1. Написать свой **stat**.

   Напечатать в терминале фразу **stat *имя файла***, посмотреть что она выдает. И написать программу по образу и подобию. Дабы это было легче делать, то стоит прочесть такую замечательную штуку как **man**. В командной строке необходимо набрать *man 2 lstat* и наслаждатьс чтением, а так же почти что написанным за вас кодом в этом самом **man**. Так же для сдачи задачи необходимо понимать чем отличаются следующие команды **lstat, statx, fstat, fstatat**.

2. (a) Записать в указанный файл, содержимое указанное в строке. Программа подразумевает под собой использование вызова **open, write, close**.

   (b) Понять что уществует еще и такой вызов как **dprintf**. Проделать тоже самое, что было проделано в 2.a.

3. (a) Теперь пора переходить к большим масштабам и копировать не просто содержимое какой-то строчки, а целого файла. Для этого знакомимся с таким системным вызовом как **read** и сочетаем его с системным вызовом **write**.

   (b) Познакомится с такими штуками как **pread, pwrite** и сделать все тоже самое, что в 3.a.

   (c) Научится копировать не только обычные файлы, но и прочие штучки, такие как FIFO, символьные ссылки, блочные и символьные устройуства. Для этого используем такие штуки как **mknod, mkfifo, readlink, symlink.**

4. Учимся изменять атрибуты у наших скопированных файлов и не только. Для этого используем **fchmod, futimens**.

5. Копируем UID и GID для этого используем **fchown**

6. (a) Читаем содержимое дирректории и вывожим на экран на подобии ls -la **opendir, readdir, closedir**

   (b) Аналогично 6.a только для каталога, указанного пользователем. Используем **opendir, dirfd, fstatat, readdir, closedir**

   (c) Решить задачу 6.1, но вместо всего прекрасного использовать getdents.

   (d) Рекурсивно реализовать 6.2.

7. (a) Копировать директорию. Используем **openat, mkdirat**.

   (b) Рекурсивно копировать директорию.

8. Вывести физическое размещение файла на диске. С помощью strace посмотреть какие системные вызовы дергают уилиты hdparm–fibmap **filefrag hdparm–fibmap**.

9. (a) Вывести занятое, свободное и доступное место пространство для файловой системы на которой расположен указанный файл или каталог. Используем **statvfs, statfs**.

(b) **quota, quotactl**

10. Напечатать информацию о появлении новых файлов в указанном каталоге. Отслеживаем с помощью **inotify**. Код программы есть в man.

11. Файловые блокировки. О том, что если несколько программ работают с одним файлом, то им нужно синхронизировать свои действия, иначе ничего хорошего не получится. Если программы будут действовать одновременно, то содержимое файла превратится в кашу.

    Соответственно требуется написать программу, которая бы подсчитывала сколько раз она запускалась. В другом файле **counter.txt** отсчитывать количесво запусков (считая текущий) и коректно отрабатывать параллельные запуски.

    Используется **flock, lockf, fcntl-lock**.

12. $*$ Вывести информацию про текущий процесс (про запущенный экземпляр этой программы) id выводить по порядку как в *credentials*(7) — иерархия процессов, группы (по кучкам) *getgroups*(2). Читаем **edentials, sched, capabilities**. Используем **prlimit, getlimit**.

13. (a) Написать программу, которая порождает дочерний процесс и ждет его завершения, затем выводит информацию о завершении дочернего процесса. Используем **wait, waitpid, waitid**. Пример в man 2 *waitpid*

    (b) Программа порождает дочерний процесс, он дожидается завершения родительского процесса. Мониторить *parentpid*, если он изменился, то родитель умер.

14. (a) **dup 2**

    (b) **execve**

    (c) **pipe** Вывести то же самое, что $last|wc{-}l$ (на основе 14.1) реализовать использование других программ из своей соединить *stdout* одной программы с *stdin* другой программы(команды пишут на вход друг другу)

    (d) Написать программу, которая в дочернем процессе запускает *gzip*, работая с ним через два пайпа; померять скорость, с которой *gzip* сжимает случайные данные **poll/select**.

15. (a) Написать программу, которая не завершается используя сигналы **signal, sigaction**

    (b) Модифицировать inotify так, чтоб он корректно завершался 10 + termination

    (c) $*$ **signalfd, sigqueue**

16. Создать очередь сообщений (*mq_open*), посмотреть на ее параметры (*mq_getattr*), в конце почистить (*mq_close* + *mq_unlink*) **mq_open, mq_getattr, mq_close, mq_unlink**

17. Написать две программы

    одна всегда запущена(сервер): создает очередь сообщений, в бесконечном цикле читает ($mq\_receive$) все сообщения(и время, когда оно прилетело) печатает (нет , печатать printf ом аккуратно); когда приходит сигнал, удаляет очередь вторая (клиент): открывает очередь посылает туда сообщение закрывает очередь !сначала закрыть дескриптор очереди, только потом в цикле обрабатывать все сообщения**mk_send, mk_receive**

18. Использовать команды **dlopen, dlsym, dlclose** для написания интегрирования.

19. (a) **pthread_mutex_t**

    (b) **sem_init, sem_wait, sem_post, sem_destroy**

    (c) **sem_open, sem_close, sem_unlink**

    (d) **atomit_fetch_add, atomic_load**

    (e) **map-reduce approach**

20. shm_server :

    создает область общей памяти **clock (shm_open** + **ftruncate)**, отображает ее в свое адресное пространство $mmap$, инициализирует область **sem_init** раз в секунду пишет в нее строку с временем **localtime**+**strftime**, по $SIGINT/SIGTERM$ прекращает цикл, и удаляет область общей памяти shm_unlink

    shm_client :

    открывает $/clock$ на чтение shm_open и отображает $mmap$, в цикле раз в секунду печатает строку из этой области, по $SIGINT/SIGTERM$ прекращает цикл.

    Не забываем про синхронизацию! (sem_init в сервере + sem_wait/sem_post в обоих).

21. UDP-сервер времени. На пакет с некоторым запросом отвечает пакетом со строкой с текущим временем.

22. TCP-сервер времени. Принимает соединения, раз в секунду в каждое живое соединение отправляет строку с текущим временем, формат:

    Плохой/злонамеренный клиент не должен иметь возможности обрушить или завесить сервер.