OS Lab 8

Status	approved
checkbox	☑
class	OS
due date	@Mar 17, 2021

Task

💡 8. Защищенный текстовый редактор

Напишите программу, которая захватывает весь файл перед вызовом редактора. Это защитит файл с правами доступа группы на изменение. После того как программа захватит файл, вызовите свой любимый редактор с помощью библиотечной функции `system(3)`. Попытайтесь в вашей программе использовать допустимое захватывание. Когда ваша программа захватит файл, попросите одногруппника исполнить вашу программу. Объясните результат. Потом попытайтесь использовать обязательное захватывание. Объясните результат.

```
#include <sys/types.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>

int main(int argc, char *argv[])
{
    struct flock lock;
    int fd;
    char command[100];

    if ((fd = open(argv[1], O_RDWR)) == -1) {
        perror(argv[1]);
        exit(1);
    }
    lock.l_type = F_WRLCK;
    lock.l_whence = SEEK_SET;
    lock.l_start = 0;
    lock.l_len = 0;
    if (fcntl(fd, F_SETLK, &lock) == -1) {
  // In the past, the variable errno was set to EACCES rather than
EAGAIN when
       // a section of a file is already locked by another proces
s.  Therefore,
       // portable application programs should expect and test fo
r either value
        if ((errno == EACCES) || (errno == EAGAIN)) {
            printf("%s busy -- try later\n", argv[1]);
            exit(2);
        }
        perror(argv[1]);
        exit(3);
    }
    sprintf(command, "nano %s\n", argv[1]);
    system(command);

    lock.l_type = F_UNLCK;    /* unlock file */
    fcntl(fd, F_SETLK, &lock);
```

Notes

В системах семейства Unix, предполагается, что программа с самого начала разрабатывается в расчёте на многозадачную среду. **Поэтому файл при открытии автоматически не блокируется.** Программист, который считает необходимым использование блокировок при работе с файлом, должен устанавливать эти блокировки явным образом.

Запись (record) - это последовательный набор байтов в файле. У записи есть начальная позиция в файле и длина, отсчитываемая от этой позиции. Начало и длина указываются с точностью до байта.

Захват (блокировка) записей в ОС UNIX - это базовое средство синхронизации процессов, осуществляющих доступ к одной и той же записи в файле, а не средство обеспечения безопасности. Захват записи осуществляется системным вызовом `fcntl(2).`

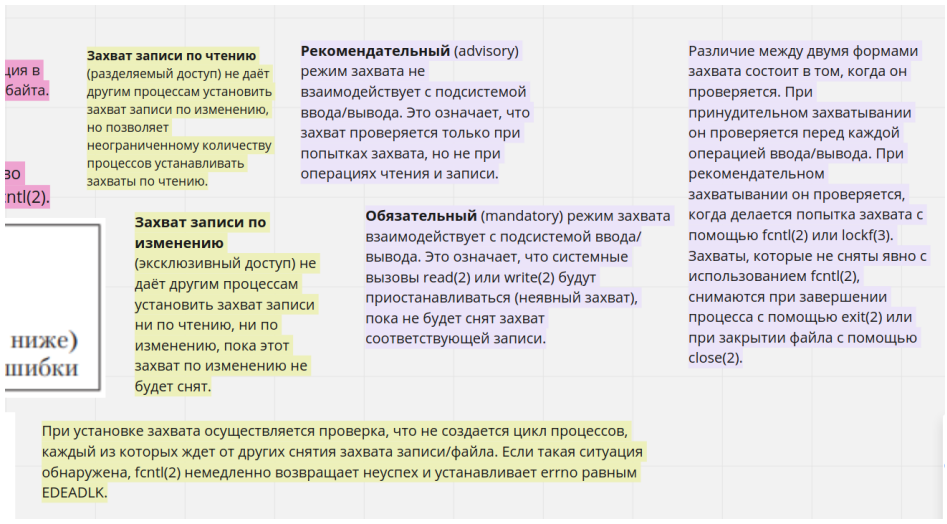
ЧТО ТАКОЕ ЗАХВАТ ЗАПИСИ И ФАЙЛА?

