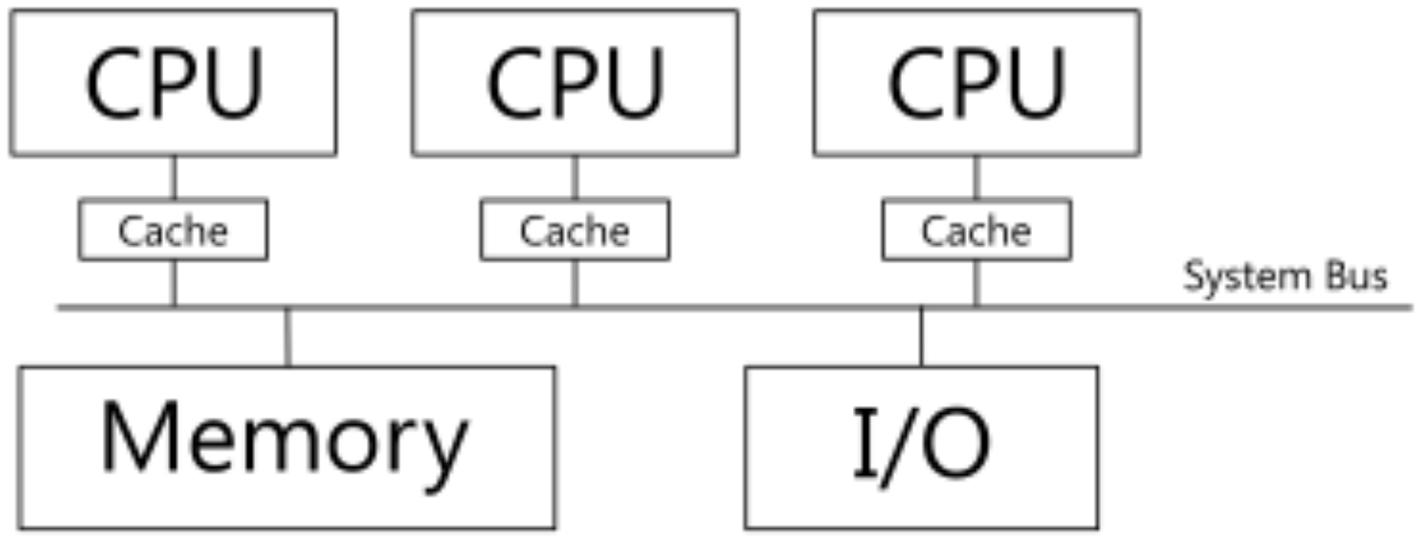
POSIX Threads

Многопоточные программы ориентированы на выполнение на системах с общей памятью (SMP - Symmetric multiprocessing).



POSIX Threads

- POSIX: Portable Operating Systems Interface for uniX

(Переносимый интерфейс операцииных систем, необходим для совместимости UNIX-подобных систем)

- Стандартный API для работы с потоками в UNIX-подобных системах с 1995 г. (IEEE/ANSI 1003.1c-1995)

- Низкоуровневый интерфейс для многопоточного программирования в среде C/UNIX

- Основа для других высокоуровных моделей (C++11-threads, OpenMP, Java threads, etc)

POSIX Threads

- Модель fork-join: потоки могут порождать другие потоки и ждать их завершения.

- Выполнение потоков планируется ОС, они могут выполняться последовательно или параллельно, могут назначаться на произвольные ядра.

- Потоки работают асинхронно.

- Для координации доступа к общим ресурсам должны использоваться механизмы (мьютексы, условные переменные и др.) взаимного исключения.

POSIX Threads

- Набор рекомендаций POSIX Threads состоит из около 100 функции.
- Названия функций начинаются с pthread_.
- Функции можно разделить на 4 группы: Управление (create, join ..), Мьютексы, Условные переменные, Синхронизации.
- Для использования необходимо подключить заголовочный файл <pthread.h>, указать -lpthread во время сборки.

