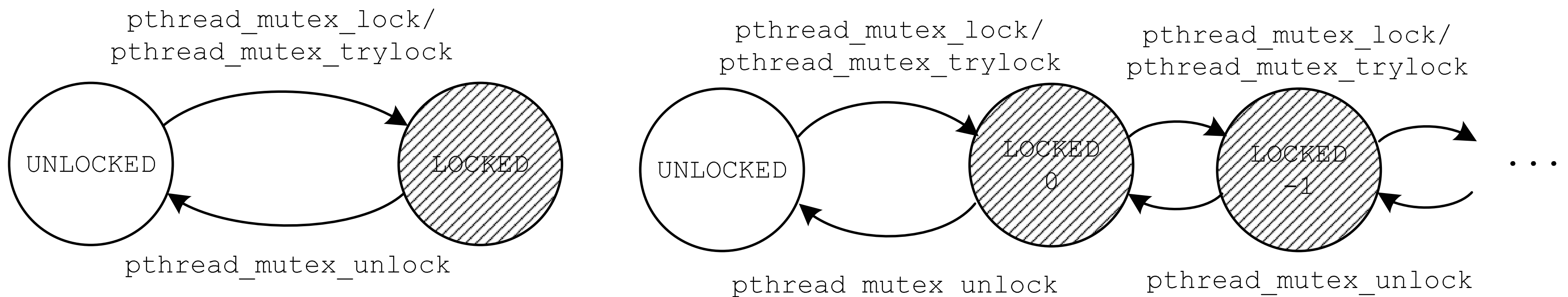
- `int` **`pthread_mutex_init`** `(pthread_mutex_t * mutex, const pthread_mutexattr_t *attr)` – инициализация мьютекса
- `int` **`pthread_mutex_destroy`** `(pthread_mutex_t *)` – уничтожение мьютекса
- **`int pthread_mutexattr_init( pthread_mutexattr_t *attr )`**
  - Mutex type (PTHREAD_MUTEX_NORMAL, PTHREAD_MUTEX_ERRCHECK, PTHREAD_MUTEX_RECURSIVE)
  - Shared flag

Функции для работы с мьютексом

- `int` **`pthread_mutex_lock`** `(pthread_mutex_t *)` – блокировка мьютекса;
- `int` **`pthread_mutex_trylock`** `(pthread_mutex_t *)` – попытка блокировки мьютекса
- `int` **`pthread_mutex_unlock`** `(pthread_mutex_t *)` – освобождение мьютекса
- Функции захвата и освобождения мьютекса должны выполняться строго из одного потока, освобождение не захваченного мьютекса является ошибкой

Типы мьютексов

- Тип мьютекса определяет атрибутом инициализации
  - PTHREAD_MUTEX_NORMAL – обычный мьютекс, допускает операции захвата и освобождения объекта в одном и том же потоке
  - PTHREAD_MUTEX_ERRORCHECK – мьютексы этого типа не допускают повторный захват, освобождение незахваченного объекта или объекта захваченного в другом потоке, в таких случаях выдается ошибка
  - PTHREAD_MUTEX_RECURSIVE – рекурсивный мьютекс допускает многократный захват объекта в одном потоке



Пример программы с мьютексом 1

```
pthread_mutex_t mutex;
int x; // Shared variable

int main(){

    pthread_mutex_init(&mutex, NULL);
    ...
    pthread_mutex_lock(&mutex);
    doSomething(x);
    pthread_mutex_unlock(&mutex);
    ...
    pthread_mutex_destroy(&mutex);

}
```

Пример программы с мьютексом 2

```
int a = 0;
pthread_mutex_t mutex;
void* f(void* b) {
    pthread_mutex_lock(&mutex);
    a = a + 1;
    pthread_mutex_unlock(&mutex);
}
int main(){
    pthread_create(&t, NULL, f, NULL);
    pthread_mutex_lock(&mutex);
    int b = a;
    pthread_mutex_unlock(&mutex);
    pthread_join(t, NULL);
    printf("%d %d", a, b);  // 1,0 или 1,1
}
```

