Трето контролно за курса “Операционни системи”, СУ, ФМИ, 17.06.2017 г.

Име:_____ ФН:_____ Спец.:__ Курс:__ Гр.:__

Задача	1 (30)	2 (30)	3 (30)	4 (30)	Общо (120)
получени точки					

Задача 1. Напишете програма на C, която приема три параметъра – имена на двоични файлове.
Примерно извикване:

```
$ ./main f1.bin f2.bin patch.bin
```

Файловете `f1.bin` и `f2.bin` се третират като двоични файлове, състоящи се от байтове (`uint8_t`). Файлът `f1.bin` е “оригиналният” файл, а `f2.bin` е негово копие, което е било модифицирано по някакъв начин (извън обхвата на тази задача). Файлът `patch.bin` е двоичен файл, състоящ се от наредени тройки от следните елементи (и техните типове):

- *отместване* (`uint16_t`) – спрямо началото на `f1.bin`/`f2.bin`
- *оригинален байт* (`uint8_t`) – на тази позиция в `f1.bin`
- *нов байт* (`uint8_t`) – на тази позиция в `f2.bin`

Вашата програма да създава файла `patch.bin`, на базата на съществуващите файлове `f1.bin` и `f2.bin`, като описва вътре само разликите между двата файла. Ако дадено отместване съществува само в единия от файловете `f1.bin`/`f2.bin`, програмата да прекратява изпълнението си по подходящ начин.

Примерен `f1.bin`:

```
00000000: f5c4 b159 cc80 e2ef c1c7 c99a 2fb0 0d8c  ...Y......../...
00000010: 3c83 6fed 6b46 09d2 90df cf1e 9a3c 1f05  <.o.kF.......<..
00000020: 05f9 4c29 fd58 a5f1 cb7b c9d0 b234 2398  ..L).X...{...4#.
00000030: 35af 6be6 5a71 b23a 0e8d 08de def2 214c  5.k.Zq.:......!L
```

Примерен `f2.bin`:

```
00000000: f5c4 5959 cc80 e2ef c1c7 c99a 2fb0 0d8c  ..YY......../...
00000010: 3c83 6fed 6b46 09d2 90df cf1e 9a3c 1f05  <.o.kF.......<..
00000020: 05f9 4c29 fd58 a5f1 cb7b c9d0 b234 2398  ..L).X...{...4#.
00000030: afaf 6be6 5a71 b23a 0e8d 08de def2 214c  ..k.Zq.:......!L
```

Примерен `patch.bin`:

```
00000000: 0200 b159 3000 35af                      ...Y0.5.
```

Задача 2. Напишете програма на C, която използвайки външни shell команди да извежда статистика за използването на различните shell-ове от потребителите, дефинирани в системата. Изходът да бъде сортиран във възходящ ред според брой използвания на shell-овете.

Примерно извикване:

```
$ ./main
      1 /bin/sync
      3 /bin/bash
      7 /bin/false
     17 /usr/sbin/nologin
```

Задача 3. Напишете програма на C, която да работи подобно на командата `cat`, реализирайки само следната функционалност:

- общ вид на изпълнение: `./main [OPTION] [FILE]...`
- ако е подаден като **първи** параметър `-n`, то той да се третира като опция, което кара програмата ви да номерира (глобално) всеки изходен ред (започвайки от 1).
- програмата извежда на `STDOUT`
- ако няма подадени параметри (имена на файлове), програмата чете от `STDIN`
- ако има подадени параметри – файлове, програмата последователно ги извежда
- ако някой от параметрите е тире (`-`), програмата да го третира като специално име за `STDIN`

Примерно извикване:

```
$ cat a.txt
a1
a2

$ cat b.txt
b1
b2
b3

$ echo -e "s1\ns2" | ./main -n a.txt - b.txt
     1 a1
     2 a2
     3 s1
     4 s2
     5 b1
     6 b2
     7 b3
```

Забележка: Погледнете `setbuf(3)` *и* `strcmp(3)`.

Задача 4. Процесът `P` създава тръба (pipe) с извикване на функцията `pipe(int pipefd[2])` в ОС GNU/Linux.

а) Кои процеси не могат да ползват тръбата?

б) Опишете друг метод за изграждане на комуникационен канал, който дава възможност на произволни процеси да изградят и ползват канала. Допълнително искаме новоизградения канал да е достъпен само за процесите, които са го създали.

Упътване: Прочетете man-страницата за функцията pipe.

Забележки за задачи 1-3:

- Полезни man страници:

  `open(2)` `close(2)` `read(2)` `write(2)`
  `fork(2)` `wait(2)` `exec(3)`
  `pipe(2)` `dup(2)`
  `exit(3)` `err(3)` `stat(2)` `printf(3)` `setbuf(3)` `strcmp(3)`

- Препоръчителни флагове на компилатора: `-std=c99 -Wall -Wpedantic -Wextra`
- Обърнете внимание на коментарите, именуването на променливи и подреждането на кода
- Пишете код, все едно проверяващият е психопат, който знае къде живеете.