- Запись - это последовательный набор байтов в файле
- Захват записи по чтению (разделяемый доступ) не дает другим процессам установить захват записи по изменению
- Захват записи по изменению (эксклюзивный доступ) не дает другим процессам установить захват записи по чтению/изменению, пока этот захват по изменению не будет снят
- Рекомендательный захват (advisory lock): захват записи проверяется только перед попыткой установки захвата
- Принудительный захват (mandatory lock): захват записи проверяется ядром перед выполнением операций ввода/вывода

```
    close(fd);
    return 0;
}
```

struct flock

```
typedef struct flock
{
  short l_type;
  /* F_RDLCK, F_WRLCK, F_UNLCK */
  short l_whence;
  /* SEEK_SET, SEEK_CUR, SEEK_END */
  off_t l_start;
  off_t l_len;
  /* len == 0 means until end of file */
  long l_sysid;
  pid_t l_pid;
  long pad[4]; /* reserve area */
} flock_t;
```



The “let’s favor writers but be fair to readers” policy

In Solaris 7, RW locks are implemented as a single-word data structure in the kernel, either 32 bits or 64 bits wide, depending on the data model of the running kernel, as depicted in Figure 3.9.

OWNER (writer) or COUNT OF READER THREADS (reader)	wrlock	wrwant	wait
63 - 3 (LP64), or 31 - 3 (ILP32)	2	1	0

The number of upper bits is determined by the data model of the booted kernel (64-bit or 32-bit) in Solaris 7.

Figure 3.9 Solaris 7 Reader/Writer Lock

The Solaris 7 RW lock defines bit 0, the `wait` bit, set to signify that threads are waiting for the lock. The `wrwant` bit (write wanted, bit 1) indicates that at least one thread is waiting for a write lock. Bit 2, `wrlock`, is the actual write lock, and it determines the meaning of the high-order bits. If the write lock is held (bit 2 set), then the upper bits contain a pointer to the kernel thread holding the write lock. If bit 2 is clear, then the upper bits contain a count of the number of threads holding the lock as a read lock.

For `rw_exit()`, which is called by the lock holder when it is ready to release the lock, the simple case is that there are no waiters. In this case, the `wrlock` bit is cleared if the holder was a writer, or the hold count field is decremented to reflect one less reader. The more complex case of the system having waiters when the lock is released is dealt with in the following manner.

📢 **advisory lock (default) работает как светофор** — он показывает, что вам не следует идти через улицу, но не препятствует этому физически.
Он влияет на другие блокировки, но не влияет на read/write
Если файл в этом режиме, чтобы прочитать его, на него блокировку ставить не надо.
А если у вас или у кого-то другого блокировка стоит, то вы всё равно можете прочитать. То же справедливо и для операции записи.

mandatory lock — работает на уровне операций read/write. То есть, если на файле стоит mandatory lock и у вас нет подходящей блокировки то прочитать вы не сможете физически. Просаживает производительность из-за проверок. Не поддерживается NFS. Является атрибутом файла. (Gрименение принудительных блокировок к отдельным файлам разрешается включением бита set-group-ID и выключением group-execute. По-скольку установка бита set-group-ID теряет смысл при сброшенном бите group-execute, разработчики выбрали именно такой способ указать, что файл должен подвергаться принудительной, а не рекомендательной блокировке). Файл с mandatory lock нельзя замеммапать.

Mandatory/advisory

- По умолчанию захват происходит в advisory режиме
- Mandatory locking включается атрибутом файла: `bash>chmod +l file`
- Не работает на NFS (ваши домашние каталоги подключаются по NFS).
- Используйте /tmp

Установка захвата

```
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>

int fcntl(int fildes, int cmd,
          struct flock *arg);
```

значения cmd
F_GETLK получить информацию о захвате записи
F_SETLK установить захват записи
F_SETLKW установить захват записи (с блокировкой)

- The kernel does a direct transfer of ownership of the lock to one or more of the threads waiting for the lock when the lock is released, either to the next writer or to a group of readers if more than one reader is blocking and no writers are blocking.

  This situation is very different from the case of the mutex implementation, where the wakeup is issued and a thread must obtain lock ownership in the usual fashion. Here, a thread or threads wake up owning the lock they were blocking on.

  The algorithm used to figure out who gets the lock next addresses several requirements that provide for generally balanced system performance. The kernel needs to minimize the possibility of starvation (a thread never getting the resource it needs to continue executing) while allowing writers to take precedence whenever possible.
- `rw_exit_wakeup()` retests for the simple case and drops the lock if there are no waiters (clear `wrlock` or decrement the hold count).
- When waiters are present, the code grabs the turnstile (sleep queue) associated with the lock and saves the pointer to the kernel thread of the next write waiter that was on the turnstile's sleep queue (if one exists).

  The turnstile sleep queues are organized as a FIFO (First In, First Out) queue, so the queue management (turnstile code) makes sure that the thread that was waiting the longest (the first in) is the thread that is selected as the next writer (first out). Thus, part of the fairness policy we want to enforce is covered.

Reading list