Пример программы с мьютексом 3

```
int a = 0;
pthread_mutex_t mutex;
void* f(void* b) {
    pthread_mutex_lock(&mutex);
    a = a + 1;
    pthread_mutex_unlock(&mutex);
}
int main(){
    pthread_mutex_lock(&mutex);
    pthread_create(&t, NULL, f, NULL);
    int b = a;
    pthread_mutex_unlock(&mutex);
    pthread_join(t, NULL);
    printf("%d %d", a, b);  // ?
}
```

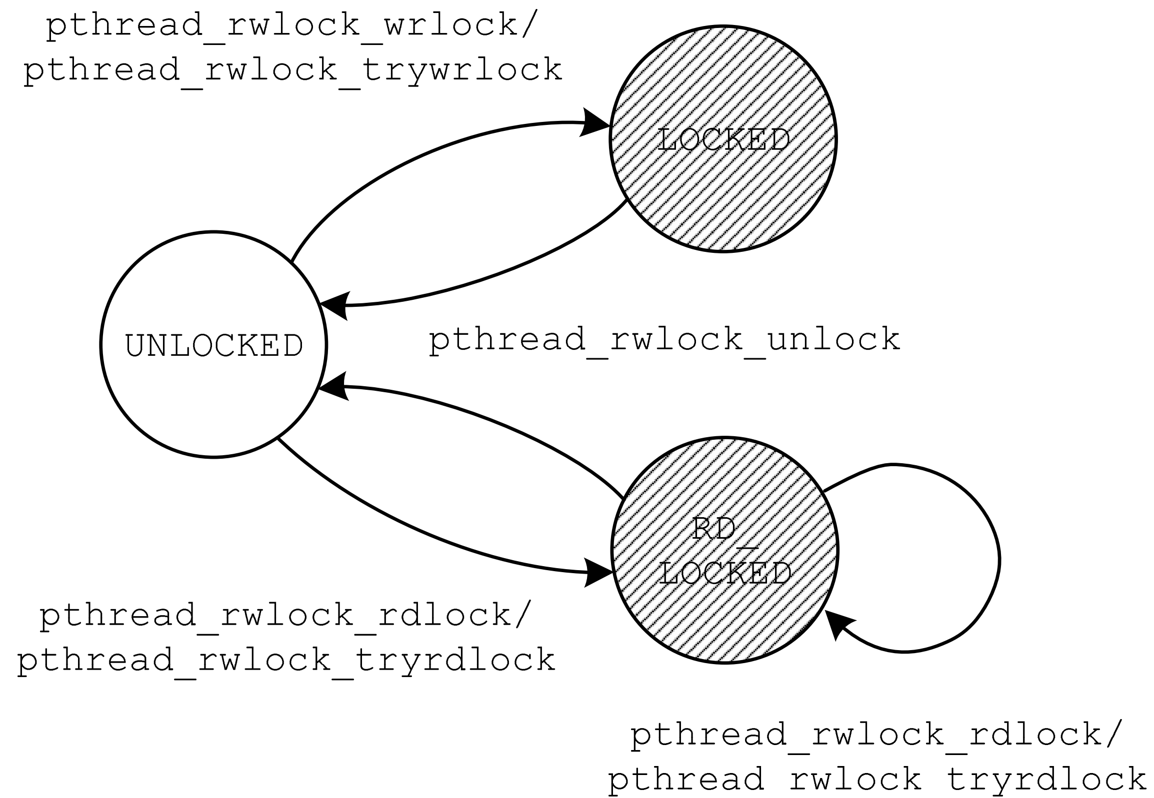
Пример программы с мьютексом 4

```
int a = 0;
pthread_mutex_t mutex;
void* f(void* b) {
    if (!pthread_mutex_trylock(&mutex)){
        a = a + 1;
        pthread_mutex_unlock(&mutex);
    }
}
int main(){
    pthread_create(&t, NULL, f, NULL);
    pthread_mutex_lock(&mutex);
    int b = a;
    pthread_mutex_unlock(&mutex);
    printf("%d %d", a, b);  // ?
}
```

