

Процессы, потоки

Определения

- Определения процесса
 - Понятие определяющее работу программы в контексте конкретной ОС;
 - Программа, выполняющаяся в собственном виртуальном адресном пространстве;
 - **Буч** Процесс = поток управления. Определяет независимую динамическую деятельность в системе.
 - **ISO9000** Совокупность взаимосвязанных и взаимодействующих действий, преобразующих входы в выходы.
- Определения потока
 - **andr2003** “облегченный” процесс – процесс с собственным программным счетчиком и стеком выполнения, но без “тяжелого” контекста (типа таблиц страниц), связанного с работой приложения.
- Процесс
 - владеет адресным пространством
 - владеет потоками
 - владеет ресурсами и устройствами
 - идентифицируется подсистемой безопасности
- Поток
 - принадлежит процессу
 - умеет исполняться
 - имеет доступ к ресурсам процесса
- Аквариум с рыбами

Примитивы pthreads

Создание

- `int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void*), void *arg);`
 - `thread` OUT В результате успешного срабатывания функции по указанному адресу будет размещен описатель порожденного потока.
 - `attr` IN Атрибуты потока. Задают свойства потока. Может быть NULL. Описание атрибутов см. ниже.
 - `start_routine` IN Указатель на функцию потока. Выполнение потока состоит в выполнении этой функции.
 - `arg` IN/OUT Указатель, который будет передан функции потока в качестве параметра. OUT – в том смысле, что функция потока может менять содержимое памяти с использованием этого указателя. `pthread_create` сожержимого не меняет, просто передает указатель функции потока. Функция потока сама интерпретирует содержание памяти по этому адресу.

Атрибуты

Объект задающий атрибуты потока имеет тип `pthread_attr_t`. Такой объект должен быть инициализирован с помощью функции

- `int pthread_attr_init(pthread_attr_t *attr);`

В результате объект будет содержать набор свойств потока по умолчанию для данной реализации потоков. А ресурсы, которые могут использоваться в системе для хранения этих атрибутов освобождаются вызовом функции (после того, как объект был использован в вызове `pthread_create` и больше не нужен)

- `int pthread_attr_destroy(pthread_attr_t *attr);`

Присоединение и отрыв

Поток может быть “присоединяемым” (`joinable`) или “оторванным” (`detached`). Для установки этого свойства в атрибутах используется функция

- `int pthread_attr_setdetachstate(pthread_attr_t *attr, int detachstate);`

где `detachstate` можно установить в `PTHREAD_CREATE_JOINABLE` или в `PTHREAD_CREATE_DETACHED` соответственно. Присоединяемые и оторванные потоки

Для каждого присоединяемого потока, один из других потоков явно должен вызвать функцию

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

Поток, вызвавший эту функцию, останавливается, пока не окончится выполнение потока `thread`. Если никто не вызывает `pthread_join` для присоединяемого потока, то завершившись, он не освобождает свои ресурсы, а это может служить причиной утечки памяти в программе. `value_ptr` (OUT) – это указатель на указатель, возвращенный функцией завершившегося потока.

Взаимное исключение потоков

Для организации взаимного исключения потоков при доступе к разделяемым данным используются мьютексы (`mutex = mutual exclusion`), объекты типа `pthread_mutex_t`. Мьютекс должен быть инициализирован перед использованием с помощью функции

- `int pthread_mutex_init(pthread_mutex_t *mutex, const pthread_mutexattr_t *attr);`
 - Параметр `attr` (IN) задает атрибуты мьютекса. Можно передать `NULL` для принятия атрибутов по умолчанию. Ресурсы, занимаемые мьютексом, могут быть освобождены функцией
- `int pthread_mutex_destroy(pthread_mutex_t *mutex);`

Для захвата мьютекса поток использует (см. пример) функцию

- `int pthread_mutex_lock(pthread_mutex_t *mutex);`

а для освобождения

- int pthread_mutex_unlock(pthread_mutex_t *mutex);

Условные переменные

Поясним использование условных переменных на примере. Например, поток номер 1 должен выполнить некоторые действия, когда значение некоторого глобального счетчика counter достигнет критического значения criticalVal, причем значение счетчика меняет поток номер 2. Тогда, чтобы первый поток не крутился в цикле, все время проверяя значение counter, можно использовать условную переменную, с помощью которой поток номер 1 устанавливается в состояние ожидания до тех пор, пока поток номер 2 не просигнализирует о наступлении нужного события:

- ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ

```
pthread_mutex_t mutex;
//условная переменная
pthread_cond_t cond;
//счетчик
int counter = 0;
```

- ПОТОК НОМЕР 1

```
//захват мьютекса
pthread_mutex_lock(&mutex);
//если значение не равно критическому
if(counter!=criticalValue)
//останавливаемся в ожидании этого события.
//При этом мьютекс будет разблокирован. Как только событие наступит,
//и мьютекс будет отпущен вторым потоком,
//мьютекс снова захватывается и выполнение потока 1 продолжается
    pthread_cond_wait(&cond, &mutex);
//обработка критического значения
processCriticalSection();
//освобождение мьютекса
pthread_mutex_unlock();
```

- ПОТОК НОМЕР 2

```
do
{
...
//захват мьютекса
pthread_mutex_lock(&mutex);
//изменение значения счетчика
doSomethingWith(&counter);
//если событие наступило
if(counter==criticalValue)
//просигнализовать об этом ждущим на условной переменной cond
    pthread_cond_signal(&cond);
//отпустить мьютекс
pthread_mutex_unlock(&mutex);
```

```
...
}while(...);
```

Условная переменная должна быть инициализирована функцией

- int pthread_cond_init(pthread_cond_t *cond, const pthread_condattr_t *attr);

А ресурсы, занятые ей, могут быть освобождены функцией

- int pthread_cond_destroy(pthread_cond_t *cond);

From:

<http://wiki.osll.ru/> - Open Source & Linux Lab



Permanent link:

<http://wiki.osll.ru/doku.php/etc:teach:parallel:threads?rev=1193641877>

Last update: **2008/01/03 02:32**