Секция № 3

Системные вызовы и ввод/вывод

Содержание

1. Справочная информация

1.1. Словарь

Системный вызов

В компьютерных науках под этим понимают запрос к ядру ОС для выполнения некоторого действия: взаимодействие с железом, создание и выполнение новых процессов, обращение к внутренним сервисам ядра (например, к планировщику).

Дескрипторы файлов

На *пользовательском низкоуровневом* уровне доступа к файлам — это целочисленные индексы из управляемой ядром таблицы файл-дескрипторов (*уровень ядра ОС*). В результате выполнения системного вызова, требующего открытия файла, ядро ОС создает файл-дескриптор и связывает его с абстракцией нижележащего файлоподобного объекта. Нижележащий объект может быть аппаратным устройством, файловой системой или чем угодно другим. Дескрипторы файлов позволяют перенаправлять запросы процесса на чтение/запись данных в правильные места ядра ОС в зависимости от типа файлоподобного объекта. При запуске программы у тебя будет три существующих дескриптора файлов.

Дескриптор файла	Файл	Комментарий
`0`	`stdin`	Стандартный ввод
`1`	`stdout`	Стандартный вывод
`2`	`stderr`	Стандартный вывод ошибок

`int open(const char *path, int flags)`

системный вызов для *открытия* файла и получения его дескриптора. После открытия файла, позиция считывания данных указывает на начало файла и равна `0`.

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

системный вызов, использующийся для чтения `count` байтов в буфер `buf` из файла с дескриптором `fd`, начиная с текущей позиции. После считывания позиция в файле увеличивается на число считанных байтов.

`size_t write(int fd, const void *buf, size_t count)`

системный вызов, использующийся для записи `count` байтов из буфера `buf` в файла с дескриптором `fd`, запись начинается с позиции в файле. позиция в файле увеличивается на число считанных байтов.

`size_t lseek(int fd, off_t offset, int whence)`

системный вызов, позволяющий изменять значение текущей позиции открытого файла. Формальный параметр `whence` может принимать следующие значения:

- `SEEK_SET` — позиция в файле устанавливается в значение параметра `offset`.
- `SEEK_CUR` — позиция в файле смещается на `offset` от текущей позиции.

- `SEEK_END` — позиция в файле устанавливается определяется как размер файла + `offset`.

`int dup(int oldfd)`

создаёт дубликат (псевдоним) для передаваемого дескриптора файла `oldfd` и возвращает его. Эта функция всегда выбирает наименьшее возможное значение для нового дескриптора файла. То есть, если программа первым делом вызовет `dup`, то вернется значение `3` (значения `0`, `1` и `2` уже заняты под `stdin`, `stdout`, `stderr`). Оригинальное значение и дубликат ссылаются на одну и ту же запись в таблице дескрипторов ОС и могут использоваться взаимозаменяемо.

`int dup2(int oldfd, int newfd)`

этот системный вызов похож на `dup`, он создаёт дубликат `oldfd`, но в этом случае новое значение не выбирается ОС, а указывается в вызове (`newfd`). Если значение `newfd` на момент вызова уже используется, то связанный с ним файл будет закрыт. Это очень удобно при перенаправлении вывода, поскольку происходит *автоматическое* закрытие открытого файла. Например, если ты хочешь перенаправить `sdtout` в файл, то надо вызвать `dup2` с аргументами: `oldfd` — дескриптор открытого файла, `newfd` — дескриптор `sdtout` (`1`).

Сигналы

Это программные прерывания, то есть способ информирования процесса о состоянии других процессов, операционной системы или железа. Сигнал является прерыванием в том смысле, что он изменяет ход выполнения программы — процесс будет приостановлен в текущем месте для обработки сигнала, либо для игнорирования сигнала, либо, в отдельных случаях, для завершения процесса — в зависимости от сигнала.

`int signal(int signum, void (*handler)(int))`

Это простейший системный вызов для обработки сигналов. Аргументами являются: собственно сигнал (`signum`) и указатель на функцию-обработчик такого сигнала (`handler`).

`SIG_IGN, SIG_DFL`

обычно второй аргумент `signal` указывает на пользовательскую функцию-обработчик. Однако, если требуется игнорировать сигнал, то можно использовать `SIG_IGN`. Если же требуется, чтобы процесс реагировал на сигнал обычным образом, то нужно использовать `SIG_DFL`.

1.2. Сигналы

Ниже перечислены стандартные сигналы в Linux.

Сигнал	Значение	Действие	Комментарий
SIGHUP	1	Завершение	Разрыв соединения с управляющим терминалом, или смерть управляющего процесса.