Функции для работы с RWLock

- `int` **`pthread_rwlock_init`** `(pthread_rwlock_t *,const pthread_rwlockattr_t *)` – инициализация объекта синхронизации
- `int` **`pthread_rwlock_destroy`** `(pthread_rwlock_t *)` – уничтожение объекта синхронизации;
- `int` **`pthread_rwlock_rdlock`** `(pthread_rwlock_t *)` – блокировка потока для выполнения операции чтения;
- `int` **`pthread_rwlock_wrlock`** `(pthread_rwlock_t *)` – блокировка потока для выполнения операции записи;
- `int` **`pthread_rwlock_tryrdlock`** `(pthread_rwlock_t *)` – попытка блокировки потока для выполнения операции чтения;
- `int` **`pthread_rwlock_trywrlock`** `(pthread_rwlock_t *)` – попытка блокировки потока для выполнения операции записи;
- `int` **`pthread_rwlock_unlock`** `(pthread_rwlock_t *)` – освобождение объекта синхронизации

Состояния объекта типа RWLock



Пример программы с RWLock

```
int a = 0;
pthread_rwlock_t lock;
void* reader(void* b) {
    pthread_rwlock_rdlock(&lock)){
    int i = a + 1;
    pthread_rwlock_unlock(&lock);
}
void* writer(void* b) {
    pthread_rwlock_wrlock(&lock)){
    a = a + 1;
    pthread_rwlock_unlock(&lock);
}
void main() {
    ...
    pthread_create(&t, NULL, reader, NULL);
    pthread_create(&t, NULL, writer, NULL);
}
```

Функции для работы с семафором

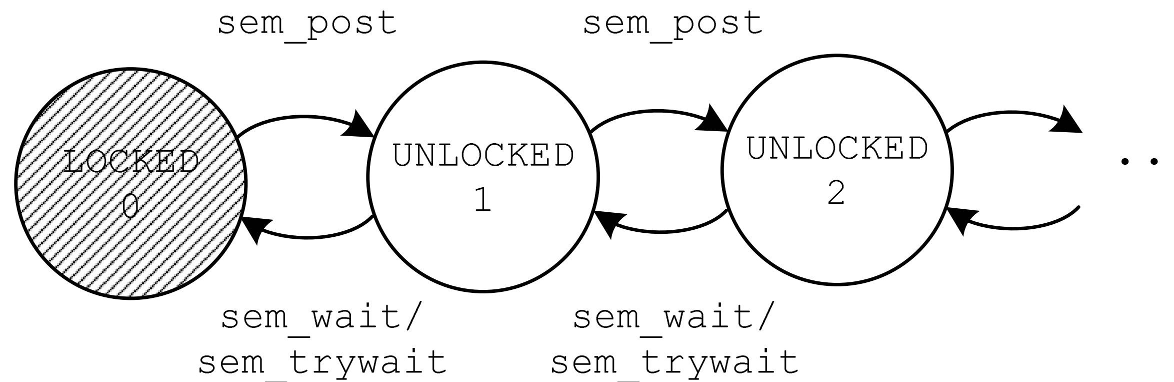
- `int` **`sem_init`** `(sem_t * sem, int shared, unsigned value)` – инициализация семафора, устанавливается начальное значение семафора
- `int` **`sem_destroy`** `(sem_t *)` – уничтожение семафора

Функции для работы с семафором

- `int` **`sem_post`** `(sem_t *)` – увеличение значения семафора. В случае если значение семафора равно нулю и имеются потоки, ожидающие открытия семафора на конструкции sem_wait, один из таких потоков разблокируется и продолжает свое выполнение. В случае если семафор имеет положительное значение или нет ожидающих потоков, то значение семафора просто увеличивается на единицу
- `int` **`sem_wait`**`(sem_t *)` – уменьшение значения семафора. В случае если значение семафора равно нулю вызывающий поток блокируется до тех пор, пока не будет разблокирован вызовом sem_post. В случае если семафор имеет положительное значение, это значение уменьшается на единицу
- `int` **`sem_trywait`**`(sem_t *)` – попытка уменьшения значения семафора
- `int` **`sem_getvalue`**`(sem_t * sem, int * sval)` – получение текущего состояния семафора