Примерни решения

Задача 1.

```
#include <err.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdint.h>
#include <stdio.h>

static off_t statx_size(const char* fname)
{
    struct stat buf;
    if (stat(fname, &buf) == -1) {
        err(1, "Cannot stat %s", fname);
    }
    if (!S_ISREG(buf.st_mode)) {
        errx(1, "%s is not a regular file", fname);
    }
    return buf.st_size;
}

static void readx(const int fd, uint8_t* const buf, const size_t sz, const char* const fname)
{
    size_t left = sz;
    while (left > 0) {
        const ssize_t n = read(fd, buf + sz - left, left);
        if (n == -1) {
            err(1, "Could not read from %s", fname);
        } else if (n == 0) {
            errx(1, "Unexpected EOF on %s", fname);
        }
        left -= n;
    }
}

static void writex(const int fd, const uint8_t* const buf, const size_t sz,
        const char* const fname, const uint16_t offset)
{
    size_t left = sz;
    while (left > 0) {
        const ssize_t n = write(fd, buf + sz - left, left);
        if (n == -1) {
            err(1, "Could not write the change for offset %04x to %s", offset, fname);
        } else if (n == 0) {
            errx(1, "Unexpected short write on %s", fname);
        }
        left -= n;
    }
}

int main (const int argc, const char* const argv[])
{

    if (argc != 4) {
        errx(1, "Invalid number of arguments. Usage: %s <f1.bin> <f2.bin> <patch.bin>", argv[0]);
    }

    const off_t f1_size = statx_size(argv[1]);
    if (f1_size != statx_size(argv[2])) {
```

```
        errx(1, "%s and %s have different file size.", argv[1], argv[2]);
    }

    if (f1_size > UINT16_MAX) {
        errx(1, "%s size %ld bigger than %d, could not process.", argv[1], f1_size, UINT16_MAX);
    }

    const int f1 = open(argv[1], O_RDONLY);
    if (f1 == -1) {
        err(1, "Could not open %s", argv[1]);
    }

    const int f2 = open(argv[2], O_RDONLY);

    if (f2 == -1) {
        const int saved_errno = errno;
        close(f1);
        errno = saved_errno;
        err(1, "Could not open %s", argv[2]);
    }

    const int pf = open(argv[3], O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    if (pf == -1) {
        const int saved_errno = errno;
        close(f1);
        close(f2);
        errno = saved_errno;
        err(1, "Could not open %s", argv[3]);
    }

    struct {
        uint16_t    offset;
        uint8_t     orgbyte;
        uint8_t     newbyte;
    } __attribute__((packed)) element;

    for (element.offset = 0; element.offset < f1_size; element.offset += sizeof(element.orgbyte)) {
        readx(f1, &element.orgbyte, sizeof(element.orgbyte), argv[1]);
        readx(f2, &element.newbyte, sizeof(element.newbyte), argv[2]);
        if (element.orgbyte != element.newbyte) {
            writex(pf, (const uint8_t *)&element, sizeof(element), argv[3], element.offset);
        }
    }

    if (close(f1) == -1)
        err(1, "Could not close %s", argv[1]);
    if (close(f2) == -1)
        err(1, "Could not close %s", argv[2]);
    if (close(pf) == -1)
        err(1, "Could not close %s", argv[3]);

    return 0;
}
```

Задача 2.

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

int main()
{
```

```
int fd1[2];
int fd2[2];
int fd3[2];

pipe(fd1);
pipe(fd2);
pipe(fd3);

pid_t cut_pid;
pid_t sort_pid;
pid_t uniq_pid;

if ((cut_pid = fork()) == -1) {
    err(1, "fork cut");
}

if (cut_pid == 0) {
    close(fd1[0]);

    if (dup2(fd1[1], 1) == -1) {
        err(1, "dup2 cut");
    }

    if (execlp("cut", "cut", "-d:", "-f7", "/etc/passwd", NULL) == -1) {
        err(1, "exec cut");
    }
}

close(fd1[1]);

if ((sort_pid = fork()) == -1) {
    err(1, "fork sort");
}

if (sort_pid == 0) {
    close(fd2[0]);

    if (dup2(fd1[0], 0) == -1) {
        err(1, "dup2 0 sort");
    }

    if (dup2(fd2[1], 1) == -1) {
        err(1, "dup2 1 sort");
    }

    if (execlp("sort", "sort", "-", NULL) == -1) {
        err(1, "exec sort");
    }
}

close(fd2[1]);

if ((uniq_pid = fork()) == -1) {
    err(1, "fork uniq");
}

if (uniq_pid == 0) {
    close(fd3[0]);

    if (dup2(fd2[0], 0) == -1) {
        err(1, "dup2 2 uniq");
    }
```

```
        if (dup2(fd3[1], 1) == -1) {
            err(1, "dup2 3 uniq");
        }

        if (execlp("uniq", "uniq", "-c", NULL) == -1) {
            err(1, "exec uniq");
        }
    }

    close(fd3[1]);

    if (dup2(fd3[0], 0) == -1) {
        err(1, "dup2 3 sort -n");
    }

    execlp("sort", "sort", "-n", NULL);

    err(1, "exec sort -n");
}
```

Задача 3.