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define NUM_THREADS 4

typedef struct _thread_data_t {
  int tid;
  double stuff;
} thread_data_t;

void* thr_func(void* arg) {
  thread_data_t *data = (thread_data_t *)arg;
  printf("hello from thr_func, thread id: %d\n", data->tid);
  pthread_exit(NULL);
}

int main(int argc, char **argv) {
  pthread_t thr[NUM_THREADS]; int i, rc;

  thread_data_t thr_data[NUM_THREADS];

  for(i = 0; i< NUM_THREADS; ++i) {
    thr_data[i].tid = i;
    if((rc = pthread_create(&thr[i], NULL, thr_func, &thr_data[i]))) {
      fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
      return EXIT_FAILURE;
    }
  }

  for(i = 0; i< NUM_THREADS; ++i) {
    pthread_join(thr[i], NULL);
  }
  return EXIT_SUCCESS;
}
```

POSIX Threads

Процедура `int pthread_create(*ptherad_t, const pthread_attr_t *attr, void* (*start_routine) (void*), void *arg)`

Возвращает 0 если в случае успешного вызова, отличное от 0 значение сигнализирует об ошибке.

Некоторые коды ошибок:

EAGAIN – у системы нет ресурсов для создания нового потока, или система не может больше создавать потоков, так как количество потоков превысило значение PTHREAD_THREADS_MAX

EINVAL – неправильные атрибуты потока (переданные аргументом attr)

EPERM – Вызывающий поток не имеет должных прав для того, чтобы задать нужные параметры или политики планировщика.

POSIX Threads

Процедура `int pthread_create(pthread_t * thread, const pthread_attr_t *attr, void *(*start_routine) (void *) start_routine, void *arg);`

`thread` - идентификатор потока (для завершения, синхронизации и т.д.)

`attr` - параметры потока (например, минимальный размер стека)

`start_routine` - функция для запуска (принимает и возвращает указатель void*)

`arg` - аргументы функции

POSIX Threads

`thread` - В результате успешного срабатывания функции по указанному адресу будет размещен описатель порожденного потока.

`attr` - Атрибуты потока. Задают свойства потока.

`start_routine` - Указатель на функцию потока. Выполнение потока состоит в выполнении этой функции.

`arg` - Указатель, который будет передан функции потока в качестве параметра. Функция потока может менять содержимое памяти с использованием этого указателя. pthread_create содержимого не меняет, просто передает указатель функции потока. Функция потока сама интерпретирует содержание памяти по этому адресу.

POSIX Threads

Атрибуты.

Атрибуты задают свойства потока. Может быть NULL. Объект задающий атрибуты потока имеет тип `pthread_attr_t`. Такой объект должен быть инициализирован с помощью функции `int pthread_attr_init(pthread_attr_t *attr).` В результате объект будет содержать набор свойств потока по умолчанию для данной реализации потоков. А ресурсы, которые могут использоваться в системе для хранения этих атрибутов освобождаются вызовом функции (после того, как объект был использован в вызове `pthread_create` и больше не нужен) `int pthread_attr_destroy.`
Поток может быть "присоединяемым" (`joinable`) или "оторванным" (`detached`). Для установки этого свойства в атрибутах используется функция `int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate),` где `detachstate` можно установить в `PTHREAD_CREATE_JOINABLE` или в `PTHREAD_CREATE_DETACHED` соответственно.

POSIX Threads

