Поправителен изпит (задачи), част II, за курса “Операционни системи”, СУ, ФМИ, 26.08.2017 г.

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

Задача	5 (20)	6 (20)
получени точки		

Задача 5. Напишете програма на C, която приема четири параметъра – имена на двоични файлове.
Примерно извикване:

```
$ ./main f1.dat f1.idx f2.dat f2.idx
```

Първите два (`f1.dat` и `f1.idx`) и вторите два (`f2.dat` и `f2.idx`) файла са *входен* и *изходен комплект* със следния смисъл:

- `DAT`-файловете (`f1.dat` и `f2.dat`) представляват двоични файлове, състоящи се от байтове (`uint8_t`);
- `IDX`-файловете представляват двоични файлове, състоящи се от наредени тройки от следните елементи (и техните типове), които дефинират поредици от байтове (низове) от съответния `DAT` файл:
  - *отместване* `uint16_t` – показва позицията на първия байт от даден низ спрямо началото на файла;
  - *дължина* `uint8_t` – показва дължината на низа;
  - *запазен* `uint8_t` – не се използва.

Първата двойка файлове (`f1.dat` и `f1.idx`) съществува, а втората трябва да бъде създадена от програмата по следния начин:

- трябва да се копират само низовете (поредици от байтове) от входния комплект, които започват с главна латинска буква (`A - 0x41, Z - 0x5A`).
- ако файловете са неконсистентни по някакъв начин, програмата да прекратява изпълнението си по подходящ начин.

Забележка: За удобство приемаме, че DAT файлът съдържа текстови данни на латински с ASCII кодова таблица (един байт за буква).

Примерен вход и изход:

```
$ xxd f1.dat
00000000: 4c6f 7265 6d20 6970 7375 6d20 646f 6c6f  Lorem ipsum dolo
00000010: 7220 7369 7420 616d 6574 2c20 636f 6e73  r sit amet, cons
00000020: 6563 7465 7475 7220 6164 6970 6973 6369  ectetur adipisci
00000030: 6e67 2065 6c69 742c 2073 6564 2064 6f20  ng elit, sed do
00000040: 6569 7573 6d6f 6420 7465 6d70 6f72 2069  eiusmod tempor i
00000050: 6e63 6964 6964 756e 7420 7574 206c 6162  ncididunt ut lab
00000060: 6f72 6520 6574 2064 6f6c 6f72 6520 6d61  ore et dolore ma
00000070: 676e 6120 616c 6971 7561 2e20 5574 2065  gna aliqua. Ut e
00000080: 6e69 6d20 6164 206d 696e 696d 2076 656e  nim ad minim ven
00000090: 6961 6d2c 2071 7569 7320 6e6f 7374 7275  iam, quis nostru
000000a0: 6420 6578 6572 6369 7461 7469 6f6e 2075  d exercitation u
000000b0: 6c6c 616d 636f 206c 6162 6f72 6973 206e  llamco laboris n
000000c0: 6973 6920 7574 2061 6c69 7175 6970 2065  isi ut aliquip e
000000d0: 7820 6561 2063 6f6d 6d6f 646f 2063 6f6e  x ea commodo con
000000e0: 7365 7175 6174 2e20 4475 6973 2061 7574  sequat. Duis aut
000000f0: 6520 6972 7572 6520 646f 6c6f 7220 696e  e irure dolor in
00000100: 2072 6570 7265 6865 6e64 6572 6974 2069   reprehenderit i
00000110: 6e20 766f 6c75 7074 6174 6520 7665 6c69  n voluptate veli
00000120: 7420 6573 7365 2063 696c 6c75 6d20 646f  t esse cillum do
00000130: 6c6f 7265 2065 7520 6675 6769 6174 206e  lore eu fugiat n
00000140: 756c 6c61 2070 6172 6961 7475 722e 2045  ulla pariatur. E
00000150: 7863 6570 7465 7572 2073 696e 7420 6f63  xcepteur sint oc
```

```
00000160: 6361 6563 6174 2063 7570 6964 6174 6174  caecat cupidatat
00000170: 206e 6f6e 2070 726f 6964 656e 742c 2073   non proident, s
00000180: 756e 7420 696e 2063 756c 7061 2071 7569  unt in culpa qui
00000190: 206f 6666 6963 6961 2064 6573 6572 756e   officia deserun
000001a0: 7420 6d6f 6c6c 6974 2061 6e69 6d20 6964  t mollit anim id
000001b0: 2065 7374 206c 6162 6f72 756d 2e0a        est laborum..

$ xxd f1.idx
00000000: 0000 0500 4f01 0200 4e01 0300            ....O...N...

$ xxd f2.dat
00000000: 4c6f 7265 6d45 78                        LoremEx

$ xxd f2.idx
00000000: 0000 0500 0500 0200                      ........
```

