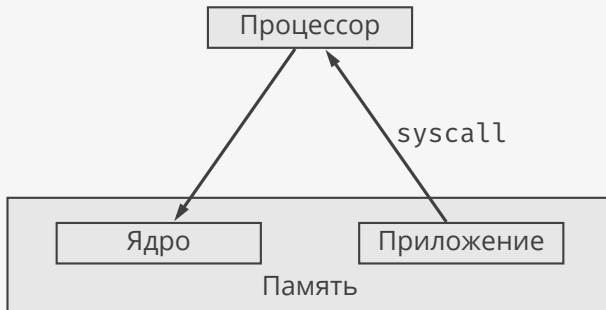


Системное программирование в Linux

Иван Ганкевич

2018

Системный вызов



```

long syscall(long n, ...) {
    va_list ap;           // от 0 до 6
    long a,b,c,d,e,f;    // аргументов
    va_start(ap, n);
    a=va_arg(ap, long);
    ...
    va_end(ap);
    return __syscall_ret(
        __syscall(n,a,b,c,d,e,f)
    );
}

long __syscall_ret(unsigned long r) {
    if (r > -4096UL) { // (-4096,0)
        errno = -r; return -1;
    }
    return r;
}

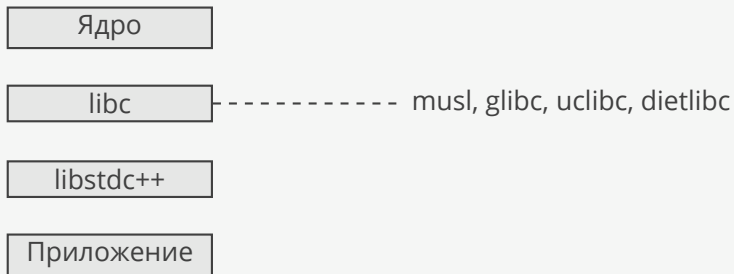
```

```

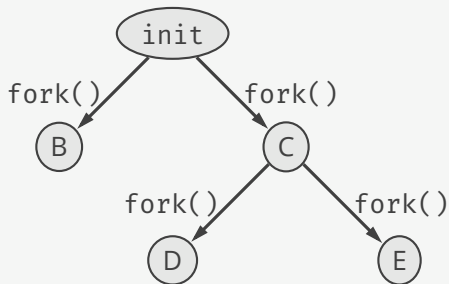
.global __syscall
.hidden __syscall
.type __syscall,@function
__syscall:
    movq %rdi,%rax
    movq %rsi,%rdi
    movq %rdx,%rsi
    movq %rcx,%rdx
    movq %r8,%r10
    movq %r9,%r8
    movq 8(%rsp),%r9
    syscall
    ret

```

- ▶ Каждый системный вызов имеет номер.
- ▶ Способ вызова системной функции у каждого процессора свой.

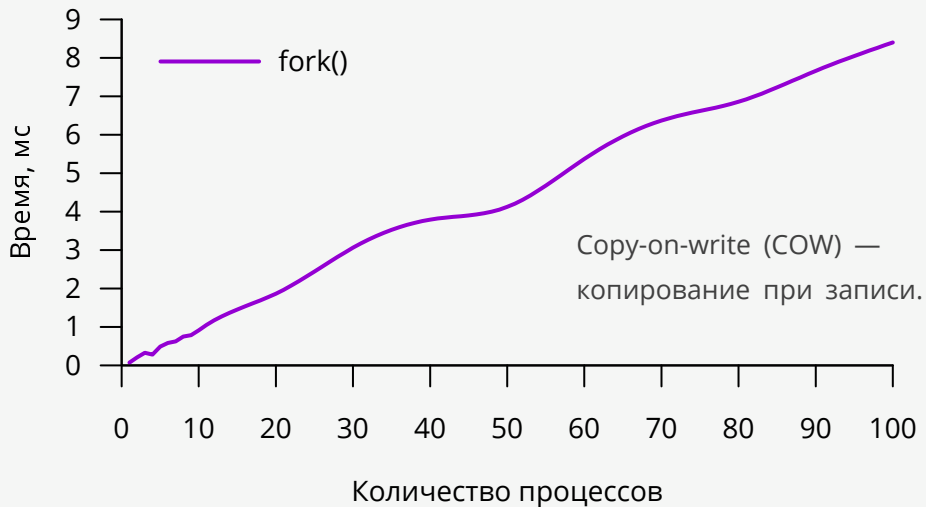


Процессы



```
int pid = fork(); // копирует
if (pid == 0) {   // процесс
    // дочерний процесс
}
// родительски процесс
```

Копирование процессов — это медленно!



