Реализация потоков POSIX

Интерфейс POSIX

POSIX (Portable Operating System Interface for Computer Environments – независимый от платформы системный интерфейс для компьютерного окружения) – это стандарт IEEE (Institute of Electrical and Electronics Engineers – институт инженеров по электротехнике и радиоэлектронике), описывающий системные интерфейсы для открытых операционных систем.

Этот стандарт подробно описывает систему виртуальной памяти, многозадачность и технологию переноса операционных систем. Таким образом, на самом деле POSIX представляет собой множество стандартов POSIX.1003.1XXX.

Компиляция программ POSIX API

Стандартным средством компиляции программ для многих UNIX-систем является компилятор GNU C (C++). Интегрированные среды разработки обычно являются программными надстройками над GNU C. В Linux такими надстройками являются среды:

- CLion
- CodeBlocks
- K-Develop (среда, связанная с менеджером окон KDE, по интерфейсу близка к Visual Studio);
- XWPE (текстовая IDE, по возможностям близка к Turbo-C);
- C-Forge (графическая IDE в X-window);
- ANJTA и др. IDE для GNU C в Linux.

Для выполнения лабораторных работ можно использовать компилятор GNU C вне зависимости от наличия IDE. В ОС Linux данный компилятор запускается из командной строки командой:

```
gcc <имя исходного файла .c> -o <имя исполняемого файла>
```

ключи:

- `L` – путь к дополнительным библиотекам (`/usr/lib` для библиотеки потоков в Linux);
- `l` (L малое) – имя библиотеки (pthread для библиотеки потоков в Linux);
- `I` – путь к включаемым файлам (`/usr/inc` или `/usr/include` для библиотеки потоков в Linux).

Пример:

```
gcc /home/username/lab.c -o /home/username/lab -I /usr/include/ -L /usr/lib/
-lpthread
```

Для компиляции программы, написанной на языке C++, например, в случае использования в коде стандартных потоков ввода-вывода `std::cin` и `std::cout` или других элементов пространства имен `std` и языка C++ в целом, следует использовать следующую команду:

```
g++ /home/username/lab.cpp -o /home/username/lab -I /usr/include/ -L
/usr/lib/ -lpthread
```

По умолчанию компилятор C++ может отсутствовать в системе и его будет необходимо установить перед первым использованием. Команда для установки компилятора GNU C++ (утилиты `g++`)

зависит от конкретного дистрибутива ОС Linux. Например, в случае отсутствия компилятора C++ в Ubuntu для установки следует выполнить команду `sudo apt-get install g++`.

Следует учесть, что библиотеки в ОС Linux имеют префикс `lib` и расширения `.so`, `.a` . При использовании ключа `-l` в обоих компиляторах `gcc` и `g++` префикс `lib` подставляется компилятором автоматически, а расширение опускается. В наших примерах библиотека потоков должна была называться `libpthread.so` и находиться в каталоге `/usr/lib/`. Кроме того, следует заметить, что в UNIX-системах не принято присваивать исполняемым файлам имена с расширениями (как в DOS или Windows), за исключением файлов с библиотеками.

Создание и завершение потоков

- Функция для создания потока с атрибутами, указанными в attr, выполняющего функцию start_routine с аргументом arg.

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

Аргументы:

`thread` – сюда будет помещен идентификатор созданного потока;
`attr` – атрибуты потока (если `NULL`, то используются стандартные);
`start_routine` – функция, которую будет выполнять поток;
`arg` – аргументы для фукнции (обычно – структура с заданием).

- Функция, приостанавливающая текущий поток до завершения другого потока.

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

Аргументы:

`thread` – поток, которого дожидается текущий поток;
`value_ptr` – возвращаемое потоком значение, или NULL, если оно не интересует.

Средства синхронизации

Мьютекс

- Функция для инициализации мьютекса.

```
int pthread_mutex_init  (pthread_mutex_t *restrict mutex,
      const pthread_mutexattr_t *restrict attr);
```

Аргументы:

`mutex` – пустая переменная (или уничтоженный флаг), которая будет инициализирована для последующего использования;
`attr` – атрибуты создаваемого мьютекса (если NULL – атрибуты по умолчанию).

