Системное программирование

Лекция 12

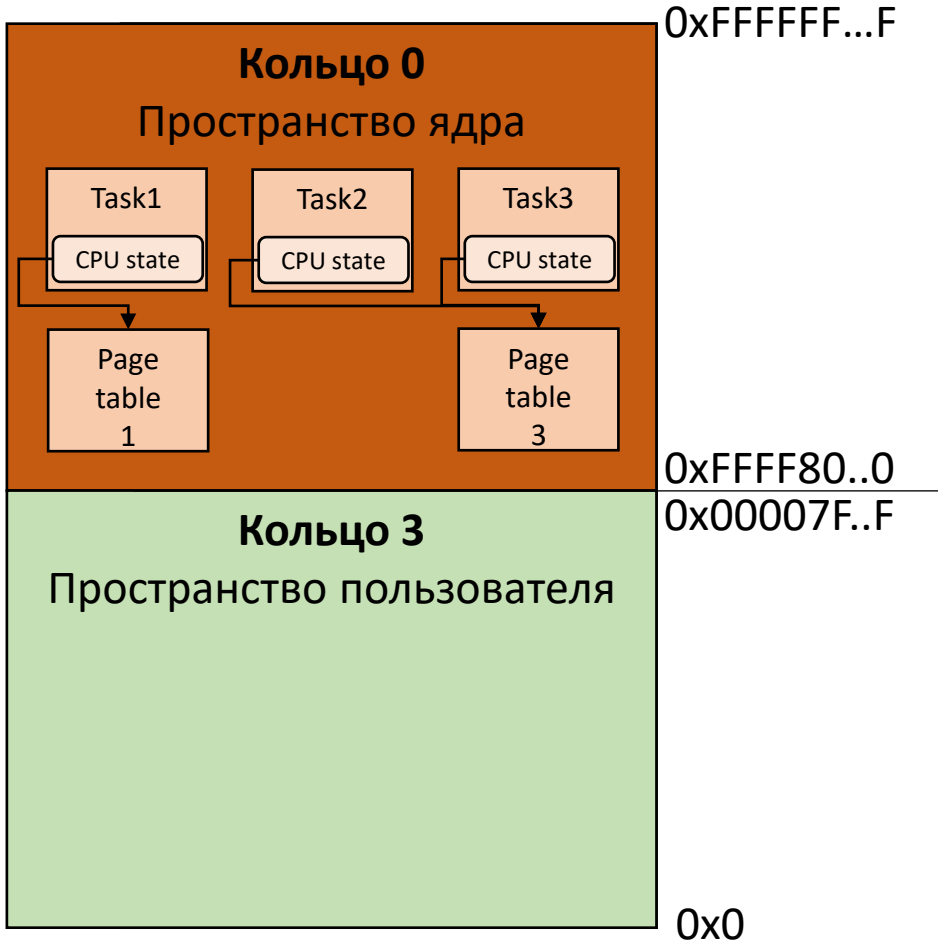
Отладка

Представление потоков в ядре

Каждому потоку в пространстве ядра соответствуют структура данных, которая хранит информацию о потоке

При переключении между потоками ядро ОС сохраняет состояние ЦП (т.е. значения всех регистров) в специальную область в данной структуре, выбирает поток для продолжения выполнения и загружает сохраненное состояние ЦП из структуры.

Так как при этом автоматически устанавливается значение регистра CR3, происходит переключение на другую таблицу страниц. Механизм подкачки автоматически загрузит страницы с кодом и данными программы.



Потоки в ядре Linux

Поток в ядре Linux является основной единицей планировки выполнения.

Каждый поток имеет собственный уникальный в пределах системы идентификатор, который можно получить вызовом `pid_t gettid(void).`

Процесс в терминологии ядра называется группой потоков. Идентификатор процесса = идентификатор группы = идентификатор первого потока группы.

Предел RLIMIT_NPROC из POSIX трактуется ядром Linux, как предел количества потоков. Кроме того, общесистемный предел также устанавливается на количество потоков, а не процессов (файл */proc/sys/kernel/threads-max*).

