Код

`unsigned long pthread_t` - это thread id в POSIX threads

Всегда успешны:
`pid_t getpid()` - process id
`pid_t getppid()` - parent pid, возвращает pid процесса, который вызвал fork, либо, если процесс уже завершился, вернет процесс, к которому был присоединен
`pid_t gettid()` - возвращает thread id. Если процесс однопоточный, то tid = pid. В многопоточном процессе все имеют одинаковый pid и разные tid.

pthread_create

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

pthread_create запускает новый поток, который начинает выполнение с start_routine, arg передается как единственный аргумент start_routine.

Поток может завершиться одним из следующих способов:

- Вызвать pthread_exit с указанием статуса завершения, который может быть прочитан другим потоком этого процесса, если вызван pthread_join.
- Выход из функции start_routine эквивалентно вызову pthread_exit.
- Отмена потока через pthread_cancel.
- Какой-либо из потоков вызвал exit или main закончил свою работу.

`pthread_attr_t attr` нужен для определения атрибутов при создании потока. Если attr == NULL, то используются атрибуты по умолчанию. Пример:

```
int main() {
    pthread_t tid;
    pthread_attr_t attr;

    pthread_attr_init(&attr);                          // инициализация атрибутов
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED); // создаём detached

    pthread_create(&tid, &attr, mythread, NULL);

    pthread_attr_destroy(&attr);                       // очистка атрибутов
    return 0;
}
```

Перед возвратом функция записывает в thread tid нового потока.

Возврат: в случае успеха возвращает 0; в случае ошибки возвращает номер ошибки, thread будет не определен.

Ошибки:

- EAGAIN недостаточно ресурсов для создания нового потока
- EAGAIN достигнул лимит по потокам
- EINVAL неверные настройки в атрибутах
- EPERM нет разрешения на установку политики планирования и параметров

strerror

`char *strerror(int errnum);` - возвращает указатель на строку описания ошибки. В случае ошибки возвращает: "Unknown error nnn"

pthread_exit

`void pthread_exit(void *retval);` - завершает работу вызывающего поток и возвращает значение с помощью retval, которое доступно другому потоку в том же процессе, который вызывает pthread_join.

Также вызывает все обработчики очистки, заданные через pthread_cleanup_push, которые еще не были запущены. Если в потоке есть данные, то после очистки вызываются функции-деструкторы.

При завершении потока общие ресурсы (мьютексы, семафоры, условные переменные и файловые дескрипторы) не высвобождаются.

Возврата нет. Ошибок нет.

pthread_self

`pthread_t pthread_self(void);` - возвращает ID потока, совпадающее с тем, что записывается в thread при pthread_create. **Всегда успешно. Ошибок нет.**

pthread_attr

`int pthread_attr_init(pthread_attr_t *attr);` - инициализирует объект атрибута потоков. После вызова можно задать отдельные атрибуты. Если вызвать функцию на уже инициализированную структуру, будет UB.

`int pthread_attr_destroy(pthread_attr_t *attr);` - когда объект атрибутов больше не нужен, его следует уничтожить. Если вызвать на уже уничтоженный объект, будет UB.

Возврат: 0 если ок, иначе ненулевое число.
Ошибки: ENOMEM

Теория

Потоки выполняются внутри одной программы независимо друг от друга, разделяя общую глобальную память (глобальные переменные и кучу). Потоки в процессе могут выполняться одновременно.