Сигнал	Значение	Действие	Комментарий
SIGINT	2	Завершение	Остановка процесса пользователем с терминала (Ctrl+c).
SIGQUIT	3	Дамп ядра	Завершение процесса пользователем с терминала (Ctrl+\).
SIGILL	4	Дамп ядра	Выполнение неправильно сформированной, несуществующей или привилегированной инструкции.
SIGABRT	6	Дамп ядра	Посылается процессом самому себе при выполнении функции `abort()`, для аварийного останова, в случае невозможности дальнейшего продолжения программы.
SIGFPE	8	Дамп ядра	Попытка выполнить ошибочную арифметическую операцию с плавающей запятой.
SIGKILL	9	Завершение	Немедленное завершение процесса без возможности перехватить или проигнорировать.
SIGSEGV	11	Дамп ядра	Обращение к несуществующей памяти или обращения с нарушением прав доступа.
SIGPIPE	13	Завершение	Запись в соединение (пайп, сокет) при отсутствии или обрыве соединения с другой (читающей) стороной.
SIGALRM	14	Завершение	Посылается процессу по истечении времени, предварительно заданного функцией `alarm()`.
SIGTERM	15	Завершение	Запрос завершения процесса.
SIGUSR1	30, 10, 16	Завершение	Пользовательский сигнал 1 для межпроцессной синхронизации и управления.
SIGUSR2	31, 12, 17	Завершение	Пользовательский сигнал 2 для межпроцессной синхронизации и управления.
SIGCHLD	20, 17, 18	Игнорировать	Дочерний процесс завершен или остановлен.
SIGCONT	19, 18, 25	Продолжить	Продолжить выполнение ранее остановленного процесса.
SIGSTOP	17, 19, 23	Стоп	Остановка выполнения процесса.
SIGTSTP	18, 20, 24	Стоп	Сигнал остановки с терминала (Ctrl+z).

Сигнал	Значение	Действие	Комментарий
SIGTTIN	21, 21, 26	Стоп	Попытка чтения с терминала фоновым процессом.
SIGTTOU	22, 22, 27	Стоп	Попытка записи на терминал фоновым процессом.

2. Задачи

2.1. Разогрев

Как остановить следующую программу?

```
int main(){
  signal(SIGINT, SIG_IGN);
  while(1);
}
```

Ответ

2.1. Обработчики сигналов

Заполните пропуски в следующей функции так, чтобы при нажатии Ctrl+C пользователю выводилось бы сообщение «Do you really want to quit [y/n]?» Если пользователь ответит «y», то программа завершится, иначе продолжит выполнение.

```
void sigint_handler(int sig)
{
    char c;
    printf("Ouch, you just hit Ctrl-C?. Do you really want to quit [y/n]?");
    c = getchar();
    if (c == 'y' || c == 'Y')

        ______________________________;
}
int main() {

    ____________________________;
    ...
}
```

2.3. Файлы

2.3.1. Файлы vs дескрипторы файлов

В чём разница между `fopen` и `open`?

Ответ

2.3.2. Быстрый пример на write и seek

Что будет в файле `test.txt` после выполнения программы?

Если смещение `offset` при записи находится за пределами файла, то байты между старым концом файла и новыми записанными данными будут выставлены в ноль.

```c
int main() {
    char buffer[200];
    memset(buffer, 'a', 200);
    int fd = open("test.txt", O_CREAT|O_RDWR);
    write(fd, buffer, 200);
    lseek(fd, 0, SEEK_SET);
    read(fd, buffer, 100);
    lseek(fd, 500, SEEK_CUR);
    write(fd, buffer, 100);
}
```

Ответ

2.4. Функции dup и dup2

2.4.1. Разогрев

Что будет выведено после запуска?

```
int main(int argc, char **argv)
{
    int pid, status;
    int newfd;
    if ((newfd = open("output_file.txt", O_CREAT|O_TRUNC|O_WRONLY, 0644)) <
0) {
        exit(1);
    }
    printf("The last digit of pi is...");
    dup2(newfd, 1);
    printf("five\n");
    exit(0);
}
```

Ответ

2.4.2. Перенаправление

Объясни что происходит и что будет выведено.

```c
int main(int argc, char **argv)
{
    int pid, status;
    int newfd;
    char *cmd[] = { "/bin/ls", "-al", "/", 0 };
    if (argc != 2) {
        fprintf(stderr, "usage: %s output_file\n", argv[0]);
        exit(1);
    }
    if ((newfd = open(argv[1], O_CREAT|O_TRUNC|O_WRONLY, 0644)) < 0) {
        perror(argv[1]);    /* open failed */
        exit(1);
    }
    printf("writing output of the command %s to \"%s\"\n", cmd[0], argv[1]);
    dup2(newfd, 1);
    execvp(cmd[0], cmd);
    perror(cmd[0]);     /* execvp failed */

    exit(1);
}
```

Ответ

2.4.3. Перенаправление в новом процессе

Измени предыдущий код так, чтобы результат `ls -al` был бы записан в файл, передаваемый аргументом, а в терминал, сразу после этого было бы выведено «all done».

Нужно использовать `fork` и `wait`.

Ответ