```
pthread_attr_t tattr;
int ret;
ret = pthread_attr_init(&tattr);
```

В случае успешного завершения `ret` будет равен `0`.
По умолчанию создается атрибут с параметрами:

Атрибут `scope - PTHREAD_SCOPE_PROCESS`;
Новый поток не ограничен - не присоединен ни к одному процессу.
Атрибут `detachstate - PTHREAD_CREATE_JOINABLE`;
Нити создаются присоединенными (для освобождения ресурсов после завершения нити необходимо вызвать pthread_join).
Атрибут `stackaddr NULL`;
Новый поток получает адрес стека, выделенного системой.
Атрибут `stacksize 0`;
Новый поток имеет размер стека, определенный системой.
Атрибут `priority 0`;
Имеет приоритет 0.
Атрибут `inheritsched` PTHREAD_EXPLICIT_SCHED;
Нить не наследует приоритет родительского потока.
Атрибут `schedpolicy SCHED_OTHER`;
Нить использует фиксированные приоритеты, задаваемые ОС. Используется вытесняющая многозадачность (нить исполняется, пока не будет вытеснена другой нитью или не заблокируется на примитиве синхронизации)

POSIX Threads

Процедура `pthread_attr_destroy` используется для того, чтобы освободить память выделенную для атрибутов во время инициализации.

```
ret = pthread_attr_destroy(&tattr);
```

`pthread_attr_destroy` возвращает 0 - после успешного завершения - или любое другое значение - в случае ошибки.

POSIX Threads

Для каждого присоединяемого потока, один из других потоков явно должен вызвать процедуру `int pthread_join(pthread_t thread, void **value_ptr);`
Процедура `pthread_join` блокирующая и поток, вызвавший эту функцию, останавливается, пока не окончится выполнение потока `thread`. Если никто не вызывает pthread_join для присоединяемого потока, то завершившись, он не освобождает свои ресурсы, а это может служить причиной утечки памяти в программе. Аргумент value_ptr это указатель на указатель, возвращенный функцией завершившегося потока.

POSIX Threads

Любой потоку по умолчанию можно присоединиться вызовом `pthread_join` и ожидать его завершения. Однако в некоторых случаях статус завершения потока и возврат значения не нужны. Необходимо завершить поток и автоматически выгрузить ресурсы обратно в распоряжение ОС. В таких случаях можно обозначить поток отсоединившимся используя вызов процедуры

```
int pthread_detach(pthread_t thread);
```

При удачном завершении `pthread_detach()` возвращает код 0, ненулевое значение сигнализирует об ошибке.

Если поток отсоединён, его уже нельзя перехватить с помощью вызова `pthread_join()`, чтобы получить статус завершения. Также нельзя отменить его отсоединенное состояние.

POSIX Threads

Процедура `void pthread_exit (void *val_ptr)` - освобождает всю память, занятую данными нити, включая стек нити. Любые данные, помещенные в стек, становятся неопределенными, так как стек освобожден и соответствующая область памяти может использоваться другой нитью. Вызов `pthread_exit` делает объект, на который указывает параметр `val_ptr`, доступным другой нити исполнения, например, в породившей завершившуюся нить.

POSIX Threads

Процедура `int pthread_cancel (pthread_t thread_id)`
При удачном завершении возвращает 0, при ошибочном положительные значения. Выполняет запрос на отмену потока по его thread_id. Может быть выполнен немедленно, проигнорирован или выполнен чуть позже. Для определения дальнейшего поведения потока есть две функции:

```
int pthread_setcancelstate(int state, int *old
                           state);
int pthread_setcanceltype(int type, int *oldty
                          pe);
```

POSIX Threads

Процедура
`int pthread_setcancelstate(int state, int *olds tate)`

`pthread_setcancelstate` вызывается, чтобы проигнорировать или принять запрос на завершение. Запрос игнорируется, если аргументом `state` является `PTHREAD_CANCEL_DISABLE`, принимается если `state` имеет значение `PTHREAD_CANCEL_ENABLE.`

POSIX Threads

Если принятие запроса разрешено то вызывается процедура

```
int pthread_setcanceltype(int type, int
*oldtype)
```

Если аргумент `type` имеет значение `PTHREAD_CANCEL_ASYNCHRONOUS` то остановка потока произойдет мгновенно. Если `type` имеет значение `PTHREAD_CANCEL_DEFERRED`, запрос на завершение откладывается до следующей точки завершения.

POSIX Threads

Точками завершения являются такие функции, где выполняется проверка на наличие ожидающего обработки запроса на завершение. Если такой запрос ждёт обработки, завершение выполняется сразу же. В общем случае любая функция, которая приостанавливает выполнение текущего потока в течение длительного времени, должна быть точкой завершения. Такие функции для работы с `pthread`, как `pthread_join`, `pthread_cond_wait`, `pthread_cond_timedwait` и `pthread_testcancel` служат точками завершения.

