OS Lab 26

Status	approved
checkbox	☑
class	OS
due date	@Apr 14, 2021

Task

💡 26. Связь с использованием функций стандартной библиотеки
Используйте стандартные библиотечные функции `popen(3)` и `pclose(3)` для выполнения тех же операций, что и в предыдущем упражнении.

Поскольку чаще всего канал создается для взаимодействия с другим процессом, чтобы получать от него или отправлять ему данные, стандартная библиотека ввода/вывода традиционно поддерживает функции `pclose` и `popen`. Эти две функции берут на себя всю рутинную работу, которую мы до сих пор выполняли самостоятельно: создание канала, создание дочернего процесса, закрытие неиспользуемых дескрипторов канала, запуск команды и ожидание завершения команды

Функция `popen` посредством `fork` и `exec` запускает команду `cmdstring` и возвращает указатель на объект `FILE`. Если в аргументе *type* передается значение `"r"`, указатель на файл будет связан со стандартным выводом *cmdstring* (рис. 15.5).

Рис. 15.5. Результат выполнения инструкции fp = popen(cmdstring, «r»)

Если в аргументе *type* передается значение `"w"`, указатель на файл будет связан со стандартным вводом *cmdstring* (рис. 15.6).

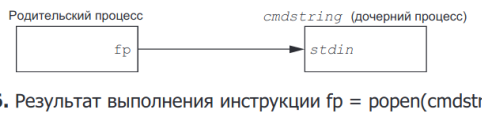

Рис. 15.6. Результат выполнения инструкции fp = popen(cmdstring, «w»)

Функция `pclose` закрывает поток ввода/вывода, ожидает завершения команды и возвращает код завершения командного интерпретатора, запущенного для выполнения команды `cmdstring`. Если командный интерпретатор не смог запуститься, функция `pclose` вернет код завершения, как если бы командная оболочка вызвала `exit(127)`

Обратите внимание, что функция `popen` никогда не должна вызываться из про-грамм, для которых установлен бит `set-user-ID` или `set-group-ID`. Выполнение коман ды функцией `popen` эквивалентно выполнению инструкции `execl("/bin/sh", "sh", "-c", command, NULL);` которая запускает командный интерпретатор и команду

```
#include <stdio.h>

FILE *popen(const char *cmdstring, const char *type);
```

Возвращает указатель на структуру FILE в случае успеха, NULL — в случае ошибки

```
int pclose(FILE *fp);
```

Возвращает код завершения *cmdstring*, –1 — в случае ошибки

```
The popen() function creates a pipe between the calling pr
ogram and the
       command to be executed.  The arguments to popen() a
re pointers to null-
       terminated strings.  The command argument consists
 of a shell command
       line.  The mode argument is an I/O mode, either r f
or reading or w for
       writing. The value returned is a stream pointer suc
h that one can write
       to the standard input of the command, if the I/O mo
de is w, by writing to
       the file stream (see Intro(3)); and one can read fr
om the standard output
       of the command, if the I/O mode is r, by reading fr
om the file stream.
       Because open files are shared, a type r command may
be used as an input
       filter and a type w as an output filter

The environment of the executed command will be as if a ch
ild process
       were created within the popen() call using fork(2).
The child is created
       as if invoked with the call:

RETURN VALUES

       Upon successful completion, popen() returns a point
er to an open stream
       that can be used to read or write to the pipe. Othe
rwise, it returns a
       null pointer and may set errno to indicate the erro
r.
```

```
The pclose() function closes a stream opened by popen() by
closing the
       pipe. It waits for the associated process to termin
ate and returns the
       termination status of the process running the comma
nd language
       interpreter.  This is the value returned by waitpid
(3C). See
       wait.h(3HEAD) for more information on termination s
```

command с окружением, унаследованным от вызывающей программы. В этом случае злоумышленник, манипулируя значениями переменных окружения, получает возможность запускать произвольные команды с повышенными привилегиями

```
#include <stdio.h>

#define LEN 43

int main(){

    char OutputText[LEN] = "Greetings from the other side
 of the pipe\n";

    FILE *pipe = popen("./printInUpper", "w");
    if (pipe == NULL){
        perror("failed to create pipe");
        return -1;
    }

    fwrite(OutputText, sizeof(char), LEN, pipe);
    pclose(pipe);
    return 0;
}
```

