Задание 1. Введение в Xv6

Полезная литература

Более подробно разобраться с xv6 поможет книга R. Cox, F. Kaashoek, R. Morris «xv6: a simple, Unix-like teaching operating system» (на английском).

Также, возможно, вам понадобится информация об архитектуре `RISC-V`, на которой запускается `Xv6`. Спецификация ISA доступна на официальном сайте.

Часть 1. Pingpong

Прежде чем перейти к основной части курса, познакомимся подробнее с `Xv6` и её системными вызовами. Мы попробуем написать немного user-space кода.

Научитесь обмениваться данными между процессами с помощью специальных FIFO-каналов — Unix pipes.

Реализуйте программу user/pingpong.c, которая должна:

1. Создать пайп.
2. Создать дочерний процесс.
3. Отправить сообщение ping из родительского процесса в дочерний.
4. Прочитать их в дочернем процессе, вывести `<child pid>: got <message>` и отправить сообщение `pong` в ответ.
5. Прочитать ответ в родительском процессе, вывести `<parent pid>: got <message>`.

Советы для выполнения задания:

- Вам понадобится несколько системных вызовов — `pipe`, `fork`, `read`, `write`, `getpid`. Воспользуйтесь утилитой `man`, чтобы узнать, что делают эти вызовы и как ими пользоваться — поведение в `Xv6` не особо отличается от других Unix-подобных операционных систем.
- Вместо привычных вам `stdlib.h` и `stdio.h` доступна местная стандартная библиотека — user/ulib.c, а также user/printf.c и user/umalloc.c. Посмотрите на другие программы в директории user/, чтобы понять, как ей пользоваться.
- Добавьте программу в `UPROGS` в Makefile, чтобы она скомпилировалась.
- В программах для `Xv6` обязательно нужно вызывать `exit(0)` для выхода.

Часть 2. Dump

В прошлой части ЛР мы использовали системные вызовы, например, `pipe` и `fork`.

Задача системных вызовов — дать программам из user-space возможность выполнять привилегированные команды.

Реализуем новый системный вызов `dump`. Он будет выводить на экран состояние регистров `s2–s12` вызывающего процесса.

Чтобы системный вызов был доступен из user-space, добавим в файл user/user.h объявление функции `dump`, как это сделано для других системных вызовов. В файл user/usys.pl добавьте строку `entry("dump")` — он отвечает за генерацию ассемблерных инструкций для совершения системного вызова.

Теперь реализуем сам системный вызов. Для этого добавьте функцию `dump` в файл kernel/proc.c. Текущий процесс можно получить с помощью функции `myproc`. Структура `proc` содержит поле `trapframe`, в котором и находятся значения всех регистров. Избегайте лишней копипасты при выводе регистров. Все регистры в `Xv6` 64-битные, однако в рамках данного задания для каждого регистра вам нужно вывести лишь младшую 32-битную часть.

Наконец, отредактируйте файлы kernel/syscall.h, kernel/sysproc.c и kernel/syscall.c так, чтобы появилась возможность вызвать `dump` из user-space. Посмотрите, как реализованы другие вызовы, и сделайте аналогично.

Осталось собрать `Xv6`. Запустите утилиту user/dumptests.c и сравните фактические значения регистров и результат вашего системного вызова.

- Функцию dump нужно также определить заголовочном файле в kernel/defs.h.
- Системный вызов должен возвращать 0 при успешном завершении, и код ошибки в остальных случаях. Наш системный вызов всегда завершается успешно.

- Если при запуске `dumptests` выводится сообщение о том, что системный вызов `dump` не найден, то попробуйте пересобрать `Xv6` с нуля.

Часть 3*. Dump2

Мы бы могли использовать системный вызов `dump`, чтобы написать собственный отладчик. Однако, у него есть два недостатка. Во-первых, он выводит значение регистров на экран, и мы не можем обработать эти значения в user-space. Во-вторых, он позволяет узнать значения регистров только у текущего процесса, что делает невозможным отладку другого процесса. Напишем ещё один системный вызов, чтобы исправить эти недостатки — `dump2`.

У этого вызова будет три аргумента:

1. `int pid` — номер процесса, для которого запрашивается значение регистра
2. `int register_num` — номер регистра, число от 2 до 11
3. `uint64 *return_value` — адрес, по которому необходимо вернуть значение

Обратите внимание, что в целях безопасности регистры процесса может смотреть только сам процесс и его родитель.

В этом системном вызове, в отличие от `dump`, вам понадобится корректно обрабатывать и возвращать ошибки:

- Верните -1, если вызвавший процесс не имеет прав смотреть требуемый регистр
- Верните -2, если процесса с таким идентификатором не существует
- Верните -3, если передан некорректный номер регистра
- Верните -4, если не удалось записать данные по переданному адресу

Полезные советы:

- Аргументы в системные вызовы передаются немного иначе, нежели в обычные функции. Посмотрите на другие системные вызовы, чтобы понять, как получать аргументы из user-space.
- Вы не можете записать данные по адресу `*return_value` — это виртуальный адрес в user-space, и использовать его в kernel-space невозможно. Вам поможет функция `copyout`.

Запустите user/dump2tests.c. Проверка происходит автоматически.

Отправка

В своем форке репозитория поместите изменения в ветку с названием в точности `lab-1` и сделайте MR в ветку `main`.