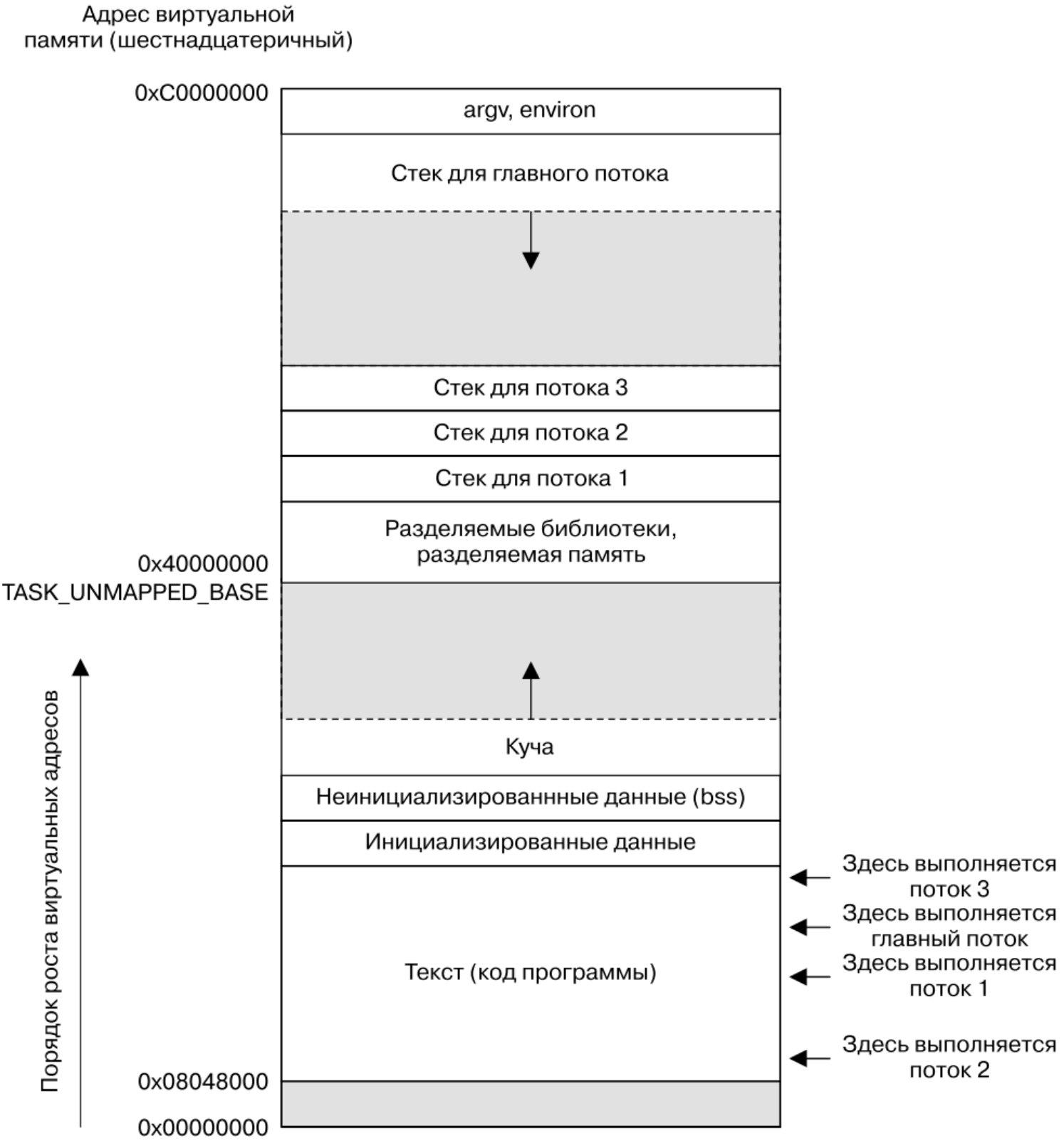
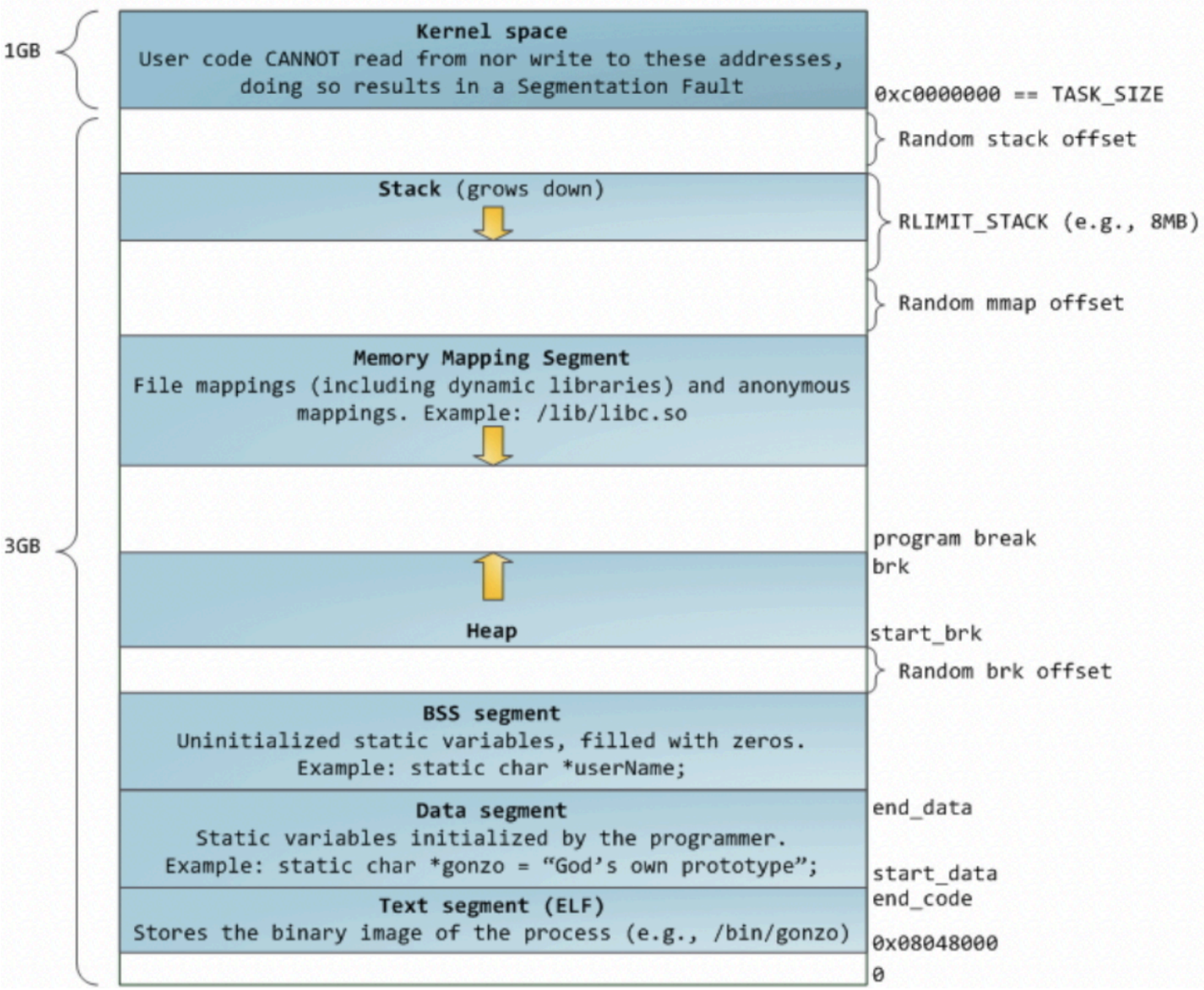