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

#define LEN 43

int main(){
    char Char;
    char string[LEN];
    int readCount;

    while ((readCount = read(STDIN_FILENO, string, LEN)) >
0){
        for (int i = 0; i < readCount; i++){
            Char = string[i];
            putchar(toupper(Char));
        }
    }

    return 0;
}
```

```
tatus. If, however, a
       call to waitpid() with a pid argument equal to the
 process ID of the
       command line interpreter causes the termination sta
tus to be unavailable
       to pclose(), then pclose() returns -1 with errno se
t to ECHILD to report
       this condition.

RETURN VALUES

       Upon successful completion, pclose() returns the te
rmination status of
       the command language interpreter as returned by wai
tpid().  Otherwise, it
       returns -1 and sets errno to indicate the error.
```

В принципе, функция `popen` похожа на тот код, который мы использовали ранее в этой главе, однако существует ряд моментов, которые необходимо принять во внимание. Прежде всего, каждый раз, когда вызывается функция `popen`, нужно запоминать идентификатор дочернего процесса и дескриптор файла либо указатель на объект `FILE`. Мы решили сохранять идентификатор дочернего процесса в массиве `childpid`, который индексируется номерами файловых дескрипторов. Благодаря этому функция `pclose`, получая указатель на объект `FILE`, сможет восстановить по нему номер дескриптора файла с помощью функции `fileno` и затем извлечь из массива идентификатор дочернего процесса, чтобы передать его функции `waitpid`. Поскольку данный процесс может вызывать функцию `popen` много раз, при первом обращении к функции `popen` мы размещаем в динамической памяти массив `childpid` максимального размера.

Обратите внимание, что функция `open_max` из листинга 2.4 может вернуть предполагаемое максимально допустимое количество открытых файлов, если это значение не определено системой. Следует проявлять особую осторожность и избегать использования файловых дескрипторов, номера которых больше (или равны) значения, возвращаемого функцией `open_max`. В функции `popen`, если значение, возвращаемое вызовом `open_max`, окажется слишком маленьким, мы закрываем дескрипторы канала, устанавливаем в переменной `errno` код ошибки `EMFILE` (чтобы сообщить, что открыто слишком много файлов) и возвращаем –1. В функции `pclose`, если файловый дескриптор, соответствующий аргументу *fp*, оказывается больше ожидаемого, мы устанавливаем в переменной `errno` код ошибки `EINVAL`

Что случится, если процесс, вызывающий `pclose`, установил обработчик сигнала `SIGCHLD`? В этом случае функция `waitpid` вернет код ошибки `EINTR`. Так как мы допускаем, что вызывающий процесс может перехватывать сигнал `SIGCHLD` (или любой другой сигнал, в результате чего может быть прервано выполнение системного вызова `waitpid`), мы просто вызываем функцию `waitpid` еще раз, если ее выполнение прервано в результате перехвата сигнала.

Обратите внимание: приложение может самостоятельно вызвать функцию `waitpid` и получить код завершения дочернего процесса, созданного функцией `popen`. В этом случае функция `waitpid`, вызываемая из `pclose`, обнаружит отсутствие дочернего процесса и вернет значение –1 с кодом ошибки `ECHILD` в `errno`. Такое поведение регламентируется стандартом POSIX.1.

627 / 1065

Некоторые ранние версии `pclose` возвращали код ошибки `EINTR`, если выполнение функции `wait` было прервано перехваченным сигналом. Кроме того, в некоторых ранних версиях `pclose` игнорировались или блокировались сигналы `SIGINT`, `SIGQUIT` и `SIGHUP`. В стандарте POSIX.1 такое поведение считается недопустимым.