Задачи със системните примитиви read() и write()

Задача 1 – *safe_rw*

Според POSIX стандарта, ако системният примитив `read()`/`write()` се прекъсне от сигнал преди да е прочел/записал каквито и да е данни, той трябва да върне `-1` и да зададе стойност `EINTR` на променливата `errno`. Ако пък `read()`/`write()` се прекъсне от сигнал след като е прочел/записал успешно някакви данни, трябва да се счита за успешно завършил и да върне броя на прочетените/записаните байтове.

Във връзка с това поведение, честа и добра практика е да се дефинират обгръщащи ги функциии (wrappers):

```
ssize_t safe_read(int fd, void * buffer, size_t nbytes);
ssize_t safe_write(int fd, void * buffer, size_t nbytes);
```

при което поведението е същото като при `read()` и `write()` с изключение на случая, в който `read()`/`write()` се прекъсне от сигнал преди да е прочел/записал каквито и да е данни – тогава те трябва просто да продължат да опитват да прочетат/запишат данните.

Задача 2 – *full_rw*

Друга честа и добра практика е да се дефинират още две обгръщащи функции (wrappers) за `read()` и `write()`:

```
ssize_t full_read(int fd, void * buffer, size_t nbytes);
ssize_t full_write(int fd, void * buffer, size_t nbytes);
```

чиято цел е, стига във файла да има, да прочетат/запишат точно `nbytes` от/в него.

Задача 3 – *our_cat*

Да се напише функция `simple_cat`, приемаща подходящи аргументи, чиято цел е да изчете съдържанието на файл и да го изведе на стандартния изход.

Да се демонстрира действието на тази функция в програма, която има поведение, подобно на програмата *cat* – приема като аргументи имена на обикновени файлове и извежда на стандартния изход съдържанието на тези файлове в указания ред. Ако няма аргументи, програмата трябва да изведе на стандартния изход прочетеното от стандартния вход.

Забележка: Ще считаме, че, стига като аргументи да са подадени имена на

съществуващи файлове, тези файлове са обикновени. Ако някой от аргументите е име на несъществуващ файл, това трябва да бъде указано на потребителя, но да не прекъсва изпълнението на цялата процедура за останалите файлове. Ако някой аргумент е "-", считаме, че той означава стандартния вход (който обаче може да не е обикновен файл, а например терминално устройство!).

Задача 4 – *our_cp*

Да се напише програма, която

а) приема два аргумента `source` и `dest` и има поведението на `cp source dest`

б) приема два аргумента source и directory и има поведението на `cp source directory`

в) приема поне два аргумента – `file1`, `file2`, …, `fileN`, `directory` – и има поведението на `cp file1 file2 … fileN directory`

г) приема поне два аргумента, като, ако са точно два, има подходящото поведение измежду а) и б), а ако са повече от два, има поведението от в).

Забележка:

а) `source`, ако е име на съществуващ файл, считаме, че това е обикновен файл. `dest` може да е както име на съществуващ, така и на несъществуващ файл.

б) `source`, ако е име на съществуващ, считаме, че това е обикновен файл. Считаме, че `directory` е име на съществуваща директория.

в) `fileX`, ако е име на съществуващ файл, считаме, че това е обикновен файл. Считаме, че `directory` е име на съществуваща директория.