См.также: tgkill(), rt_sigqueueinfo()

Отладчики

Отладчик – программа, предназначенная для поиска ошибок в программах.

Отладчики используют *предоставляемые ОС* средства для контроля отлаживаемой программы.

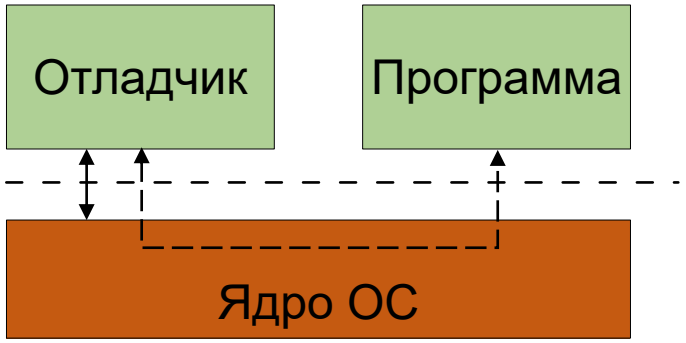
Как минимум, отладчики предоставляют функционал просмотра памяти целевой программы и контроля выполнения программы путем создания точек останова.

Точка останова (breakpoint, сленг. “бряк”) – место в коде программы, по достижении которого выполнение программы прерывается.

Отладчики и ОС

Обычно запущенные программы изолированы друг от друга. Однако ОС предоставляет системные вызовы и/или иные средства, которые *позволяют нарушить изоляцию*. Обычно, есть ограничения на использование данных средств (как минимум, пользователь не должен иметь возможность вклиниться в программу другого пользователя).

Отладчики активно эксплуатируют данные средства.

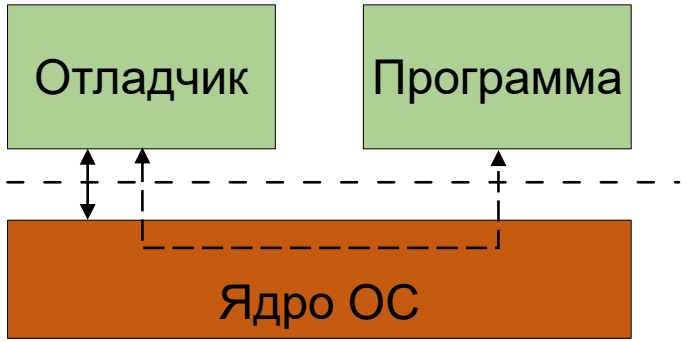


Отладчики и ОС

В случае возникновения аппаратных исключений ОС «преобразует» в ***сигнал***, который отправляется программе. Если программа не готова к пришедшему сигналу, она обычно завершается с ошибкой.

Отладчики «договариваются» с ОС о том, что сигнал сначала отправляется отладчику, который принимает решение о дальнейших действиях. Программа при этом просто приостанавливается.

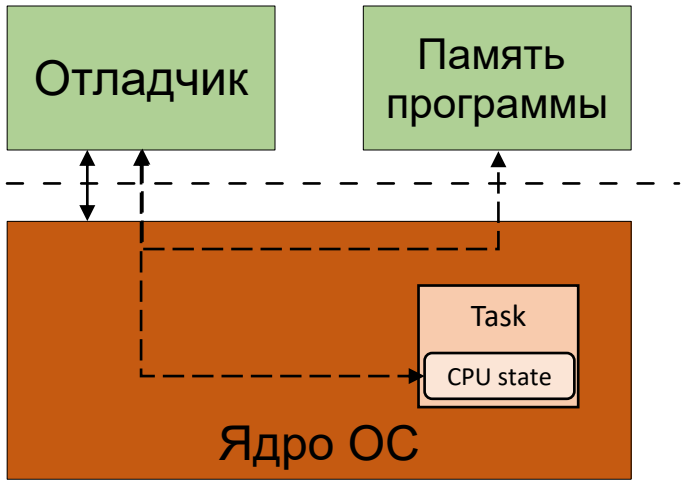
Кроме того, отладчик «договаривается» с ОС о доступе в адресное пространство программы.