Состояния семафора

- Функции для семафора могут вызываться из произвольных потоков



Пример программы с семафором 1

```
#include <semaphore.h>
sem_t sem;

int main(){
  sem_init(&sem, 0, 0);
  pthread_create(&t, NULL, f, NULL);
  ...
  sem_wait(&sem);
  ...
  sem_post(&sem);
  ...
  sem_destroy(&sem);
}
```

Пример программы с семафором 2

```
int a = 0;
sem_t sem;
void* f(void* b) {
   a = a + 1;
   sem_post(&sem);
}
int main(){
   sem_init(&sem, 0, 0);
   pthread_create(&t, NULL, f, NULL);
   sem_wait(&sem);
   int b = a;
   pthread_join(t, NULL);
   printf("%d %d", a, b);  // ?
}
```

Пример программы с семафором 3

```
int a = 0;
sem_t sem;
void* f(void* b) {
    a = a + 1;
    sem_post(&sem);
}
int main() {
    sem_init(&sem, 0, 0);
    pthread_create(&t, NULL, f, NULL);
    if (!sem_trywait(&sem)) {
        a = a + 1;
    }
    printf("%d", a); // ?
}
```

Пример программы с семафором 4

```
int a = 0;
sem_t sem;

1. if (!sem_trywait(&sem)) {
        a = a + 1;
   }

2. if (sem_getvalue(&sem) > 0) {
        sem_wait(&sem)
        a = a + 1;
   }
```

Функции для работы с condition variables

- `int` **`pthread_cond_init`** `(pthread_cond_t * cond, const pthread_condattr_t * attr)` – инициализация переменной условия
- `int` **`pthread_cond_destroy`** `(pthread_cond_t *)` – уничтожение переменной условия
- `int` **`pthread_cond_wait`** `(pthread_cond_t* cond, pthread_mutex_t* mutex)` – атомарное освобождение мьютекса и переход к ожиданию события изменения состояния переменной условия, блокировка мьютекса и завершение
- `int` **`pthread_cond_signal`** `(pthread_cond_t *cond)` – генерация события изменения состояния переменной условия для одного потока, ожидающего на вызове функции pthread_cond_wait
- `int` **`pthread_cond_broadcast`** `(pthread_cond_t *cond)` – генерация события изменения состояния переменной условия для всех потоков, ожидающих на вызове функции pthread_cond_wait

