Лабораторная 3

Описание работы программы:

Вывод программы: строчки потоков могут перемешиваться. То есть сначала может вывести несколько, например 3 строчки одного потока потом другого, а потом снова срока первого. Вывод не будет всегда одинаковым, послежовательность вывода потоками определяется планировщиком. Но единая строка выводимая printf не может быть смешана со строкой выводимой другим потоком. Это обеспечивается вызовами flockfile и funlockfile, находящимися внутри вызова printf, которые блокируют и разблокируют соответсвенно доступ к печати (в stdin).

Функции stdio являются потокобезопасными. Это достигается путем присвоения каждому файловому объекту значения lockcount и (если значение lockcount не равно нулю) потока-владельца. Для каждого вызова библиотеки эти функции ожидают, пока объект FILE больше не будет заблокирован другим потоком, затем блокируют его, выполняют запрошенный ввод-вывод и снова разблокируют объект.

`stdin` И `stdout` - это не что иное, как указатели файла для стандартного ввода и вывода. из-за того, что вы можете изменить их в своем коде или с помощью командной строки, они не могут указывать на фактический ввод, потому что тогда вы не сможете его изменить.

`stdin` И `stdio` просто берут информацию из файлов, из которых `standard output` или `standard input` считывают или записывают. таким образом, его намного проще изменить с помощью команд и кодов.

Передача агрументов: в последнем аргументе pthread_create передаём указатель на аргумент функции. Это может быть как одна переменная, так и структура, если необходимо передать несколько аргументов. В данном случае передаём указаетель на структуру, храняющуюся на стеке main потока. Этот указатель будет оставаться валидным для созданного потока, так как гарантированно, что main поток завершится после всех соданных потоков. Это обеспечивается засчёт того, что мы делаем join всех созданных потоков в мэйне. Если бы этих джойнов не было и в конце мейна был бы pthread_exit, который не завершает весь процесс в отличие от return или exit, то указаетли всё равно бы остались валидными для нового созданного потока, так как если поток не является detached, а всё потоки в

том числе и main по умолчанию являются joinable, то его ресурсы возвращаются системе после вызова pthread_join с его идентификатором, либо после завершения процесса.

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 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 - было достигнуто 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 сделали свою работу полностью. 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

Сигнатура функции :

```
#include <pthread.h>

int pthread_join(pthread_t thread, void **retval);
```

Использование в программе:

- **return_code = pthread_join(thread_id, NULL);**

Функция ожидает завершения потока с указанным идентификатором. Если указанный поток уже завершил выполнение, то функция возвращается немедленно.

После успешного вызова функции pthread_join() вызывающему гарантируется, что указанный поток завершен.

Соединение с потоком, с идентификатором которого уже была выполнена функция pthread_join, приводит к неопределенному поведению.

аргументы функции:

1. первый - идентификатор потока
2. область памяти, куда размещается возращаемое потоком значение (оно было передано либо через pthread_exit либо для всех кроме main потока через return (неявно вызывается pthread_exit))

   Так как в данной задаче нам неинтересно возвращаемое значение, то мы указываем в этом параметре NULL

возвращаемое значение :

0 - успешное завершение

код ошибки - ошибка при выполнении функции

возможные ошибки :

EDEADLK — deadlock так как два потока попытались соединиться друг с другом; или идентификатор потока является идентификатором вызывающего потока.

EINVAL — указанный поток не является joinable.

EINVAL — другой поток уже ждет, чтобы присоединиться к указанному потоку

ESRCH — не удалось найти поток с идентификатором thread.

Не рекомендуется не делать join с joinable потоками, так как это приводит к созданию “зомби потоков” которые потребляют системные ресурсы, и когда накопилось достаточное количество потоков зомби, больше не будет возможности создавать новые потоки (ограничение в /proc/sys/kernel/threads-max).

Помимо joinable threads существуют ещё detached threads

Когда detached поток завершается, его ресурсы автоматически высвобождаются обратно в систему без необходимости соединения другого потока с завершенным потоком

Сделать поток detahed можно с помощью фукнции pthread_detach

Сигнатура фукнции :

```
#include <pthread.h>

int pthread_detach(pthread_t thread);
```

Функция pthread_detach() помечает поток, идентифицированный thread, как отсоединенный.

Попытка отсоединить уже отсоединенный поток приводит к неопределенному поведению.

аргументы функции :

1. индектификатор потока, который помечается как detached

Возможные ошибки :

EINVAL - указанный поток не является joinable

ESRCH -нет потока с таким идентификатором

Как только поток был отсоединен, он не может быть соединен с pthread_join(3) или снова стать соединяемым.

Новый поток может быть создан в detached состоянии с помощью pthread_attr_setdetachstate(3) для установки аргумента аттрибутов pthread_create(3).

Следует вызвать либо pthread_join(3), либо pthread_detach() для каждого потока, который создает приложение, так как иначе ресурсы любых потоков, для которых одно из этих действий не было выполнено, будут будет освобожден только когда завершится процесс.