Отслеживание системных вызовов

```
$ strace -e openat ./myprog # отследить открытие файла
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libstdc++.so.6", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libm.so.6", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
openat(AT_FDCWD, "/lib64/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
```

Создание процесса

Программа:

```
pid_t pid = fork(); // создание дочернего процесса
if (pid == 0) {
    exit(0);          // завершение дочернего процесса
}
int status = 0;
wait(&status);       // подождать завершения дочернего процесса
```

Вывод strace:

```
clone(
    child_stack=NULL,
    flags=CLONE_CHILD_CLEARTID | CLONE_CHILD_SETTID | SIGCHLD,
    child_tidptr=0x7f25ede00990
) = 7296
```


Потоки

Программа:

```
std::thread t{[] () {}}; // создание дочернего потока  
t.join(); // подождать завершения дочернего потока
```

Вывод strace:

```
clone(  
    child_stack=0x7f901f84cfb0,  
    flags=CLONE_VM | CLONE_FS | CLONE_FILES | CLONE_SIGHAND |  
          CLONE_THREAD | CLONE_SYSVSEM | CLONE_SETTLS |  
          CLONE_PARENT_SETTID | CLONE_CHILD_CLEARTID,  
    parent_tidptr=0x7f901f84d9d0,  
    tls=0x7f901f84d700,  
    child_tidptr=0x7f901f84d9d0  
) = 7345
```

Опция	Назначение
CLONE_VM	общая память
CLONE_FS	общие рабочая директория, корень ФС и маска
CLONE_FILES	общие файловые дескрипторы
CLONE_SIGHAND	общие обработчики сигналов
CLONE_SYSVSEM	общие примитивы синхронизации
CLONE_THREAD	добавление в группу потоков
CLONE_SETTLS	выделение локальной памяти потока

Контейнеры

Программа:

```
size_t stack_size = 1024*10;  
char* child_stack = new char[stack_size];  
void* child_stack_end = child_stack + stack_size;  
pid_t pid = clone(child_main, child_stack_end,  
    CLONE_NEWUTS | CLONE_NEWUSER | SIGCHLD, 0);  
int status = 0;  
wait(&status);  
delete[] child_stack;
```

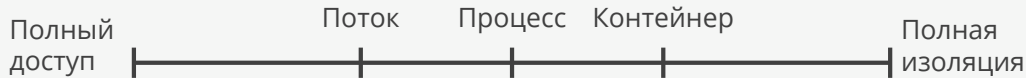
Вывод strace:

```
clone(  
    child_stack=0x7f741fa0c000,  
    flags=CLONE_NEWUTS | CLONE_NEWUSER | SIGCHLD  
) = 7427
```

Опция	Пространство имен
CLONE_NEWNS	точки монтирования
CLONE_NEWPID	процессы
CLONE_NEWUSER	идентификаторы пользователей и групп
CLONE_NEWNET	сетевые устройства
CLONE_NEWUTS	имя хоста и доменное имя машины
CLONE_NEWIPC	примитивы синхронизации
CLONE_NEWCGROUP	контрольные группы

Процессы, потоки и контейнеры

- ▶ Процесс — единица планирования системных ресурсов.
- ▶ Поток — процесс с большим количеством общих с родительским процессом ресурсов.
- ▶ Контейнер — процесс, использующий новые пространства имен.



Запуск программ

```
pid_t pid = fork(); // а если здесь создать поток?  
if (pid == 0) {  
    char* const argv[] = {"ls", "-l", 0}; // аргументы  
    execvp(argv[0], argv); // нулевой аргумент — имя программы  
    exit(0);  
}  
int status = 0;  
wait(&status);
```

Некоторые системные вызовы

```
getpid()      // номер процесса
getppid()     // номер родительского процесса
getuid()      // номер пользователя
getgid()      // номер группы
getenv()      // переменные среды
getcwd()      // рабочая директория
```

Переменные среды

Программа:

```
char** first = environ;  
while (*first) {  
    std::cout << *first << '\n';  
    ++first;  
}
```

Вывод:

```
...  
HOME=/home/myuser  
LANG=ru_RU.utf8  
...
```


ВВОД/ВЫВОД

Программа:

```
int fd = open("myfile", O_CREAT|O_WRONLY|O_TRUNC, 0644);  
const char msg[] = "hello\n";  
write(fd, msg, sizeof(msg));  
close(fd);
```

Содержимое myfile:

```
hello
```

Проверка на ошибки

```
int fd = open("myfile", O_CREAT|O_WRONLY|O_TRUNC, 0644);
if (fd == -1) {
    throw std::system_error(errno, std::generic_category());
}
const char msg[] = "hello\n";
ssize_t nwritten = write(fd, msg, sizeof(msg));
if (nwritten == -1) {
    throw std::system_error(errno, std::generic_category());
}
if (close(fd) == -1) {
    throw std::system_error(errno, std::generic_category());
}
```

Проверка на ошибки

```
#define CHECK(func) \  
    if ((func) == -1) { \  
        throw std::system_error(errno, std::generic_category());  
    }  
  
int fd; ssize_t nwritten;  
CHECK(fd = open("myfile", O_CREAT|O_WRONLY|O_TRUNC, 0644));  
const char msg[] = "hello\n";  
CHECK(nwritten = write(fd, msg, sizeof(msg)));  
CHECK(close(fd));
```