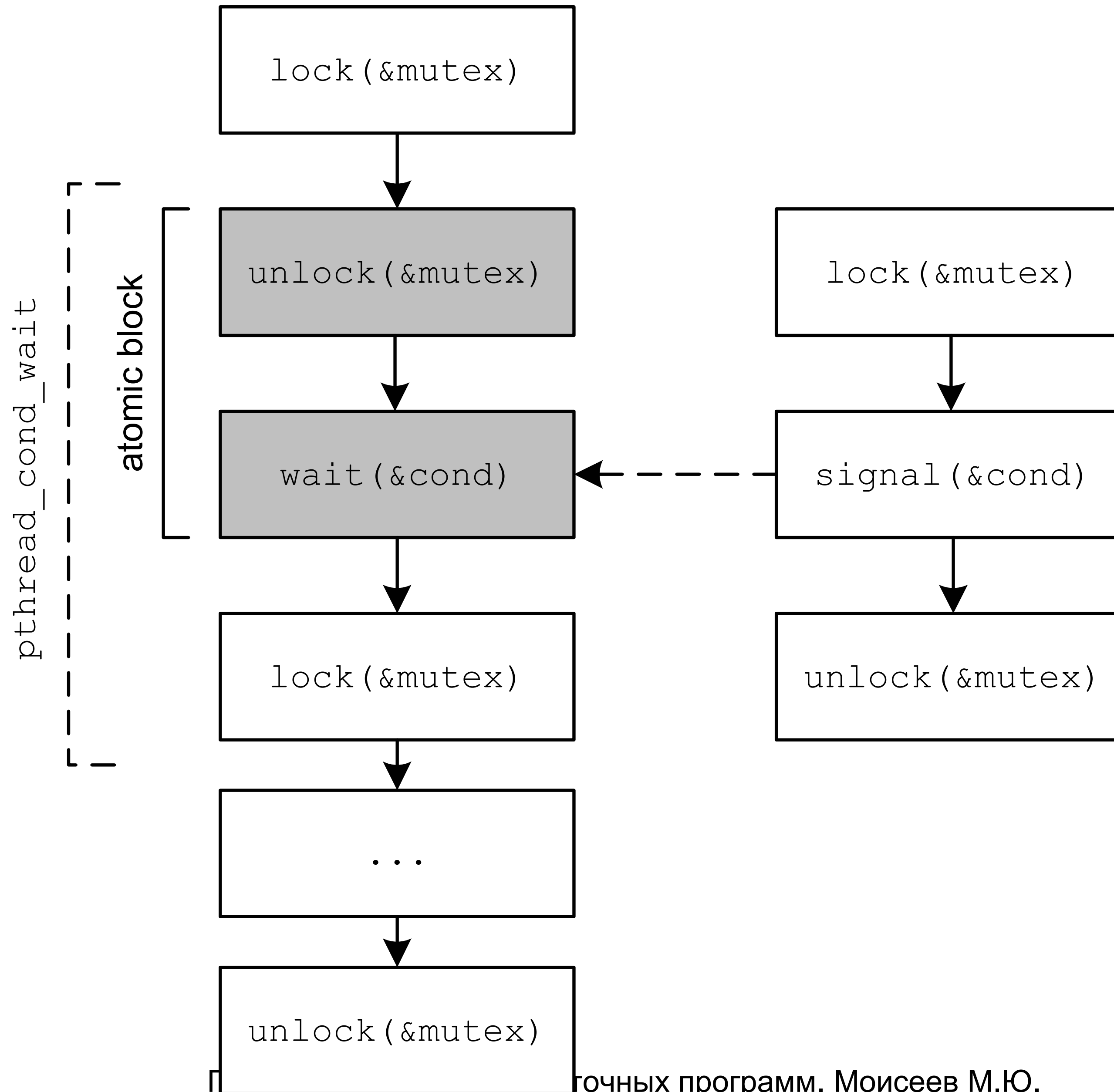
Использование condition variables

- Используются два объекта синхронизации
  - мьютекс – обеспечивает атомарность обработки сигналов
  - переменная условия – реагирует на полученный сигнал
- Вызов функции `pthread_cond_wait` обязательно выполняется под защитой мьютекса
- Вызов функций `pthread_cond_signal` и `pthread_cond_broadcast` также обычно выполняется под защитой того же мьютекса

```
pthread_mutex_lock(&mutex);
pthread_cond_wait(&cond, &mutex);
... //
pthread_mutex_unlock(&mutex);
```

```
pthread_mutex_lock(&mutex);
pthread_cond_signal(&cond);
pthread_mutex_unlock(&mutex);
```

