

Практикум на ЭВМ

Системные вызовы.

Ввод, вывод, fork, exec.

Баев А.Ж.

Казахстанский филиал МГУ

13 октября 2018

Справочные страницы man

Можно посмотреть информацию о команде shell, системном вызове и прочее.

```
man cd  
man fork  
man man
```

Можно делать приближенный поиск и уточнять раздел поиска

```
man -k read  
man -k . -s 2
```

Выход из справки: q.

- 1 Исполняемые программы или команды оболочки (shell)
- 2 Системные вызовы (функции, предоставляемые ядром)
- 3 Библиотечные вызовы (функции, предоставляемые программными библиотеками)
- 4 Специальные файлы (обычно находящиеся в каталоге /dev)
- 5 Форматы файлов и соглашения, например о /etc/passwd
- 6 Игры
- 7 Разное (включает пакеты макросов и соглашения), например man(7), groff(7)
- 8 Команды администрирования системы (обычно, запускаемые только суперпользователем)
- 9 Процедуры ядра [нестандартный раздел]

Нам нужен второй раздел

```
man 2 mkdir  
man mkdir
```

Используйте те типы, которые указаны в описании функции.
Для указания количества или размера используются

```
size_t  
ssize_t
```

которые являются аналогами *unsigned int* и *int* соответственно.
Отрицательные значения, как правило, должны сигнализировать об ошибке.

Системный ввод и вывод

Низкоуровневый ввод и вывод:

```
#include <unistd.h>
```

```
ssize_t read(int fd, void *buf, size_t count);  
ssize_t write(int fd, void *buf, size_t count);
```

здесь fd — файловый дескриптор (0 - ввод, 1 - вывод, 2 - ошибки, остальные - по умолчанию закрыты), buf — указатель на буфер при чтении или записи, count — максимальный размер. В случае ошибки (возвращает отрицательно число), номер ошибки сохраняется в errno.

```
#include <stdio.h>
```

```
void perror(const char *s);
```

Выводим текст ошибки со своими комментариями.

Системный ввод и вывод

```
int main(int argc, char **argv) {
    char buf[10];
    size_t count = 10;
    size_t len = read(0, buf, count);
    if (len < 0) {
        perror("My comment of failed read()");
        return 1;
    }
    if (write(1, buf, len) < 0) {
        perror("My comment of failed write()");
        return 1;
    }
    return 0;
}
```

Если попытаться вывести в дескриптор 3. то будет

```
My comment of failed write(): Bad file descriptor
```

Фильтр потока ошибок

```
./prog 1>/dev/null
```

Логирование потока ошибок

```
./prog 2>log.txt
```

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

int open(const char *pathname, int flags);
int open(const char *pathname, int flags, mode_t m);
int close(int fd);
```



```
int fd = open("info.txt",
              O_WRONLY|O_CREAT|O_TRUNC,
              S_IRUSR|S_IWUSR);

if (fd < 0) {
    perror("Hey! You could not open file, man");
    return 1;
}

if (write(fd, "Hello", 6) < 0) {
    perror("Hey! You could not write to file, man");
    return 1;
}

if (close(fd) < 0) {
    perror("Hey! You could not close the file, man");
    return 1;
}
```

O_WRONLY, O_CREAT, O_TRUNC — открыть файла на запись; создать, если нет файла; начать запись файла с начала.

S_IRUSR, S_IWUSR — дать права запустившему пользователю на чтение и

Работа с директориями

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

int mkdir(const char *pathname, mode_t mode);
int mkdir(const char *pathname);
int chdir(const char *pathname);
int rmdir(const char *pathname);
char *getcwd(char *buf, size_t size);
```

```
#include <sys/types.h>
#include <dirent.h>

DIR *opendir(const char *pathname);
struct dirent *readdir(DIR *dir);
int closedir(DIR *);
```

Идентификатор процесса pid

Посмотреть топ процессы

```
top
```

PID	USER	VIRT	%CPU	%MEM	TIME+	COMMAND
17800	alen	4508	100,0	0,0	0:07.04	prog03
17802	alen	52604	12,5	0,1	0:00.02	top
1275	alen	3837112	6,2	10,6	159:14.75	gnome-shell
16965	alen	2302848	6,2	6,9	1:31.32	firefox

Предварительно запустили висящий процесс с именем prog03.