```
#include <stdbool.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <err.h>

#define BUFFER_SIZE 8

static char buffer[BUFFER_SIZE];

struct config_t {
    bool number;
    bool at_start;
    unsigned int line;
};

static bool reader(const int fd, struct config_t* const cfg)
{
    while (1) {
        char ch;

        const ssize_t rd = read(fd, &ch, 1);
        if (rd == -1) {
            return false;
        } else if (rd == 0) {
            return true;
        }

        if (cfg->number && cfg->at_start) {
            int s = snprintf(buffer, BUFFER_SIZE, "%6u ", cfg->line);
            if (s >= BUFFER_SIZE || s < 0) {
                return false;
            }
            if (write(1, &buffer, s) != s) {
                return false;
            }
            cfg->at_start = false;
        }
```

```
        if (write(1, &ch, 1) != 1) {
            return false;
        }

        if (cfg->number && ch == '\n') {
            cfg->line++;
            cfg->at_start = true;
        }
    }
}

int main (const int argc, const char* const argv[])
{

    struct config_t cfg = {
        .number = argc > 1 && strcmp(argv[1], "-n") == 0,
        .at_start = true,
        .line = 1
    };

    int res = 0;

    if (argc > 1 + cfg.number) {
        for (int idx = 1 + cfg.number; idx < argc; idx++) {
            if (strcmp(argv[idx], "-") == 0) {
                if (!reader(0, &cfg)) {
                    warn("Could not process stdin");
                    res = 1;
                }
            } else {
                const int fd = open(argv[idx], O_RDONLY);
                if (fd == -1) {
                    warn("Could not open %s", argv[idx]);
                    res = 1;
                    continue;
                }
                if (!reader(fd, &cfg)) {
                    warn("Could not process %s", argv[idx]);
                    res = 1;
                }
                if (close(fd) == -1) {
                    warn("Could not close %s", argv[idx]);
                    res = 1;
                }
            }
        }
    } else {
        if (!reader(0, &cfg)) {
            warn("Could not process stdin");
            res = 1;
        }
    }

    return res;
}
```

Задача 4.

а) Тръбата е достъпна само чрез файловите дескриптори `pipefd[0]` и `pipefd[1]`. Те са видими само за процеса `P` и неговите наследници. Процесите, които не са наследници на `P`, не могат да ползват тръбата.

б) За да бъде използван от произволен процес в изчислителната среда, комуникационният канал трябва да бъде видим (адресуем, именуван). В повечето UNIX системи има възможност за създаване на именувана тръба (FIFO), тя обаче се създава от един процес и е достъпна за всички останали, тоест нарушава

допълнителното условие на т. (б).
Друг вариант е един процес да създаде именуван обект, който да послужи като адрес при изграждането на връзка от друг процес. Използваната абстракция се нарича socket. Сокетът се дефинира като единия край на комуникационен канал. Един процес, наричан обичайно сървер, изпълнява следната поредица:

```
sfd=socket(domain, type, protocol); // създава socket
bind(sfd, &my_addr, addrlen); // присвоява име на socketa
listen(sfd, backlog); // започва приемане на заявки за връзки
cfd = accept(sfd, &peer_addr, addrlen); // приема заяка за изграждане на връзка
```

Друг процес, наричан обичайно клиент, изпълнява следната поредица:

```
fd=socket(domain, type, protocol); // създава socket
connect(fd, &server_addr, addrlen); // подава заяка за изграждане на връзка
```

Сърверът създава сокета `sfd` и му дава име чрез `bind`. Извикването `listen` активира процеса на изграждане на връзки.
Клиентът създава сокета `fd`, без да е нужно да го именува. Извикването connect е заявка за изграждане на връзка към именувания сокет `sfd`.
Сърверът приема заявката на клиента чрез `accept`. Изградената връзка е между файловите дескриптори `cfd` на сървера и `fd` на клиента. Те ги ползват за обмен на информация.
Файловият дескриптор `sfd` на сървера продължава да приема нови заявки от клиенти. Благодарение на присвоеното му име `sfd` дава възможност на другите процеси да се свържат със сървера.
Името на `sfd` определя какви клиенти могат да ползват сървера. Ако то е име в интернет (IP адрес, порт), всички процеси, изпълнявани на компютри, имащи достъп до интернет, могат да се свържат към сървера.