Использование condition variables



Пример программы с condition variable 1

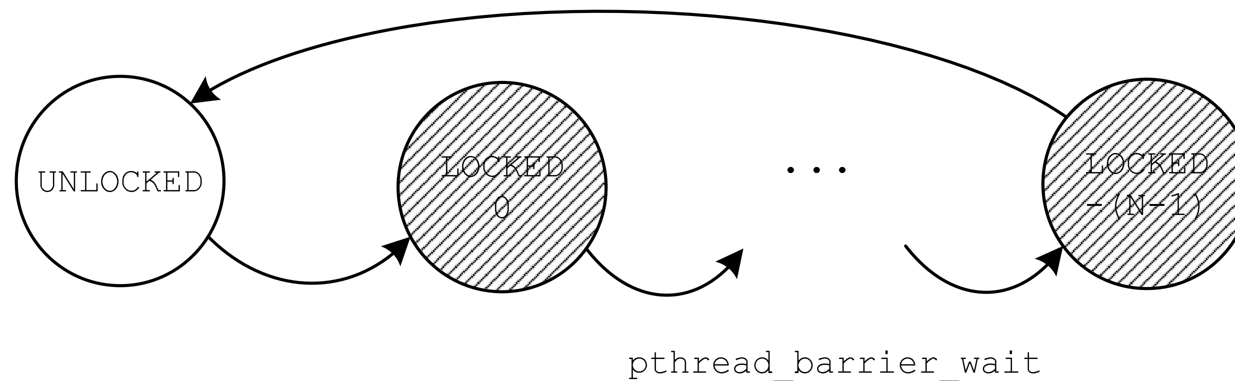
```
int a = 0;
pthread_cond_t cond;
pthread_mutex_t mutex;

int f() {
   pthread_mutex_lock(&mutex);
   a++;
   if (a < SOME_VALUE) {  // Проверка некоторого условия
      pthread_cond_wait(&cond, &mutex);
   } else {
      pthread_cond_broadcast(&cond); // Разбудить всех
   }
   pthread_mutex_unlock(&mutex);
}
```

Пример программы с condition variable 2

```
int a = 0;
int producer() {
    pthread_mutex_lock(&mutex);
    a++;
    if (a => 2) {
        pthread_cond_signal(&cond);
    }
    pthread_mutex_unlock(&mutex);
}
void consumer() {
    pthread_mutex_lock(&mutex);
    pthread_cond_wait(&cond, &mutex);
    a = a - 2;
    pthread_mutex_unlock(&mutex);
}
```

Функции для работы с барьером

- `int` **`pthread_barrier_init`**`(pthread_barrier_t * bar, pthread_barrierattr_t * attr, unsigned value)` – инициализация барьера
- `int` **`pthread_barrier_destroy`**`(pthread_barrier_t *)` – уничтожение барьера
- `int` **`pthread_barrier_wait`**`(pthread_barrier_t *)` – ожидание открытия барьера, барьер открывается когда заданное число потоков вызовет функцию pthread_barrier_wait



Пример программы с барьером

```
int a = 0;
pthread_barrier_t bar;
void f* f(void* b) {
   a++;
   pthread_barrier_wait(&bar);
   printf("%d", a); // ?
}
void main() {
   pthread_barrier_init(&bar, NULL, 3);
   pthread_create(&t1, NULL, f, NULL);
   pthread_create(&t2, NULL, f, NULL);
   pthread_create(&t3, NULL, f, NULL);
   ...
}
```

Функции для работы с переменными однократного выполнения

- `pthread_once_t once = PTHREAD_ONCE_INIT` – инициализация
- `int` **`pthread_once`** `(pthread_once_t *, void (*init_routine)(void))` – производит вызов функции в текущем потоке программы, если это первый вызов функции с данным объектом pthread_once_t, в противном случае вызов функции игнорируется
- Используется для динамической инициализации подключаемых библиотек

Пример программы с переменными однократного выполнения

```
pthread_once_t once = PTHREAD_ONCE_INIT;
void init_library() {
    ...;
}

int function()
{
    pthread_once(&once, init_library);
    ...
}
```

Функции для работы с ключами данных

- `int` **`pthread_key_create`**`(pthread_key_t* key, void(*destructor)(void*))` – создание ключа, деструктор для данных ключа является опциональным
- `int` **`pthread_key_delete`**`(pthread_key_t key)` – удаление ключа
- `int` **`pthread_setspecific`**`(pthread_key_t key, const void * data)` – связывание ключа с локальными данными текущего потока
- `void*` **`pthread_getspecific`**`(pthread_key_t key)` – получение данных текущего потока по ключу

Пример программы с ключами данных

```
pthread_key_t key;
pthread_once_t key_once = PTHREAD_ONCE_INIT;
void make_key() {
    pthread_key_create(&key, NULL);
}
func() {
    pthread_once(&key_once, make_key);
    if ((ptr = pthread_getspecific(key)) == NULL) {
        ptr = malloc(OBJECT_SIZE);
        ...
        pthread_setspecific(key, ptr);
    }
    ...
}
```