Задача 6. Напишете програма на C, която приема незадължителен параметър – име на команда. Ако не е зададена команда като параметър, да се ползва командата `echo`. Максималната допустима дължина на командата е 4 знака.

Програмата чете низове (с максимална дължина 4 знака) от стандартния си вход, разделени с интервали (`0x20`) или знак за нов ред (`0x0A`). Ако някой низ е с дължина по-голяма от 4 знака, то програмата да терминира със съобщение за грешка.

Подадените на стандартния вход низове програмата трябва да третира като множество от параметри за дефинираната команда. Програмата ви трябва да изпълни командата колкото пъти е необходимо с *максимум два* низа като параметри, като изчаква изпълнението да приключи, преди да започне ново изпълнение.

Примерни вход, извиквания и изходи:

```
$ cat f1
a1
$ cat f2
a2
$ cat f3
a3
$ echo -e "f1\nf2 f3" | ./main cat
a1
a2
a3
$ echo -e "f1\nf2 f3" | ./main
f1 f2
f3
```

Забележки за задачи 5-6:

- Полезни man страници:

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

- Препоръчителни флагове на компилатора: `-std=c99 -Wall -Wpedantic -Wextra`
- Обърнете внимание на коментарите, именуването на променливи и подреждането на кода.

Примерни решения

Задача 5.

```
#include <sys/types.h>
#include <sys/stat.h>

#include <err.h>
#include <errno.h>
#include <fcntl.h>
#include <stdbool.h>
#include <stdint.h>
#include <unistd.h>

int main (const int argc, const char* const argv[])
{
    if (argc != 5) {
        errx(1,
          "Invalid number of arguments. Usage: %s <f1.dat> <f1.idx> <f2.dat> <f2.idx>",
          argv[0]);
    }

    const int f1_dat = open(argv[1], O_RDONLY);
    if (f1_dat == -1) {
        err(1, "Could not open %s", argv[1]);
    }

    const int f1_idx = open(argv[2], O_RDONLY);
    if (f1_idx == -1) {
        const int saved_errno = errno;
        close(f1_dat);
        errno = saved_errno;
        err(1, "Could not open %s", argv[2]);
    }

    const int f2_dat = open(argv[3], O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    if (f2_dat == -1) {
        const int saved_errno = errno;
        close(f1_dat);
        close(f1_idx);
        errno = saved_errno;
        err(1, "Could not open %s", argv[3]);
    }

    const int f2_idx = open(argv[4], O_RDWR | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR);
    if (f2_idx == -1) {
        const int saved_errno = errno;
        close(f1_dat);
        close(f1_idx);
        close(f2_dat);
        errno = saved_errno;
        err(1, "Could not open %s", argv[4]);
    }

    typedef struct {
        uint16_t    off;
        uint8_t     len;
        uint8_t     res;
    } __attribute__((packed)) entry_t;
```

```
entry_t new = { .off = 0, .len = 0, .res = 0, };

while (true) {
    entry_t old;
    if (read(f1_idx, &old, sizeof(old)) != sizeof(old))
        break;

    if (old.len < 1) { continue; }

    if (lseek(f1_dat, old.off, SEEK_SET) == -1) {
        const int saved_errno = errno;
        close(f1_dat);
        close(f1_idx);
        close(f2_dat);
        close(f2_idx);
        errno = saved_errno;
        err(1, "Could not seek to %04x in %s", old.off, argv[1]);
    }

    char buffer;
    bool skip = false;

    for (int idx = 0; idx < old.len; ++idx) {
        if (read(f1_dat, &buffer, sizeof(buffer)) != sizeof(buffer)) {
            const int saved_errno = errno;
            close(f1_dat);
            close(f1_idx);
            close(f2_dat);
            close(f2_idx);
            errno = saved_errno;
            err(1,
              "Could not read byte of set [%04x : %02x] from file %s",
              old.off, old.len, argv[1]);
        }

        if ((idx == 0) && ((buffer < 'A') || (buffer > 'Z'))) {
            skip = true;
            break;
        }

        if (write(f2_dat, &buffer, sizeof(buffer)) != sizeof(buffer)) {
            const int saved_errno = errno;
            close(f1_dat);
            close(f1_idx);
            close(f2_dat);
            close(f2_idx);
            errno = saved_errno;
            err(1,
              "Could not write byte of set [%04x : %02x] to file %s",
              old.off, old.len, argv[3]);
        }
    }

    if (skip) { continue; }

    new.off += new.len;
    new.len = old.len;
```

```
        if (write(f2_idx, &new, sizeof(new)) != sizeof(new)) {
                const int saved_errno = errno;
                close(f1_dat);
                close(f1_idx);
                close(f2_dat);
                close(f2_idx);
                errno = saved_errno;
                err(1,
                  "Could not add new set description to file %s, %s contains non-indexed data.",
                  argv[4], argv[3]);
        }
    }

    close(f1_dat);
    close(f1_idx);
    close(f2_dat);
    close(f2_idx);

    return 0;
}
```

