Код

Отмена потока

`int pthread_cancel(pthread_t thread);` - отправляет заданному потоку запрос отмены. Выполнив запрос, она возвращается, не дожидаясь завершения заданного потока. Что именно происходит с потоком и в какой момент, зависит от его состояния и типа отмены.

Отмена потока происходит следующим образом:

- Вызываются обработчики очистки
- Вызываются деструкторы
- Поток завершается

После отмены потока при вызове join запишется значение **PTHREAD_CANCELED**.

Возврат: При успехе 0, иначе код ошибки.
Ошибки: ESRCH не найден поток с данным id

`void pthread_testcancel(void);` - искусственно созданная точка отмены. Если отмена отложена во время вызова данной функции, это означает, что вызывающий поток завершен. Если возможность отмены отключена, ничего не делает.

Возврата нет, ошибок нет.

Обработчики

```
void pthread_cleanup_push(typeof(void (void *)) *routine, void *arg);
void pthread_cleanup_pop(int execute);
```

Эти функции управляют стеком обработчиков очистки при отмене потока. Обработчик очистки - это функция, которая автоматически выполняется при отмене потока.

Функция pthread_cleanup_push() помещает routine в начало стека обработчиков очистки. При последующем вызове подпрограммы в качестве аргумента будет передан arg.
Функция pthread_cleanup_pop() удаляет routine, находящуюся в верхней части стека обработчиков очистки, и при необходимости выполняет её, если параметр execute не равен нулю.

Обработчик извлекается и выполняется, когда:

- Поток отменяется и все обработчики выполняются в порядке стека
- Поток завершается при pthread_exit и все обработчики выполняются
- Поток вызывает pthread_cleanup_pop исполняется верхний обработчик на стеке

Это парные функции: у push "{", у pop "}", так что использование return, break, continue или goto приводит к UB.

Реализация pthread_cleanup_push и pop:

```
do {
    __pthread_unwind_buf_t __cancel_buf;
    void (*__cancel_routine) (void *) = (routine);
    void *__cancel_arg = (arg);
    int __not_first_call = __sigsetjmp_cancel (__cancel_buf.__cancel_jmp_buf, 0);
    if (__glibc_unlikely (__not_first_call)) {
        __cancel_routine (__cancel_arg);
        __pthread_unwind_next (&__cancel_buf);
        /* NOTREACHED */
    }
    __pthread_register_cancel (&__cancel_buf);
    do {

        // Наш код

        do { } while (0);/* Empty to allow label before pthread_cleanup_pop.  */
    } while (0);
    __pthread_unregister_cancel (&__cancel_buf);
    if (execute)
        __cancel_routine (__cancel_arg);
} while (0);
```

Возврата нет, ошибок нет.

Теория

Состояние и тип отмены

`int pthread_setcancelstate(int state, int *oldstate);`
`int pthread_setcanceltype(int type, int *oldtype);`
Устанавливают флаги, которые позволяют управлять реакцией вызывающего потока на запрос отмены.

pthread_setcancelstate приводит состояние отмены потока к одному из двух значений:

- PTHREAD_CANCEL_DISABLE - поток нельзя отменить. Если получен запрос отмены, он будет отложен до момента включения возможности отмены.
- PTHREAD_CANCEL_ENABLE - поток можно отменить. Это состояние по умолчанию.

Предыдущее состояние отмены записывается в oldstate.

Временное отключение отмены бывает полезно, когда мы хотим выполнить участок кода целиком.

Если поток можно отменить, то реакция на запрос определяется типом:

- PTHREAD_CANCEL_ASYNCHRONOUS - отменить немедленно

- PTHREAD_CANCEL_DEFERRED - процедура отмены задерживается до точки отмены. Используется по умолчанию.

Предыдущий тип отмены записывается в oldtype.

Точки отмены

Запрос на отмену выполняется, когда поток достигает следующей точки отмены, то есть вызова одной из функций, определенных системой.