```
int main() {
 int res;
 pthread_t a_thread;
 void *thread_result;
 res = pthread_create(&a_thread, NULL, thread_function, NULL);
 if (res != 0) { perror("Thread creation failed»); exit(EXIT_FAILURE);}
 sleep(3); printf("Canceling thread...n");
 res = pthread_cancel(a_thread);
 if (res != 0) { perror("Thread cancelation failed»);
exit(EXIT_FAILURE);}
 printf("Waiting for thread to finish...n");
 res = pthread_join(a_thread, &thread_result);
 if (res != 0) { perror("Thread join failed»); exit(EXIT_FAILURE);}
 exit(EXIT_SUCCESS);
}
void *thread_function(void *arg) {
 int i, res;
 res = pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
 if (res != 0) {
  perror("pthread_setcancelstate failed");exit(EXIT_FAILURE);
 }
 res = pthread_setcanceltype(PTHREAD_CANCEL_DEFERRED, NULL);
 if (res != 0) {
  perror{"Thread pthread_setcanceltype failed");
  exit(EXIT_FAILURE);
 }
 printf("thread_function is runningn");
 for(i = 0; i < 10; i++) { printf("still run (%d)\n», i);sleep(1);}
 pthread_exit(0);
}
```

POSIX Threads

Мьютекс (`Mutex - mutual exclusion` или взаимное исключение) – это объект, который может находиться в двух состояниях. Он либо заблокирован каким-то потоком, либо свободен. Поток, который захватил мьютекс, работает с участком кода. Остальные потоки, когда достигают мьютекса, ждут его разблокировки.

POSIX Threads

Мьютекс это экземпляр типа `pthread_mutex_t`. Перед использованием мьютекс необходимо инициализировать с помощью функции:

```
int pthread_mutex_init(pthread_mutex_t
*mutex, const pthread_mutexattr_t *attr);
```

Первый аргумент это указатель на мьютекс `mutex`, второй аргумент это указатель на атрибуты мьютекса `attr`. Для атрибутов допускается передавать `NULL`. В этом случае используются атрибуты поумолчанию.

POSIX Threads

Мьютексы бывают разных типов: обычный, рекурсивный, с проверкой ошибок. `PTHREAD_MUTEX_NORMAL` — для этого типа не проводится контроль `deadlock` в ситуации, когда поток, захвативший мьютекс, пытается захватить его повторно. Поэтому при попытке повторного захвата такого мьютекса тем же потоком этот поток будет безусловно блокирован, такой мьютекс уже некому разблокировать так как мьютекс может разблокироваться только своим владельцем.

`PTHREAD_MUTEX_ERRORCHECK` — включается контроль ошибок. В этом режиме регистрируются следующие ситуации - попытка повторного захвата мьютекса тем же потоком, попытка освобождения мьютекса, захваченного другим потоком, освобождение свободного мьютекса.
`PTHREAD_MUTEX_RECURSIVE` — мьютекс, допускающий рекурсивный захват. Поток, пытающийся захватить мьютекс, уже захваченный в этом потоке, сможет это сделать, при этом количество захватов будет учитываться при освобождении мьютекса. Другой поток сможет захватить такой мьютекс только тогда, когда он будет освобожден столько же раз, сколько был захвачен. Если поток пытается освободить мьютекс, захваченный другим потоком, или свободный мьютекс, то будет возвращено сообщение об ошибке (регистрируются ошибки, предусмотренные предыдущим типом, за исключением повторного захвата, который является для рекурсивного мьютекса штатным действием).

POSIX Threads

Типы мьютекса определяются с помощью процедуры:

