POSIX
Командная оболочка и системные вызовы

Роман Чепляка

22 октября 2010
НТУУ КПИ

Что такое POSIX?

POSIX

- Portable Operating System Interface for Unix
- Семейство стандартов, описывающих программные интерфейсы ОС
- POSIX реализован в:
  - Linux, Android
  - Darwin (Mac OS X, iOS)
  - Windows NT (Subsystem for UNIX-based Applications)
  - FreeBSD,OpenBSD,NetBSD
  - Solaris
  - и др.

Что описывает POSIX?

Стандарт POSIX описывает:

- Системные вызовы
- Командную оболочку (Shell) и утилиты

Запуск программы

Shell

```
ls
```

Запуск программы

Shell

```
ls
```

C

```
pid_t p = fork();
int status;
if (p) {
    waitpid(p, &status, options);
} else {
    const args[] = {"ls", 0};
    execvp("ls", args);
}
```

Перенаправление вывода

Shell

```
ls > a.txt
```

Файловые дескрипторы и описания

Файловый дескриптор

- уникальное для данного процесса целое число
- ссылается на описание открытого файла

Описание открытого файла

- содержится в глобальной таблице операционной системы
- содержит информацию о самом файле, текущей позиции и режиме доступа
- на одно описание могут ссылаться разные дескрипторы, возможно из разных процессов

Файловые дескрипторы

Откуда берутся файловые дескрипторы

- Наследуются при вызовах fork и exec (указывают на то же самое описание)

Стандартные файловые дескрипторы

0 – стандартный ввод (stdin)

1 – стандартный вывод (stdout)

2 – стандартный вывод ошибок (stderr)

Файловые дескрипторы

Откуда берутся файловые дескрипторы

- Наследуются при вызовах fork и exec
  (указывают на то же самое описание)
- Создаются вызовом

  int open(**const char** *pathname, **int** flags, **mode_t** mode)

  (создается новое описание)

Файловые дескрипторы

Откуда берутся файловые дескрипторы

- Наследуются при вызовах fork и exec
  (указывают на то же самое описание)
- Создаются вызовом

  int open(**const char** *pathname, **int** flags, **mode_t** mode)

  (создается новое описание)
- Копируются вызовом

  int dup2(**int** oldfd, **int** newfd);

  (oldfd и newfd указывают на одно и то же описание)

Перенаправление вывода

C

```c
pid_t p = fork();
int status;
if (p) {
    waitpid(p, &status, options);
} else {
    int flags = O_WRONLY | O_CREAT | O_TRUNC;
    int fd = open("a.txt", flags, 0666);
    dup2(fd, 1);
    close(fd);
    execvp("ls", args);
}
```

Перенаправление вывода и ошибок

Shell

```
find > a.txt 2> a.txt
```

Перенаправление вывода и ошибок

C

```
int flags = O_WRONLY | O_CREAT | O_TRUNC;
int fd;

fd = open("a.txt", flags, 0666);
dup2(fd, 1);
close(fd);

fd = open("a.txt", flags, 0666);
dup2(fd, 2);
close(fd);

execvp("find", args);
```

Копирование файловых дескрипторов

Shell

```
cmd 2>&1
```

C

```
dup2(1,2);
```

Перенаправление вывода и ошибок

Shell

```
find >a.txt 2>&1
```

Перенаправление вывода и ошибок

Shell

```
find >a.txt 2>&1
```

Shell

```
find 2>&1 >a.txt
```

Перенаправление вывода и ошибок

Shell

```
find >a.txt 2>&1
```

C

```
int flags = O_WRONLY | O_CREAT | O_TRUNC;
int fd;
fd = open("a.txt", flags, 0666);
dup2(fd, 1);
close(fd);
dup2(1, 2);
execvp("find", args);
```

Запуск программы в фоновом режиме

Shell

```
rm * &
```

Запуск программы в фоновом режиме

Shell

```
rm * &
```

C

```
pid_t p = fork();
if (!p) {
    int fd = open("/dev/null", O_RDONLY);
    dup2(fd, 0);
    close(fd);
    execvp("rm", args);
}
```

Каналы

Shell

```
mount | grep tmp
```

Каналы

C

```
int fd[2], status;
pid_t p1,p2;
pipe(fd);
if ((p1 = fork()) == 0) {
      dup2(fd[1], 1); close(fd[0]); close(fd[1]);
      execvp("mount", args1);
}
if ((p2 = fork()) == 0) {
      dup2(fd[0], 0); close(fd[0]); close(fd[1]);
      execvp("grep", args2);
}
waitpid(p1, NULL, options);
waitpid(p2, &status, options);
```

The End

Пишите письма!

roma@ro-che.info