Задача 6.

```
#include <sys/types.h>
#include <sys/wait.h>
#include <err.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

#define MAXPAR 2
#define MAXLEN 4

typedef char par_array[MAXPAR][MAXLEN+1];

static void call(const char* const command, par_array params, const size_t n) {

    const pid_t child_pid = fork();
    if (child_pid == -1) {
        err(1, "cannot fork");
    }

    if (child_pid == 0) {
        if (n == 2) {
            if (execlp(command, command, params[0], params[1], NULL) == -1) {
                err(1, "cannot exec");
            }
        } else if (n == 1) {
            if (execlp(command, command, params[0], NULL) == -1) {
                err(1, "cannot exec");
            }
        }
    }

    int wstatus;
    waitpid(child_pid, &wstatus, 0);
```

```
}

int main (const int argc, const char* const argv[])
{
    if (argc > 2) {
        errx(1, "Usage: %s [optional-command-name]\n", argv[0]);
    }

    if ((argc == 2) && (strlen(argv[1]) > MAXLEN)) {
        errx(1, "Max command-name length: %d", MAXLEN);
    }

    const char* const default_cmd = "echo";
    const char* const cmd = (argc == 2) ? argv[1] : default_cmd;

    par_array par;

    char buffer;

    ssize_t rs = 0;
    size_t curr_par = 0;
    size_t curr_pos = 0;

    bool ready = false;

    while ( (rs = read(0, &buffer, sizeof(buffer))) == sizeof(buffer)) {

        if ((buffer == ' ') || (buffer == '\n')) {
            par[curr_par][curr_pos] = '\0';

            if (curr_pos == 0) {
                if (curr_par == 0)
                    continue;
            } else {
                if (curr_par == 0) {
                    curr_par = 1;
                    curr_pos = 0;
                } else {
                    curr_par = 0;
                    curr_pos = 0;
                    ready = true;
                }
            }
        } else {
            if (curr_pos == MAXLEN) {
                errx(1, "input too long");
            }
            par[curr_par][curr_pos++] = buffer;
        }

        if (ready) {
            ready = false;
            call(cmd, par, MAXPAR);
        }
    }

    par[curr_par][curr_pos] = '\0';
```

```
    if (curr_par != 0 || curr_pos != 0) {
        size_t count = curr_pos != 0 ? curr_par + 1 : curr_par;
        call(cmd, par, count);
    }

    return 0;
}
```