Идентификатор процесса pid

```
#include <unistd.h>

pid_t getpid(void);
pid_t getppid(void);
```

Родительский процесс — тот кто создал (по умолчанию init — процесс с id 1).

```
#include <unistd.h>
#include <stdio.h>

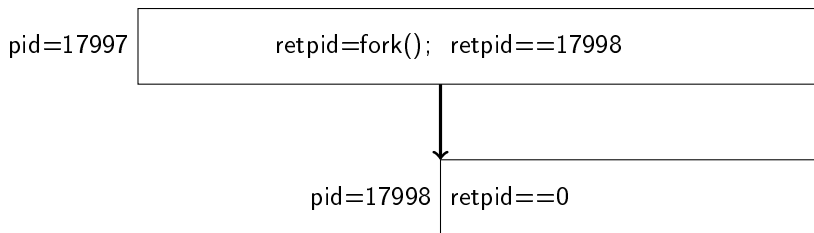
int main(int argc, char **argv) {
    pid_t pid = getpid();
    printf("%u\n", pid);
    while(1) {
    }
    return 0;
}
```

Создание дочернего процесса

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

pid_t fork(void);
```

Отличия: результат возврата fork (в дочке – ноль, в родителе – pid дочки).
Копируются: сегмент кода и сегменты данных.
Сохраняются: дескрипторы.



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

int main(int argc, char **argv) {
    pid_t pid = fork();
    if (pid == 0) {
        printf("I am child, my pid is %u\n", getpid());
    } else {
        printf("I am parent, my pid is %u\n", getpid());
    }
    while (1) {
    }
    return 0;
}
```

Вывод:

```
I am parent, my pid is 17997  
I am child, my pid is 17998
```

Результат top:

PID	USER	VIRT	%CPU	%MEM	TIME+	COMMAND
17997	alen	4508	100,0	0,0	0:15.20	prog04
17998	alen	4508	100,5	0,0	0:15.19	prog04

Замена процесса exec

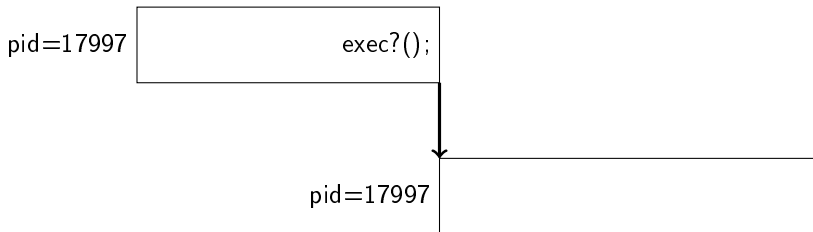
```
#include <unistd.h>

int execl(const char *path, const char *arg,..., NULL);

int execl(const char *path, const char *arg,..., NULL);
int execlp(const char *file, const char *arg,..., NULL);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
```

Заменяются: сегмент код, сегмент данных. Сохраняются: дескрипторы.

Суффикс p: ищет в bin и /usr/bin.



```
#include <unistd.h>
#include <stdio.h>
int main(int argc, char **argv) {
    if (execlp("ls", "ls", "-l", "-a", NULL) < 0) {
        perror("ls failed");
        return 1;
    }
    return 0;
}
```

Запускается команда *ls*. В качестве аргументов передается: *ls* и *-l*.

```
#include <unistd.h>
#include <stdio.h>
int main(int argc, char **argv) {
    char cmd[] = "ls";
    char arg1[] = "-l";
    char arg2[] = "-a";
    char *arg_vec[4] = {cmd, arg1, arg2, NULL};
    if (execvp(cmd, arg_vec) < 0) {
        perror("ls failed");
        return 1;
    }
    return 0;
}
```

`arg_vec` — массив из четырех указателей типа `char *` (то есть массив из 4 строк).

Классическая схема запуска другого процесса

```
if (fork() > 0){  
    /* parent */  
} else {  
    /* child */  
    if (exec?(cmd, ...) < 0) {  
        perror("exec failed");  
        return 1;  
    }  
}
```