- Функция для удаления мьютекса.

```
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

Аргументы:

`mutex` – жертва функции, мьютекс, который будет уничтожен.

После удаления мьютекс не может быть использован. Удалять заблокированные мьютексы не рекомендуется – в таких случаях дальнейшее поведение и результат работы программы становятся непредсказуемыми.

- Функция для перевода мьютекса в заблокированное состояние.

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

Аргументы:

`mutex` – блокируемый мьютекс.

Если мьютекс уже заблокирован, то выполнение кода приостанавливается до тех пор, пока мьютекс не будет разблокирован и функция не сможет нормально отработать.

- Функция, освобождающая мьютекс.

```
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

Аргументы:

`mutex` – разблокируемый мьютекс.

При вызове этой функции мьютекс должен быть в заблокированном состоянии.

- Неблокирующая функция, пытающаяся захватить мьютекс.

```
int pthread_mutex_trylock  ((pthread_mutex_t *__mutex));
```

Аргументы:

`mutex` – мьютекс, который пытается захватить поток.

При вызове этой функции вызывающий поток ждет некоторое время и, если мьютекс недоступен, продолжает работу без его захвата.

Семафор

- Функция для инициализации семафора.

```
int sem_init  ((sem_t *__sem, int __pshared, unsigned int __value));
```

Аргументы:

`sem` – инициализированный объект, возвращаемый функцией;

`pshared` – если отличен от нуля, то семафор разделяется между процессами, в противном случае разделение возможно только между потоками управления вызывающего процесса;

`value` – начальное значение создаваемого неименованного семафора.

- Функция для освобождения системных ресурсов, выделенных объекту семафора.

```
int sem_destroy  ((sem_t *__sem));
```

Аргументы:

`sem` – семафор.

- Функция для захвата семафора.

```
int sem_wait  ((sem_t *__sem));
```

Аргументы:

`sem` – семафор.

Функция уменьшает на 1 текущее значение общего семафора, на идентификатор которого указывает аргумент `sem`. Если прежнее значение семафора было 0, поток, выполняющий функцию `sem_wait`, блокируется до изменения значения семафора.

- Функция для неблокирующего захвата семафора.

```
int sem_trywait  ((sem_t *__sem));
```

Аргументы:

`sem` – семафор.

Функция проверяет, равен ли семафор 0. Если равен, то уменьшает его на единицу. В противном случае функция сразу возвращает ошибку.

- Функция для освобождения семафора.

```
int sem_post  ((sem_t *__sem));
```

Аргументы:

`sem` – семафор.

Функция увеличивает на 1 текущее значение общего семафора, на идентификатор которого указывает аргумент sem. Если прежнее значение семафора было 0 и есть потоки, заблокированные в своей операции `sem_wait`, то один из них разблокируется, но какой именно – не определено

- Функция для просмотра значения переменной семафора без его захвата или освобождения.

```
int sem_getvalue ((sem_t *__sem, int *__sval));
```

Аргументы:

`sem` – семафор;

`sval` – адрес, по которому сохраняет текущее значение семафора.

Функция записывает текущее значение общего семафора, на идентификатор которого указывает аргумент `sem`, в целочисленную переменную, расположенную по адресу, задаваемому аргументом `sval`. Если семафор заблокирован, возвращается либо 0, либо отрицательное число, модуль которого соответствует количеству потоков, ожидающих разблокирования семафора.

Тривиальный пример организации потоков

```
myfunc()
{
    * * *
    //выполняем задание,
    // если нужно --- читаем данные по адресу в аргументе
    * * *
    return return_value;
}
main()
{
    * * *
    // что-то делаем
    * * *
    pthread_create(...,myfunc,...)
    // не забываем запомнить идентификатор потока
    * * *
    // что-то делаем
    * * *
    pthread_join(&return_value)
    // если нужно – анализируем выходное значение
    * * *
}
```