```
int
pthread_mutexattr_settype( pthread_mutexat
tr_t* attr, int type);
```

Получить тип можно с помощью процедуры:

```
int pthread_mutexattr_gettype( const
pthread_mutexattr_t* attr, int* type);
```

POSIX Threads

Мьютекс может быть объявлен статическим образом с помощью константы `PTHREAD_MUTEX_INITIALIZER`.

После создания мьютекса он может быть захвачен с помощью функции:

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

После этого участок кода становится недоступным остальным потокам – их выполнение блокируется до тех пор, пока мьютекс не будет освобождён. Освобождение должен провести поток, заблокировавший мьютекс, вызовом:

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

После использования мьютекса его необходимо уничтожить с помощью процедуры:

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

POSIX Threads

pthread_mutex_init возвращает 0 в случае успешного выполнения.
Во время инициализации возможны следующие проблемы
`EAGAIN` – нехватка ресурсов (не памяти) для создания мьютекса
`ENOMEM` - нехватка памяти для инициализации
`EPERM` – нет разрешения на создание
`EBUSY` – попытка повторной инициализации не уничтоженного мьютекса.
`EINVAL` – неверные атрибуты функции.

POSIX Threads

`pthread_mutex_init` возвращает `0` в случае успешного выполнения. Во время инициализации возможны следующие проблемы
`EAGAIN` – нехватка ресурсов (не памяти) для создания мьютекса
`ENOMEM` - нехватка памяти для инициализации
`EPERM` – нет разрешения на создание
`EBUSY` – попытка повторной инициализации не уничтоженного мьютекса.
`EINVAL` – неверные атрибуты функции.

`pthread_mutex_destroy` может вернуть следующие ошибки
`EBUSY` – попытка уничтожить захваченный мьютекс, или уничтожить мьютекс, на который кто-то ссылается
`EINVAL` – значение мьютекса неверное

POSIX Threads

`pthread_mutex_lock` и `pthread_mutex_unlock` могут вылететь с ошибками:

`EINVAL` – ссылка на объект, который не является мьютексом

`EAGAIN` – нельзя захватить рекурсивный мьютекс, так как превышена максимальная глубина рекурсии

`EDEADLK` – поток, захвативший мьютекс, пытается его взять ещё раз.

`EPERM` – текущий поток не владеет мьютексом (попытка освободить чужой мьютекс)

```
#define NUM_THREADS 8
typedef struct _thread_data_t {
    int tid;
    double stuff;
} thread_data_t;

double shared_x;
pthread_mutex_t lock_x;

void *thr_func(void *arg) {
     thread_data_t *data = (thread_data_t *)arg;
    printf("hello from thr_func, thread id: %d\n", data->tid);
    pthread_mutex_lock(&lock_x);
    shared_x += data->stuff;
    printf("x = %f\n", shared_x);
    pthread_mutex_unlock(&lock_x);

    pthread_exit(NULL);
}
```

```
int main(int argc, char **argv) {
  pthread_t thr[NUM_THREADS];
  int i, rc;
  thread_data_t thr_data[NUM_THREADS];
  shared_x = 0;
  pthread_mutex_init(&lock_x, NULL);
  for(i = 0; i< NUM_THREADS; ++i) {
    thr_data[i].tid = i;
    thr_data[i].stuff = (i + 1) * NUM_THREADS;
    if((rc = pthread_create(&thr[i], NULL, thr_func, &thr_data[i]))) {
      fprintf(stderr, "error: pthread_create, rc: %d\n", rc);
      return EXIT_FAILURE;
    }
  }
  for(i = 0; i< NUM_THREADS; ++i) {
    pthread_join(thr[i], NULL);
  }

  return EXIT_SUCCESS;
}
```