Рис. 29.1. Четыре потока, выполняющиеся внутри процесса (Linux/x86-32)

Где же находятся стеки потоков? Они расположены в memory mapping segment, там же где и динамические библиотеки.



Преимущества потоков перед процессами:

- Обмен информацией у процессов медленный и сложный
- Создание процесса потребляет относительно много ресурсов
- Обмен данными между потоками простой и быстрый, достаточно скопировать данные в общие переменные
- Создание потока минимум в 10 раз быстрее, так как атрибуты просто разделяются и не нужно копировать страницы и таблицы страниц

Важные разделяемые атрибуты:

 Идентификаторы процесса и его родителя.
 Идентификаторы группы процессов и сессии.
 Управляющий терминал.
 Учетные данные процесса (идентификаторы пользователя и группы).
 Дескрипторы открытых файлов.
 Блокировки записей, созданные с помощью вызова fcntl().
 Действия сигналов.
 Информация, относящаяся к файловой системе: umask, текущий и корневой каталог.

 Интервальные таймеры (setitimer()) и POSIX-таймеры (timer_create()).
 Значения отмены семафоров (semadj) в System V.
 Ограничения на ресурсы.
 Потребленное процессорное время (полученное из times()).
 Потребленные ресурсы (полученные из getrusage()).

Уникальные атрибуты потока:
 Идентификатор потока
 Маска сигнала.
 Данные, относящиеся к определенному потоку
 Альтернативный стек сигналов (sigaltstack()).
 Переменная **errno**.
 Настройки плавающей запятой (см. env(3)).
 Политика и приоритет планирования в режиме реального времени
 Привязка к ЦПУ
 Стек (локальные переменные и сведения о компоновке вызовов функций)

`pthread_self` используется функциями из состава pthread (pthread_join(), pthread_detach(), pthread_cancel() и pthread_kill();), а также идентификатором можно маркировать динамические структуры данных, что позволяет определить поток, который должен выполнить какие-то последующие действия с это структурой.

`pthread_equal` необходима из-за того, что тип pthread_t должен восприниматься как непрозрачный. В Linux он имеет тип unsigned long, но в других системах он может быть указателем или структурой.