Отладчики и ОС

Отладчик имеет доступ:

1. К адресному пространству процесса для чтения/изменения данных и анализа исполняемого кода.
2. К сохраненному состоянию программы – для определения текущей точки выполнения (путем анализа RIP), чтения/изменения регистров общего назначения и регистра флагов (в частности, флага TF, см. далее).



Аппаратные точки останова

Аппаратные точки останова выставляются в специальных отладочных регистрах (регистры DR0-3, DR6-7 на x86). Отладочные регистры позволяют установить 3 типа точек останова:

1. На исполнение кода
2. На запись
3. На чтение или запись

При срабатывании точки останова генерируется аппаратное исключение 1 (#DB).

Для точек останова типа 1 исключение генерируется *до* того, как целевая инструкция будет выполнена (условие RIP==<address>).

Для точек останова 2 и 3 исключение генерируется *после* выполнения инструкции, выполнившей чтение или запись.

Установка аппаратных точек останова требует изменения сохраненного состояния ЦП.

Программные точки останова

Отладочные регистры позволяют создать только 4 точки останова. Поэтому для создания обычных точек останова (на выполнение) используется иная техника.

При создании точки останова по заданному адресу отладчик:

1. читает 1 байт по заданному адресу и сохраняет его;
2. подменяет байт на значение 0xCC (инструкция INT3).

Когда исполнение доходит до точки останова, инструкция INT3 вызывает исключение 3 (#BP). Управление передается отладчику. Отладчик:

1. заменяет INT3 на исходное значение;
2. устанавливает флаг RFLAGS.TF в сохраненном состоянии программы;
3. передает выполнение программе.

Установка программной точки останова требует записи в сегмент кода программы.

Пошаговое выполнение

Флаг RFLAGS.TF включает режим пошагового выполнения.

При RFLAGS.TF=1 исключение #DB генерируется после выполнения *каждой инструкции*.

Обычно отладчик устанавливает этот флаг после срабатывания точки останова и снимает данный флаг, когда программист решает продолжить выполнение (Continue).

Установка RFLAGS.TF требует изменения сохраненного состояния ЦП.

Вызов ptrace

Для осуществления *трассировки* потока используется вызов ptrace():

```
long ptrace(int request, pid_t pid, ??? arg1, ??? arg2);
```

Параметры:

`request` – тип запроса;
`pid` – идентификатор потока;
`arg1, arg2` – трактовка зависит от типа запроса.

Возможности трассировки

Трассирующий процесс может:

- перехватывать сигналы, отправляемые трассируемому процессу;
- останавливать/возобновлять трассируемого;
- изменять сохраненное состояние ЦП;
- изменять данные и код трассируемого процесса;
- отслеживать системные вызовы, выполняемые трассируемым процессом;
- и др.

Активация трассировки

Если в качестве трассировщика выступает поток из процесса-родителя, до в целевом потоке достаточно вызвать `ptrace(PTRACE_TRACEME)`.

Трассировщик может присоединиться к потоку недочернего процесса вызовом `ptrace(PTRACE_ATTACH,<target_thread_id>,NULL,NULL)` или `ptrace(PTRACE_SEIZE,<target_thread_id>,NULL,NULL)`.

Возможность подсоединения трассировщика зависит от конфигурации системы (обычно процесс-родитель имеет право трассировать ребенка).

Отличие `PTRACE_ATTACH` от `PTRACE_SEIZE` в том, что `PTRACE_ATTACH` останавливает трассируемый процесс, в то время как `PTRACE_SEIZE` просто присоединяет процесс трассировщик.

Трассировка и waitpid()

Трассировщик отслеживает состояние трассируемого с помощью вызова `pid_t waitpid(pid_t pid, int *wstatus, int options)`.