Таблица 32.1. Функции, которые должны быть точками отмены согласно стандарту SUSv3

accept()	nanosleep()	sem_timedwait()
aio_suspend()	open()	sem_wait()
clock_nanosleep()	pause()	send()
close()	poll()	sendmsg()
connect()	pread()	sendto()
creat()	pselect()	sigpause()
fcntl(F_SETLKW)	pthread_cond_timedwait()	sigsuspend()
fsync()	pthread_cond_wait()	sigtimedwait()
fdatasync()	pthread_join()	sigwait()
getmsg()	pthread_testcancel()	sigwaitinfo()
getpmsg()	putmsg()	sleep()
lockf(F_LOCK)	putpmsg()	system()
mq_receive()	pwrite()	tcdrain()
mq_send()	read()	usleep()
mq_timedreceive()	readv()	wait()
mq_timedsend()	recv()	waitid()
msgrcv()	recvfrom()	waitpid()
msgsnd()	recvmsg()	write()
msync()	select()	writev()

Это не все функции. В зависимости от ОС они могут дополняться. Чаще всего это функции, которые способны заблокировать выполнение потока.

Асинхронная отмена

Выполняется немедленно, что может вызвать немало проблем. Самая главная - мы не можем определить состояние потока: какие ресурсы были выделены, закрывались ли мьютексы и т.п. Также отмена может произойти **в любой момент** - например, во время вызова malloc, что приведет к непредвиденным результатам.

Это значит, что в асинхронно отменяемых потоках мы не можем выделять никаких ресурсов, владеть мьютексами, семафорами или блокировками, что делает невозможным использование ряда библиотечных функций.

В прочем, она может быть и полезной, например, при отмене потока, находящегося в вычислительном цикле.

Подробнее о pthread_cancel

Функция помечает поток как отмененный в поле структуры потока pd->cancelhandling. Записывает oldval и newval. Если у потока асинхронная отмена - генерирует сигнал SIGCANCEL.

Когда поток выполняет функцию, являющуюся точкой отмены, обертка внутри libc проверяет флаг отмены. Это обязательно для реализации по стандарту. Затем вызываются обработчики, деструкторы и pthread_exit.

Подробнее о pthread_testcancel

Функция считывает флаг и если отмена доступна и поток отменен, ставим в поле result PTHREAD_CANCELED и запускаем обработчики и деструкторы.

Затем мы возвращаем состояние на start_routine, которое мы запомнили перед вызовом поточной функции (записали в структуру потока) с помощью getcontext, setcontext.

Обработчики, освобождающие ресурсы

Если поток просто завершится при достижении точки отмены, разделяемые переменные и объекты библиотеки pthread (например, мьютексы) могут оказаться в несогласованном состоянии, что ведет к некорректным результатам, взаимным блокировкам, утечке памяти или аварийному завершению программы.

Обработчики для освобождения ресурсов - это функции, которые автоматически выполняются при отмене потока. Например, они изменяют значения глобальных переменных, освобождают память и закрывают мьютексы.

Каждый поток может иметь стек обработчиков. При отмене потока они выполняются по принципу LIFO (как стек).

Где же хранится этот стек и как это работает? Рассмотрим как работает (упрощенно) каждая функция:

```
struct pthread {
    ...
    /* List of cleanup buffers.  */
    struct _pthread_cleanup_buffer *cleanup_jmp_buf; // в структуре потока есть указатель на
стек чистильщиков
}
```

```
pthread_cleanup_push { // в linux это макрос
    pthread_unwind_buf_t cancel_buf; // обработчик создается на стеке потока
    // init
    pthread_register_cancel(&cancel_buf); // регистрируем обработчик
}

void pthread_register_cancel(buf) {
    buf->prev = THREAD_GETMEM(self, cleanup_jmp_buf) // записываем предыдущую вершину стека
    THREAD_SETMEM(self, cleanup_jmp_buf, buf) // записываем в структуру потока
}

pthread_cleanup_pop(execute) {
    pthread_unregister_cancel(&cancel_buf); // удаляем с вершины стека
    if (execute)
        cancel_routine() // если execute != 0 - выполняем
}

void pthread_unregister_cancel(&buf) {
    THREAD_SETMEM (THREAD_SELF, cleanup_jmp_buf, buf->prev); // ставим указатель на
предыдущий обработчик
}
```

Итого обработчики хранятся на стеке потока. В структуре потока хранится указатель на вершину стека обработчиков.

Как посмотреть все точки отмены

man 7 pthreads -> cancelation points