Работа с памятью

```
size_t size = 4096;
void* ptr = mmap(                                // выделить страницы памяти
    nullptr,                                    // адрес
    size,                                       // размер в байтах
    PROT_READ|PROT_WRITE,                      // права доступа
    MAP_PRIVATE|MAP_ANONYMOUS,                // опции
    -1,                                       // файловый дескриптор
    0                                         // отступ внутри файла
);
munmap(ptr, size);                             // освободить страницы памяти
```

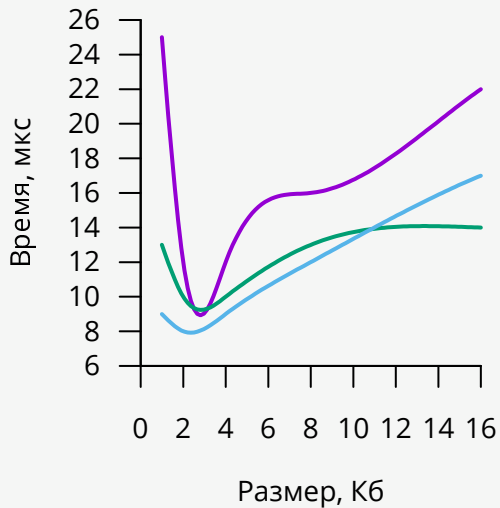
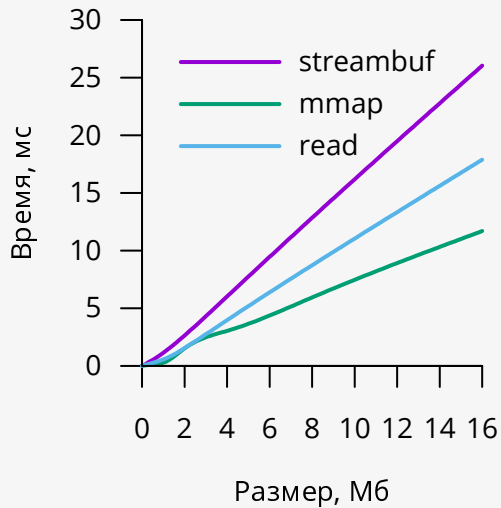
Системный вызов mmap

- ▶ либо выделяет недоступные другим процессам страницы,
- ▶ либо отображает файл на страницы памяти.

```
ptr = mremap(  
    ptr,                // текущий адрес  
    size,               // текущий размер  
    size*2,             // новый размер  
    MREMAP_MAYMOVE      // опции  
);  
size *= 2;  
madvise(ptr, size, MADV_SEQUENTIAL); // управление страницами
```

Опции:

MADV_SEQUENTIAL	последовательный
MADV_RANDOM	произвольный
MADV_WILLNEED	скоро понадобится
MADV_DONTNEED	больше не нужен



- ▶ Объем выделенной памяти кратен размеру страницы (4Кб).
- ▶ При чтении/записи ядро отображает содержимое файлов на страницы памяти, даже если не использовать mmap.
- ▶ Все считанные файлы попадают в кэш.
- ▶ Память можно сделать «видимой» другим процессам.

```
$ free -m
```

	total	used	free	shared	buff/cache	available
Mem:	7973	1669	3860	64	2442	6140
Swap:	7999	55	7944			

Стандарты

Немного статистики:

- ▶ В Linux около 300 системных вызовов.
- ▶ В библиотеке `libc` около 1300 функций.
- ▶ Каждая программа использует хотя бы один системный вызов/функцию.
- ▶ Обратная совместимость соблюдается строго.

Стандарты:

- ▶ System V (SVr4) — устаревший стандарт.
- ▶ Single UNIX Specification (SUS).
- ▶ Portable Operating System Interface (POSIX).

Можно ли запустить Linux-программу на другой ОС?

Совместимость с POSIX:

ОС	POSIX1	POSIX2	Год
Linux	62%	33%	2008
MacOS	37%	46%	2001
Windows	<i>все еще ставятся обновления</i>		

Сложно портировать:

- ▶ OpenGL/OpenCL/CUDA и т.п.

Страницы руководства

Разделы:

1. Исполняемые файлы
2. Системные вызовы.
3. Библиотечные вызовы.
4. Специальные файлы.
5. Файлы конфигурации.
6. Игры.
7. Общие сведения.
8. Команды для администратора.

В терминале:

```
man 1 ls
man 2 mmap
man 3 realpath
man 4 null
man 5 passwd
man 6 rot13
man 7 tcp
man 8 sudo
```

Онлайн:

www.kernel.org/doc/man-pages/

Ссылки

- ▶ [The Cathedral and the Bazaar.](#)
- ▶ [Сравнение различных libc.](#)
- ▶ [Страницы руководства Linux.](#)
- ▶ [Программа для проверки совместимости с POSIX.](#)