Лабораторная 4

Использованные функции :

Сигнатура функции :

```
#include <pthread.h>

int pthread_cancel(pthread_t thread);
```

pthread_cancel функция, которая отправляет запрос на отмену в поток, чей идентификатор подаётся как аргумент в функцию. /* В Linux отмена реализована с помощью сигналов. */

Среагирует ли целевой поток на запрос отмены и когда, зависит от двух атрибутов: его cancelability state и cancelability type.

- **cancelability state**

Состояние отмены потока (cancelability state), которое можно выствить с помощью **pthread_setcancelstate** , может быть включено (*enabled*) *(*по умолчанию для новых потоков) или отключено(*disabled*).

1. Если у потока cancelability state *disabled* , то запрос на отмену остается в очереди до тех пор, пока поток включает cancellation state.
2. Если поток включил cancelability state, то его тип cancelability type определяет, когда произойдет отмена.

- **cancelability type**

Тип отмены потока, который можно выствить с помощью **pthread_setcanceltype**, может быть либо асинхронным (*asynchronous*), либо отложенным(*deferred*) (значение по умолчанию для новых потоков).

1. Асинхронная возможность отмены означает что поток может быть отменен в любое время (обычно немедленно, но система этого не гарантирует. случай, когда не немедленно - например, планировщик запланировал другой поток в это время, а тот, который отменяют в это время не выполняется).

2. Возможность отсроченной отмены означает, что отмена будет отложена до того, пока поток не вызывет функцию, которая является точкой отмены (*cancellation point*). Примеры функций, которые являются или могут быть точками отмены

```
Являются :
           accept()
           close()
           connect()
           creat()
           fcntl() F_SETLKW
           nanosleep()
           open()
           pause()
           poll()
           pthread_join()
           pthread_testcancel()
           read()
           recv()
           send()
           sleep()
           system()
           usleep() [POSIX.1-2001 only (function removed in POSIX.1-2008)]
           wait()
           waitid()
           waitpid()
           write()
Могут быть :
           printf()
```

При выполнении запроса на отмену выполняются следующие действия для указанного в агрументах потока(в этом порядке):

1. Обработчики очистки отмены извлекаются (в обратном порядке, в котором они были запущены) и вызываются.
2. Вызываются деструкторы данных, зависящие от потока, в неуказанном порядке.
3. Поток завершается.

После завершения отмененного потока, если выполняется соединение с этим потоком используя pthread_join, получаем **PTHREAD_CANCELED** как статус выхода из потока. (Присоединение к потоку - это единственный способ узнать, что отмена завершена.)

возвращаемое значение :

запрос на отмену успешно поставлен в очередь — 0

ошибка — ненулевой код ошибки

возможные ошибки :

ESRCH — Не удалось найти поток с указанным идентификатором

сигнатура функций, описанных выше

```
#include <pthread.h>

int pthread_setcancelstate(int state, int *oldstate);
int pthread_setcanceltype(int type, int *oldtype);
```

1. **pthread_setcancelstate** устанавливает новое состояние (state), второй параметр нужен для того, чтобы функция поместила указатель на предыдущее состояние
   Состояние может иметь одно из двух значений :
   PTHREAD_CANCEL_ENABLE — поток можно отменить, значение по умолчанию
   PTHREAD_CANCEL_DISABLE — поток нельзя отменить. Если запрос на отмену получен, он блокируется до тех пор, пока не будет включена возможность отмены
2. **pthread_setcanceltype** устанавливает тип возможности отмены вызывающего потока на значение, указанное в параметре функции. Второй параметр - указатель на буфер, куда помещается предыдущий cancelability type

   Тип может иметь одно из двух значений :

   PTHREAD_CANCEL_DEFERRED — запрос на отмену откладывается до следующего потока вызывает функцию, которая является точкой отмены. Этот тип отмены по умолчанию для всех потоков, включая начальный поток.

   PTHREAD_CANCEL_ASYNCHRONOUS — поток может быть отменен в **любое время**. (Как правило, он будет отменен **сразу** же после получения уведомления об отмене запрос, но **система этого не гарантирует**.)

Операция set-and-get, выполняемая каждой из этих функций, является атомарный по отношению к другим потокам в процессе, вызывающем ту же функцию.

возвращаемое значение :

успех — 0

ошибка — ненулевой номер ошибки

возможные ошибки :

EINVAL — не валидный первый аргумент (новое состояние / новый тип)

Дополнительно :

Установка типа отмены на **PTHREAD_CANCEL_ASYNCHRONOUS** не всегда (скорее даже редко) является полезной. Поскольку поток может быть отменен в любое время, он не может безопасно резервировать ресурсы (например, выделять память с помощью malloc, получать мьютексы, семафоры и так далее.) Резервирование ресурсов небезопасно, потому что у приложения нет возможности знать, в каком состоянии находятся эти ресурсы, когда поток отменяется; т. е. произошла ли отмена до того, как ресурсы были зарезервированы, в то время как они были зарезервированы, или после того, как они были вышел? Кроме того, некоторые внутренние структуры данных (например, связанный список свободных блоков, управляемых семейством malloc функции) может остаться в несогласованном состоянии, если отмена происходит в середине вызова функции. Следовательно, *clean-up handlers* перестают быть полезными.

Я использую эти функции в поточной функции, так как printf() так же может быть точкой отмены и если поток будет отменён на этой строчке, то в выходном буфере который использует printf могут остаться данные, которые выведутся либо по завершении программы, либо если после присоединения потока сделать sleep и это не похоже на то поведение, которого мы ожидаем.

Так как по заданию через две секунды после создания дочерней нити, родительская нить должна прервать ее вызовом функции pthread_cancel, надо выждать эти две секунды, что можно сделать с помощью функции **sleep**

Сигнатура функции :

```c
#include <unistd.h>

unsigned int sleep(unsigned int seconds);
```

sleep() переводит вызывающий поток в спящий режим либо до тех пор, пока не истечет количество секунд реального времени, указанное в секундах, либо до тех пор, пока не поступит сигнал, который не будет проигнорирован.

возвращаемое значение :

запрошенное время истекло — 0

вызов был прерван обработчиком сигнала — количество секунд находился в спящем режиме