Лабораторная 1

До этого мы изучали процессы и их взаимодействие. Процесс - объект необходимый для исполнения программы. Поток же это контекст исполнения, то есть *каждый поток обладает своими : стеком, регистрами, счётчиком команд*. Каждый процесс может иметь один или более потоков. Все потоки одного процесса могут совместно пользоваться ресурсами процесса : открытые файлы, глобальные переменные (т.е. потоки одного процесса имеют *общее адресное пространство*).

Используемые функции :

Сигнатура функции

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

Использование в программе

- **return_code = pthread_create(&thread_id, NULL, &print_n_str, (void *)&args_for_child);**

pthread_create - функция запускающая новый поток в вызывающем процессе

Новый поток имеет доступ к адресному пространству процесса и наследует от вызывающего потока среду окружения арифметического сопроцессора и маску сигналов, однако набор сигналов, ожидающих обработки, для нового потока очищается.

аргументы функции :

1. первый аргумент, это указатель на pthread_t - указатель на область памяти, куда перед возвратом функции будет помещён идентификатор потока. Гарантируется, что все потоки однрого процесса имеют разные идентификаторы. Но система может повторно использовать идентификатор потока после присоединения(pthread_join) завершенного потока или завершения отсоединенного (detached) потока.

2. второй - указатель на структуру с атрибутами потока. Если же нам необходимо установить аттрибуты по умолчанию (как в условии задачи №1) то мы передаём NULL.

   основные аттрибуты

   detachstate

   Если для него установлено значение PTHREAD_CREATE_DETACHED, новый поток не может быть присоединён с помощью функции pthread_join () и его ресурсы автоматически возвращаются обратно в систему, после завершения потока. По умолчанию используется состояние PTHREAD_CREATE_JOINABLE. В таком случае предполагается, что создающий поток будет ожидать его завершения и выполнять pthread_join(). Joinable поток не освободит весь ресурс потока даже после завершения функции потока, пока какой-либо другой поток не вызовет pthread_join() с его идентификатором.

   stackaddr

   Указатель на область памяти, куда может быть размещён стек потока. По умолчанию NULL - выбор отдаётся системе

   stacksize

   Размер стека нового потока. По умолчанию - 2 Мбайта

   scope

   Указывает диапазон конкуренции между потоками, то есть эффективный диапазон приоритетов потоков. Стандарт POSIX определяет два значения: PTHREAD_SCOPE_SYSTEM и PTHREAD_SCOPE_PROCESS. Первое означает конкуренцию со всеми потоками в системе за процессорное время, а второе означает конкуренцию только с потоками в том же процессе за CPU. В настоящее время LinuxThreads реализует только значение PTHREAD_SCOPE_SYSTEM .

3. третий аргумент - указатель на функцию, которую должен исполнять создаваемый поток

4. указатель на аргумент функции. Это может быть как одна переменная, так и структура, если необходимо передать несколько аргументов.

Отметим также, что в качестве последнего аргумента мы передаём указатель на данные стека main потока. Они остаются валидными для нового созданного потока, так как если поток не является detached, то его ресурсы возвращаются системе после вызова pthread_join с его идентификатором, либо после завершения процесса.

Возвращаемое значение :

0 - успешное завершение функции

код ошибки - ошибка при создании потока

Возможные ошибки :

EAGAIN - был достигнут лимит RLIMIT_NPROC, который ограничивает количество процессов и потоков для реального идентификатора пользователя, либо общесистемное ограничение ядра на количество процессов и потоков, это значение можно узнать в /proc/sys/kernel/threads-max; или было достигнуто максимальное количество PID (/proc/sys/kernel/pid_max)

EINVAL - Переданы невалидные значения аттрибутов

EPERM - Нет разрешения на установку политики планирования указанную в аттрибутах.

Если функция возращает не SUCCESS_CODE (0), то выводится ошибка с помощью функции strerror_r

Сигнатура функции :

```
int strerror_r(int errnum, char *buf, size_t buflen);
                /* XSI-compliant */

char *strerror_r(int errnum, char *buf, size_t buflen);
                /* GNU-specific */
```

Использование в программе

- **strerror_r(return_code, buf, sizeof buf);**

XSI-совместимый strerror_r() помещает строку, описывающую ошибку с кодом errnum, в предоставленном пользователем буфер buf длины buflen

Специфичный для GNU strerror_r() возвращает указатель на строку содержащий сообщение об ошибке. Это может быть либо указатель на
строку, которую функция сохраняет в buf, либо указатель неизменяюмую строку (в этом случае buf не используется).

Функция работает подобно strerror, которая не является потокобезопасной по причине того, что использует внутренний статический буфер, который может быть изменён другим потоком (так как статические переменные совместно используются потоками).

аргументы функции

1. первый аргумент - код ошибки, который использованная до этого функция
2. второй - буфер, в который strerror_r поместит описание этой ошибки
3. размер буфера

Возвращаемое значение

XSI-compliant **strerror_r**() :

0 - успешное выполнение

код ошибки - ошибка при выполнении

GNU-specific **strerror_r**() :

описание ошибки - успешное выполнение

"Unknown error nnn" - ошибка при выполнении

Возможные ошибки :

EINVAL Переданное значение кода ошибки не является допустимым номером ошибки.

ERANGE Недостаточно места предлоставлено буфером для помещения описания ошибки

До этого зачастую при ошибках выводили их с помощью perror(), однако функции семейства pthread, как правило, возвращают код ошибки в случае неудачи. Они не изменяют значение переменной errno подобно другим функциям POSIX.

Экземпляр переменной errno для каждого потока предоставляется только для сохранения совместимости с существующими
функциями, которые используют эту переменную.

Поток может завершиться в следующих случаях :

1. Другой поток вызвал pthread_cancel с идентификатором этого потока
2. Поток вернулся (с помощью return) из переданной ему функции или (эквивалентное) был вызван pthread_exit в функции потока
3. Была вызвана функция exit() из любого потка или (эквивалентное)main thread выполнил return
4. был вызван pthread_exit

Отметим, что в ситуации пункта 3 завершается весь процесс, а не только поток, в котором были вызваны соответсвующие функции.

Не определено, какой поток закончит работу первее, но нам необходимо, чтобы оба потока распечатали свои 10 строк. Поэтому в конце функции main() мы вызываем функцию pthread_exit, которая, в отличие от return или exit не завершит созданный (и, возможно, ещё не закончивший свою работу) поток.

Сигнатура функции

```
#include <pthread.h>

noreturn void pthread_exit(void *retval);
```

Использование в программе

- **pthread_exit(NULL);**

Функция pthread_exit завершает вызвавший её поток и возвращает значение (через агрумент), которое можно получить (если поток joinable) через вызов pthread_join в другом потоке этого процесса с соответствующим идентификатором потока.

Возвращаемое значение

Фукнция не возвращается в вызвавщему её потоку

Возвожные ошибки

Функция всегда успешна