Код

см. Lab 1.1

pthread_join

`int pthread_join(pthread_t thread, void **retval);` - ждет завершения потока, обозначенного аргументом thread (если поток уже завершился, она сразу же возвращается). Это операция присоединения.

После успешного вызова pthread_join вызывающий поток может совершить необходимую чистку (выделенные данные остаются на куче).

Если аргумент retval - ненулевой указатель, функция получает копию возвращаемого значения завершенного потока. Если вызвать pthread_join для уже присоединенного потока, то это может привести к UB: например, можем присоединиться к потоку, который был создан позже с этим идентификатором.

Если поток отсоединен, мы должны присоединиться к нему. Иначе завершенный поток превратится в аналог процесса-зомби, который будет впустую тратить ресурсы.

С для потоков аналогична вызову waitpid для процессов, но есть отличия:

- Потоки не имеют иерархии. Любой поток внутри процесса может присоединиться к другому потоку.
- Невозможно присоединиться к любому потоку.
- Не существует неблокирующей операции присоединения. Но можно реализовать что-то похожее с помощью переменных.

В сущности, поток может присоединяться только к тем потокам, о которых знает. Это создано для безопасности приватных (библиотечных) потоков, ведь иначе можно перехватить и легко сломать.

Возврат: в случае успеха возвращает 0; в случае ошибки возвращает номер ошибки
Ошибки:

- EDEADLK обнаружен дедлок, два потока пытаются присоединиться друг к другу или поток ждет завершения самого себя
- EINVAL thread не присоединяемый
- EINVAL другой поток уже ожидает присоединения к этому потоку
- ESRCH поток с id thread не найден

pthread_detach

`int pthread_detach(pthread_t thread);`

По умолчанию потоки являются присоединяемыми и требуют вызова join. Иногда возвращаемый статус неважен, нам нужно, чтобы поток просто отработал, освободил ресурсы и завершился. В этом случае мы помечаем поток как отсоединенный.

Чаще всего поток отсоединяет сам себя: `pthread_detach(pthread_self());`

Если поток был отсоединен, мы больше не сможем получить его возвращаемый статус. Также не можем снова сделать его присоединяемым.

pthread_detach отвечает за поведение потока после его завершения, но не за то, в каких обстоятельствах он завершается. Он также не устойчив вызову exit.

Возврат: в случае успеха возвращает 0; в случае ошибки возвращает номер ошибки
Ошибки:

- EINVAL thread не присоединяемый
- ESRCH поток с id thread не найден

pthread_attr

🔹 Атрибуты потока (`pthread_attr_t`)

Атрибут	Возможные значения	Значение по умолчанию	Описание
detachstate	`PTHREAD_CREATE_JOINABLE`, `PTHREAD_CREATE_DETACHED`	`PTHREAD_CREATE_JOINABLE`	Определяет, нужно ли ждать завершения потока через `pthread_join` или поток сам освобождает ресурсы
stacksize	Любое >= `PTHREAD_STACK_MIN`	8 МБ (Linux x86_64)	Размер стека потока
stackaddr	Любой корректный адрес памяти	NULL (ядро выделяет само)	Адрес начала стека потока (редко используется)
inheritsched	`PTHREAD_INHERIT_SCHED`, `PTHREAD_EXPLICIT_SCHED`	`PTHREAD_INHERIT_SCHED`	Наследует ли поток политику планирования родителя или использует явно заданную
schedpolicy	`SCHED_OTHER`, `SCHED_FIFO`, `SCHED_RR`	`SCHED_OTHER`	Политика планирования потока
schedparam	Любой приоритет, допустимый для политики	Текущий приоритет родителя	Приоритет потока
scope	`PTHREAD_SCOPE_SYSTEM`, `PTHREAD_SCOPE_PROCESS`	`PTHREAD_SCOPE_SYSTEM`	Видимость потока: в рамках системы или процесса (Linux игнорирует, всегда SYSTEM)
guardsize	Любое >= `PTHREAD_STACK_MIN`	4096 байт (1 страница)	Размер guard page, защищающей стек от переполнения

`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

`int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);` - устанавливает атрибут detachstate. detachstate может быть PTHREAD_CREATE_DETACHED или PTHREAD_CREATE_JOINABLE.

Возврат: в случае успеха возвращает 0; в случае ошибки возвращает номер ошибки
Ошибки: EINVAL недопустимое значение detachstate

Теория

Чтобы понимать, что происходит, попробуем написать правдоподобный аналог pthread_create:

```
typedef void *(*start_routine_t)(void*);

typedef struct _mythread {
    int mythread_id;
    start_routine_t start_routine;
    void* arg;
    void* retval;
    volatile int joined; // изменяется при вызове join
    volatile int exited;
    // detached flag
} mythread_struct_t;

int mythread_startup(void* args) {
    mythread_struct_t *mythread = (mythread_struct_t *)arg;
    mythread->retval = mythread->start_routine(mythread->arg);
    mythread->exited = 1;
    // wait until join (futex), а пока
    while(mythread->joined)
        sleep(1);
}

mythread_create(pthread_t *tid, start_routine, void* args) {
    mythread_struct_t *mythread;
    create_stack()

    mythread->mythread_id = thread_num
    mythread->start_routine = start_routine
    mythread->arg = arg;

    clone(mythread_startup) <- task

    deallocate_stack()
    ...
}

int mythread_join(mythread_t mytid, void** retval) {
    mythread_struct_t *mythread = mytid;
    // wait until thread ends
    while (!mythread_exited)
        sleep(1);

    *retval = mythread->retval;

    mythread->joined = 1;
    return 0;
```

```
}

thread_func() {
    check_cancel()
}
```

Создается структура, которая хранит состояние потока. Положим ее прямо перед началом стека потока. В tid запишем уникальный номер, который будет идентифицировать адрес этой структуры.

Вопросы

Как работает возврат?

После завершения нашей поточной функции возвращаемое значение будет записано в структуру потока. При этом стек функции очевидно будет освобожден, поэтому мы не можем хранить наши данные на стеке.

В пункте b наше число после завершения функции запишется в регистр, а затем в нашу структуру потока. Поэтому мы можем как выделить место на куче через malloc, так и просто вернуть число. Мы сообщим, что поток завершился, и будем ждать вызова join.

В пункте c мы также можем выделить место на куче или же просто вернуть строку, так как она будет располагаться не на стеке, а в сегменте data (или rodata).

Почему у нас заполняется память?

После завершения поточной функции стек освобождается, но структура остается до вызова join. А так как на этом месте нельзя создать новый поток, он выделяет новую память.

Какие данные остаются после завершения функции до вызова pthread_join?

Стек освободился. Структура потока (потоковый дескриптор), выделенное на куче, разделяемые данные остались

Что такое Deadlock?

Взаимная блокировка потоков. Когда первый дожидается завершения второго, а второй дожидается завершения первого. Или поток ожидает завершение самого себя.