СИНТАКСИС

```
#include <sys/ptrace.h>

long
ptrace(enum __ptrace_request request, pid_t pid, void *addr, void *data);
```

ВВЕДЕНИЕ

Часто для того, чтобы разобраться почему код работает не так, программист прибегает к помощи отладчиков, однако, то как они устроены и каким образом действуют, известно далеко не всем. В этой статье будет дано краткое описание механизма отладки, который используется операционной системой Linux и приведён короткий пример его использования.

ОСНОВНЫЕ СВЕДЕНИЯ О **ptrace**()

В ОС Linux для отладки используется системный вызов ptrace(2). В секции СИНТАКСИС приведён синтаксис этого вызова. Аргумент *request* определяет тип операции: будет ли это попытка начать отладку процесса, или это будет запрос данных по какому-либо адресу. Все прочие аргументы являются опциональными и зависят от значения *request*.
Начать отладку процесса можно двумя способами:

1. С помощью запроса PTRACE_TRACEME текущий процесс будет отлаживаться его родителем. Любой сигнал, полученный текущим процессом, вызовет его остановку, а родительский процесс может быть оповещён об этом сигналом SIGCHLD. Затем родитель при помощи системного вызова wait(2) (или подобного) узнаёт идентификатор остановленного процесса. Следует заметить, что для этого типа запроса все остальные аргументы игнорируются.

2. С помощью запроса PTRACE_ATTACH можно подключиться к уже существующему процессу, передав в *pid* его идентификатор.

При успешном завершении всех типов запросов, за исключением PTRACE_PEEK*, **ptrace**() возвращает 0. Если запрос завершился с ошибкой, **ptrace**() возвращает -1, а код ошибки заносится в *errno*.

Кратко опишем некоторые типы запросов:

PTRACE_GETREGS, PTRACE_SETREGS, PTRACE_GETFPREGS, PTRACE_SETFPREGS
: Получаем или изменяем основные регистры или регистры FPU. Указатель на структуру с регистрами передаётся в *data*. Для основных регистров используется структура *struct user_regs_struct*, для регистров FPU *struct user_fpregs_struct*. Определения структур находятся в <sys/user.h>.

PTRACE_PEEK_DATA, PTRACE_PEEK_TEXT, PTRACE_POKE_DATA, PTRACE_POKE_TEXT
: Получаем или изменяем данные в памяти исследуемого процесса по переданному в *addr* адресу. При запросе PTRACE_PEEK* в случае успеха возвращаются запрошенные данные. Так как -1 (0xffffffff) также может быть адресом, для проверки на ошибку необходимо дополнительно проверять значение *errno*.

PTRACE_CONT
: Продолжает выполнение остановленного процесса. Если *data* не NULL и не SIGSTOP, значение интерпретируется как сигнал, который посылается процессу.

PTRACE_SINGLESTEP, PTRACE_SYSCALL
: Продолжает выполнение остановленного процесса, как и в случае с PTRACE_CONT, но указывает, что процесс должен остановиться при переходе к следующей инструкции (PTRACE_SINGLESTEP) или при входе/выходе из системного вызова (PTRACE_SYSCALL). Аргумент *data* будет интерпретирован как и в случае с PTRACE_CONT.

PTRACE_DETACH
: Отменяет эффекты PTRACE_TRACEME, PTRACE_ATTACH для процесса с указанным *pid*, и продолжает выполнение как в случае с PTRACE_CONT.

Важно помнить, что все запросы, перечисленные выше, работают только если отлаживаемый процесс остановлен, в чём нужно убедиться с помощью системного вызова `wait`(2), иначе **`ptrace`**() вернёт -1 и установит *errno* в `ESRCH`. Описание остальных запросов, не перечисленных выше, можно посмотреть в `ptrace`(2).

ПРИМЕРЫ

Теперь напишем простой пример использования **`ptrace`**(). Программа будет отслеживать все попытки вызова `open`(2), сделанные дочерним процессом. В нашем примере роль отлаживаемой программы будет играть **`/bin/ls`**. Для краткости, опущены проверки на ошибки и не учитывается порядок байтов. В реальной программе, пожалуй, так делать не стоит. Две версии файла с примером (одна для x86, другая для x86_64) запакованы в архив вместе с данной статьёй.