На первой неделе предлагается ознакомиться с понятием системного вызова. Что это такое, зачем он нужен и как его применяют пользовательские приложения.

Почитайте тут:
https://en.wikipedia.org/wiki/System_call
https://habr.com/ru/articles/347596/

Еще я пришлю пару книг. Первая - «Ядро Linux», в ней есть глава 10, посвященная системным вызовам, но она довольно сложная. Тем не менее рекомендую ее прочитать, пусть многое останется непонятным, но, думаю, какие-то моменты наоборот добавят понимания. Вторая - «Программирование для Linux. Профессиональный подход», я ее читал уже довольно давно, но она мне запомнилась как более понятная и прикладная. В ней есть глава про strace и еще кое-что интересное.

Все дальнейшие действия предполагают работу в командной строке какой-либо операционной системы семейства linux. Не надо искать экстравагантные версии, проще всего взять более-менее современный debian или ubuntu, естественно 64-битный. Кроме того предполагается владение одним из языков программирования C или C++ и умение компилировать исходные коды, написанные на этих языках, при помощи gcc/g++. Предполагается, что все дальнейшие запуски linux-утилит и скомпилированных программ производятся из командной строки, так что некоторое владение ею также потребуется.

Дальше необходимо познакомиться с утилитой strace. Можно либо в командной строке набрать команду:
man strace
либо сделать то же самое в поисковой строке браузера и перейти по ссылке.
В обоих случаях появится информация об этой утилите и о том, как ей пользоваться. Также можно, как я уже писал выше, обратиться к книге «Программирование для Linux. Профессиональный подход».

На данный момент в linux'e реализовано несколько сотен системных вызовов (в районе 500 штук), естественно все разбирать не будем. Поэтому предлагаю поступить следующим образом. Для начала написать на C или C++ несколько простейших программ:
- пустой main;
- hello world;
- разбор параметров бинарника (для простоты можно поддержать опции —help и —version, а все остальные считать некорректными);
- чтение данных из файла;
- выделение под новый объект динамической памяти (не на стеке) через malloc или new.

Каждую из этих задач скомпилировать в четырех вариантах: слинковать статически и динамически, а также собрать в 32-битном и 64-битном режимах. В этом, я надеюсь, моя помощь не понадобится, в крайнем случае считаю, что это может быть неким самостоятельным занятием. Далее запустить каждый из полученных исполняемых файлов через утилиту strace и постараться выделить в общем потоке ее вывода только ту часть, которая относится непосредственно к исходному коду (не коду стандартных библиотек). Если с этим будут проблемы, могу дать подсказку. Одна из предложенных задач должна показаться «лишней», какая?

Получить информацию по системным вызовам можно также при помощи команды man, только на этот раз ее надо набирать следующим образом:
man 2 <имя системного вызова>

например:
man 2 exit

Если с этим все понятно, то предлагаю разобрать программы посложнее, использующие библиотеку libpthread. Для этого в книге «Программирование для Linux» необходимо почитать раздел 4.1 и посмотреть на выдачу strace (разобраться в ней) для всех приведенных там задач.

Для закрепления материала предлагаю выполнить следующее задание. В man-данных на системные вызовы обычно есть раздел про коды ошибок. Предлагаю написать несколько программ, которые бы отправляли в ядро «кривые» запросы. Надо понимать, что обычно программист использует в своих кодах библиотечные функции, которые уже сами реализуют набор необходимых запросов в ядро с привлечением системных вызовов. Однако, при помощи функции syscall (см. man 2 syscall) можно попытаться выполнить системный вызов напрямую. В man-странице есть описание интерфейса syscall - о том на каких регистрах хранится номер системного вызова и его аргументы, на данный момент эта информация не понадобится, но для будущих работ прошу обратить внимание. Для написания таких программ понадобится знать номер системного вызова, который планируется «сломать». В конце man-страницы для syscall есть короткий пример, в частности, в нем фигурируют системные вызовы, заданные при помощи предопределенных значений SYS_gettid и SYS_tgkill. Чтобы найти значения для других системных вызовов надо найти определения этих символов в include-файлах, сделать это можно примерно так:

find /usr/include -name syscall.h -exec grep -H SYS_gettid {} \;

Что касатеся подключения хедеров для своих тестов, то можно дублировать инклуды из примера на man-странице syscall.

Попробуйте «сломать» системные вызовы open, mmap и link, чтобы они возвращали разные коды ошибок, а программа об это сигнализировала при помощи сообщения об ошибке, раскрывающего ее суть.

Еще раз напоминаю, что данный курс во многом является экспериментальным. Его никто прежде не проходил, поэтому допускаю, что у стажера могут возникнуть некоторые проблемы с пониманием материала. В таком случае прошу задавать мне вопросы, на которые я постараюсь ответить. Не исключаю и другой крайности, когда стажеру все это уже известно и задания будут казаться простыми. В таком случае также прошу дать об этом знать. Сразу отмечу, в выходные прошу меня по теме стажировки не отвлекать, на все вопросы буду отвечать только в будние дни.

По ходу выполнения предоставленных заданий хотел бы получать отклики о пройденных этапах, в идеале — в конце дня выслать короткий отчет о проделанной работе. Так мне была бы более понятна сложность темы для стажера.