В обычных условиях вызов `waitpid()` позволяет процессу-родителю отслеживать состояние процесса-потомка.

При активации трассировки вызов `waitpid()` приобретает иное значение – он позволяет потоку-трассировщику отслеживать изменение состояния трассируемого потока. Как следствие, первым аргументом вызова должен передаваться идентификатор потока.

Стоит напомнить, что waitpid() позволяет отслеживать не только уничтожение, но и остановку/возобновление.

Остановка и возобновление потока

Для остановки трассируемого потока достаточно отправить ему некоторый сигнал, который затем будет перехвачен трассировщиком(см.след. слайд).

Для возобновления работы потока исползуется
`ptrace(PTRACE_CONT, <tracee_id>,NULL,  <signo>)`.

При возобновлении потока последний аргумент интерпретируется, как номер сигнала, который нужно доставить трассируемому потоку при пробуждении.

Если signo==0, то никакой сигнал не доставляется.

Перехват сигналов (пример 1)

При получении сигнала трассируемый поток приостанавливается, но не получает сигнал.

Трассировщик получает уведомление об остановке потока, и дальше может:

- «поглотить» сигнал и возобновить поток без получения сигнала;
- разрешить получение отправленного сигнала;
- подменить номер сигнала;
- подменить информацию, пересылаемую с сигналом (PTRACE_SETSIGINFO).

Номер исходного сигнала можно получить, анализируя значение `wstatus`, возвращаемое `waitpid()`.

Чтение и запись состояния ЦП (пример 2)

Состояние ЦП является частью структуры user из заголовочного файла <sys/user.h>, вид которой зависит от архитектуры машины. Каждому потоку соответствует 1 такая структура в памяти ядра.

Чтение машинного слова (4 или 8 байт, в зависимости от разрядности) в пределах данной структуры производится через

`ptrace(PTRACE_PEEKUSER, <tracee_id>, <offset>, NULL).`

Чтение производится по смещению <offset>.Прочитанные данные возвращаются, как результат вызова.

Запись машинного слова производится через

`ptrace(PTRACE_PEEKUSER, <tracee_id>, <offset>, <new_value>).`

см.также: `PTRACE_GETREGS/PTRACE_SETREGS`

Чтение и запись кода и данных (пример 3)

Чтение машинного слова (4 или 8 байт, в зависимости от разрядности) из адресного пространства трассируемого потока производится через

`ptrace(PTRACE_PEEKDATA, <tracee_id>, <address>, NULL).`

Чтение производится по адресу <address>.Прочитанные данные возвращаются, как результат вызова.

Запись машинного слова производится через

`ptrace(PTRACE_PEEKUSER, <tracee_id>, <address>, <new_value>).`

Адреса в обоих случаях берутся из адресного пространства трассируемого процесса.

Карту памяти трассируемого процесса можно прочитать из */proc/<PID>/maps*

Отслеживание системных вызовов

Для отслеживания системных вызовов трассировщик должен (после остановки) возобновить трассируемого через

```
ptrace(PTRACE_SYSCALL, <tracee_id>,NULL,  <signo>)
```

В последний аргумент – номер сигнала, который будет доставлен потоку при возобновлении.

Если поток был возобновлен таким образом, то трассируемый поток будет остановлен при *следующем* входе в системный вызов или возврате из системного вызова. Используя доступ к состоянию ЦП и адресного пространства, трассировщик может проанализировать аргументы вызова и его результат.

Детектирование отладчика (пример 4)

Обычно отладчики выполняют 3 базовых действия:

1. вызывают fork()
2. в ребенке – вызывают ptrace(PTRACE_TRACEME), kill(getpid(), SIGSTOP) и execve() для запуска целевой программы.
3. в родителе – ptrace(PTRACE_CONT).

Поток может вызвать ptrace(PTRACE_TRACEME) только 1 раз.

Как следствие, если мы в программе вызываем ptrace(PTRACE_TRACEME), и вызов проваливается – значит, скорее всего отладчик присоединен.