Идентификаторы POSIX и обычного потока - не одно и то же. Идентификатор POSIX-потока присваивается и обслуживается реализацией поточной библиотеки. Идентификатор обычного потока представляет собой число, которое назначается ядром. Необходимо использовать POSIX для возможности переносимости между системами.

`pthread_join(pthread_t thread, void **retval)` ждет завершения потока, обозначенного аргументом thread (аналог wait). Эта операция называется соединением.

Атрибуты потоков

Атрибуты содержат такие сведения, как местоположение и размер стека потока, политику его планирования и приоритет, а также информацию о том, является ли поток присоединяемым или отсоединенным.

По умолчанию в NPTL:
Detach state = PTHREAD_CREATE_JOINABLE
Scope = PTHREAD_SCOPE_SYSTEM
Inherit scheduler = PTHREAD_INHERIT_SCHED
Scheduling policy = SCHED_OTHER
Scheduling priority = 0
Guard size = 4096 bytes

Stack address = 0x40196000
Stack size = 0x201000 bytes

Пример:

```
#include <pthread.h>
#include "tlpi_hdr.h"

static void *
threadFunc(void *x)
{
    return x;
}

int
main(int argc, char *argv[])
{
    pthread_t thr;
    pthread_attr_t attr;
    int s;

    s = pthread_attr_init(&attr);    /* Присваиваем значения по умолчанию */
     if (s != 0)
        errExitEN(s, "pthread_attr_init");

    s = pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_DETACHED);
    if (s != 0)
        errExitEN(s, "pthread_attr_setdetachstate");

    s = pthread_create(&thr, &attr, threadFunc, (void *) 1);
    if (s != 0)
        errExitEN(s, "pthread_create");

    s = pthread_attr_destroy(&attr);    /* Больше не нужен */
    if (s != 0)
        errExitEN(s, "pthread_attr_destroy");

    s = pthread_join(thr, NULL);
    if (s != 0)
        errExitEN(s, "pthread_join failed as expected");
    exit(EXIT_SUCCESS);
}
```

Вопросы

Почему созданный поток не выполняется? На инициализацию потока требуется время. Сразу после создания вызывается exit(0), который завершает процесс. Поэтому функция не успевает отработать.

Варианты решения:

- pthread_join(tid, NULL); - main ожидает завершения дочернего потока. Минус в том, что main блокируется до окончания работы mythread.
- pthread_exit(NULL); - процесс не завершится, пока все не-детачд потоки не завершатся. Нет лишних проверок. Минус в том, что exit-код не возвращается напрямую.

Почему можно использовать printf? Согласно документации printf является потоко-безопасной функцией.

Какие аргументы в clone используются при создании потока?

```
int clone(int (*fn)(void *), void *child_stack,
          int flags, void *arg, ...
          /* pid_t *ptid, void *newtls, pid_t *ctid */ );
```

```
CLONE_VM       // разделять память (адресное пространство)
CLONE_FS       // разделять fs info (текущий каталог, umask)
CLONE_FILES    // разделять таблицу открытых файлов
CLONE_SIGHAND  // разделять обработчики сигналов
CLONE_THREAD   // один PID, но разные TID (как в pthread)
CLONE_SYSVSEM  // разделять SysV семафоры
CLONE_SETTLS   // установить thread-local storage (TLS)
CLONE_PARENT_SETTID // записать tid в *ptid
CLONE_CHILD_CLEARTID // очистить *ctid при exit
```

Какого размера стек потока? Задается через атрибуты, по умолчанию 2МБ (0x201000 bytes)

Что за поле ---p рядом со стеком потока? Это stack guard - он защищает программу от переполнения стека потока. Если мы попали в guard блок, то будет отправлен сигнал SIGSEGV, что у нас проблемы.

Что такое Thread local storage? Thread Local Storage (TLS) - это механизм, при котором у каждого потока есть собственные копии глобальных/статических переменных, изолированные от других потоков и не требующие синхронизации для доступа к самим переменным. TLS позволяет объявлять переменные, которые выглядят как обычные глобальные или статические, но фактически существуют отдельно для каждого потока. Данные таких переменных читаются и изменяются только тем потоком, в контексте которого выполняется код. Это устраняет гонки за счёт изоляции, но не отменяет необходимость синхронизации для разделяемых ресурсов.

Как попадаем в поточную функцию(call stack внутри потока)?
pthread_create → clone получает новый стек → ядро создает новый task
между pthread_create и нашей функцией есть обертка start_routine, которая и вызывает нашу функцию.