Семинар 6. Нити исполнения в UNIX.

Понятие о нити исполнения (thread) в UNIX. Идентификатор нити исполнения. Функция *pthread_self().* Во многих современных операционных системах существует расширенная реализация понятия процесс, когда процесс представляет собой совокупность выделенных ему ресурсов и набора нитей исполнения. Нити процесса разделяют его программный код, глобальные переменные и системные ресурсы, но каждая нить имеет свой собственный программный счетчик, свое содержимое регистров и свой собственный стек. Поскольку глобальные переменные у нитей исполнения являются общими, то они могут использовать их, как элементы разделяемой памяти, не прибегая к механизму, описанному выше.

В различных версиях операционной системы UNIX существуют различные интерфейсы, обеспечивающие работу с нитями исполнения. Мы с вами кратко ознакомимся с некоторыми функциями, позволяющими разделить процесс на thread'ы и управлять их поведением, соответствующими стандарту POSIX. Нити исполнения, удовлетворяющие стандарту POSIX, принято называть POSIX thread'ами или кратко pthread'ами.

К сожалению, операционная система Linux не полностью поддерживает нити исполнения на уровне ядра системы. При создании нового thread'a запускается новый традиционный процесс, разделяющий с родительским традиционным процессом его ресурсы, программный код и данные, расположенные вне стека, т.е. фактически действительно создается новый thread, но ядро не умеет определять, что эти thread'ы являются составными частями одного целого. Это знает только специальный процесс-координатор, работающий на пользовательском уровне и стартующий при первом вызове функций, обеспечивающих POSIX интерфейс для нитей исполнения. Поэтому мы сможем наблюдать не все преимущества использования нитей исполнения (в частности, ускорить решение задачи с их помощью вряд ли получится), но даже в этом случае, thread'ы можно использовать как очень удобный способ для создания процессов с общими ресурсами, программным кодом и разделяемой памятью.

Каждая нить исполнения, как и процесс, имеет в системе свой собственный уникальный номер - идентификатор thread'a. Поскольку традиционный процесс в концепции нитей исполнения трактуется как процесс, содержащий единственную нить исполнения, то мы можем узнать идентификатор этой нити и для любого обычного процесса. Для этого используется функция *pthread_self()*. Нить исполнения создаваемую при рождении нового процесса принято называть начальной или главной нитью исполнения этого процесса.

Создание и завершение thread'a. Функции *pthread_create(), pthread_exit(), pthread_join().* Нити исполнения, как и традиционные процессы, могут порождать нити-потомки, правда, только внутри своего процесса. Каждый будущий thread внутри программы должен представлять собой функцию с прототипом

*void *thread(void *arg);*

Параметр *arg* передается этой функции при создании thread'a и может, до некоторой степени, рассматриваться как аналог параметров функции *main()*. Возвращаемое функцией значение может интерпретироваться как аналог информации, которую родительский процесс может получить после завершения процесса-ребенка. Для создания новой нити исполнения применяется функция *pthread_create()*. Мы не будем рассматривать ее в полном объеме, так как детальное изучение программирования с использованием thread'ов не является целью данного курса. Важным отличием этой функции от большинства других системных вызовов и функций является то, что в случае неудачного завершения она возвращает не отрицательное, а положительное значение, которое определяет код ошибки, описанный в файле errno.h. Значение системной переменной *errno* при этом не устанавливается. Результатом выполнения этой функции является появление в системе новой нити исполнения, которая будет выполнять функцию, ассоциированную со thread'ом, передав ей специфицированный параметр, параллельно с уже существовавшими нитями исполнения процесса.

Созданный thread может завершить свою деятельность тремя способами

С помощью выполнения функции *pthread_exit()*. Функция никогда не возвращается в вызвавшую ее нить исполнения. Объект, на который указывает параметр этой функции, может быть изучен в другой нити исполнения, например, в породившей завершившийся thread. Этот параметр, следовательно, должен указывать на объект, не являющийся локальным для завершившегося thread'a, например, на статическую переменную.

С помощью возврата из функции, ассоциированной с нитью исполнения. Объект, на который указывает адрес, возвращаемый функцией, как и в предыдущем случае, может быть изучен в другой нити исполнения, например, в породившей завершившийся thread, и должен указывать на объект, не являющийся локальным для завершившегося thread'a.

Если в процессе выполняется возврат из функции *main()* или где-либо в процессе (в любой нити исполнения) осуществляется вызов функции *exit()*, то это приводит к завершению всех thread'ов процесса.

Одним из вариантов получения адреса, возвращаемого завершившимся thread'ом, с одновременным ожиданием его завершения является использование функции *pthread_join()*. Нить исполнения, вызвавшую эту функцию, переходит в состояние

ожидание до завершения заданного thread'a. Функция позволяет также получить указатель, который вернул завершившийся thread в операционную систему.

Прогон программы с использованием двух нитей исполнения. Для иллюстрации вышесказанного предлагается написать программу в которой работают две нити исполнения. При работе редактора связей необходимо явно подключить библиотеку функции для работы с pthread'ами, которая не подключается автоматически. Это делается с помощью добавления к команде компиляции и редактирования связей параметра -*lpthread* - подключить библиотеку pthread. Откомпилируйте эту программу и запустите на исполнение. Обратите внимание на отличие результатов этой программы от похожей программы, иллюстрировавшей создание нового процесса. Программа, создававшая новый процесс, печатала дважды одинаковые значения для переменной a, так как адресные пространства различных процессов независимы, и каждый процесс прибавлял 1 к своей собственной переменной a. Рассматриваемая программа печатает два разных значения, так как переменная a является разделяемой, и каждый thread прибавляет 1 к одной и той же переменной.