Контролно 3, курс 'Операционни системи', СУ, ФМИ, 18.06.2016 г.

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

Задача	1	2a	2b	3	4	Б	Общо
получени точки							
максимум точки	30	20	20	30	30	30	160

Забележки:

- За отлична оценка са достатъчни 100 точки!
- Препоръчваме програмите да се компилират със следните флагове на компилатора:
  `-std=c99 -Werror -Wall -Wpedantic -Wextra`
- Обърнете внимание на коментарите, именуването на променливи и подреждането на кода

Задача 1.

Напишете програма на C, която приема параметър - име на (двоичен) файл с байтове. Програмата трябва да сортира файла.

Задача 2a.

Напишете програма на C, която по подадено име на (текстови) файл като параметър, извежда съдържанието на файла сортирано, чрез употреба на външните програми cat и sort през pipe().

Задача 2b.

Напишете програма на C, която реализира simple command prompt. Тя изпълнява в цикъл следната поредица действия:

1. Извежда промпт на стандартния изход.
2. Прочита име на команда.
3. Изпълнява без параметри прочетената команда.

Командите се търсят в директорията /bin. За край на програмата се смята въвеждането на exit.

Задача 3.

Двоичните файлове f1 и f2 съдържат 32 битови числа без знак (`uint32_t`). Файлът f1 е съдържа n двойки числа, нека i-тата двойка е $< x_i, y_i >$. Напишете програма, която извлича интервалите с начало x_i и дължина y_i от файла f2 и ги записва залепени в изходен файл f3.

Пример: f1 съдържа 4 числа (2 двойки): `30000 20 19000 10`
Програмата записва в f3 две поредици 32-битови числа, взети от f2 както следва:
Най-напред се записват числата, които са на позиции 30000, 30001, 30002, ... 30019.
След тях се записват числата от позиции 19000, 19001, ... 19009.

Забележка: С пълен брой точки ще се оценяват решения, които работят със скорост, пропорционална на размера на изходния файл f3.

Задача 4.

Опишете разликата между синхронни и асинхронни входно-изходни операции.
Дайте примери за програми, при които се налага използването на асинхронен вход-изход.

Задача Б (бонус).

Забележка: Не се фокусирайте върху задачата, преди да сте приключили с останалите. За тази задача ще се оценяват (частично) и описания с думи (коментари) на схемата на решаване.

Напишете програма на C приемаща параметър – име на (двоичен) файл с `uint32_t` числа. Програмата трябва да сортира файла. Числата биха могли да са максимум 100 000 000 на брой, и са били записани посредством `write(fd, &a, 4);` където `a` е променлива от тип `uint32_t`.

Програмата трябва да работи на машина със същия endianness, както машината, която е създала файла. Програмата трябва да работи на машина с 256 MB RAM и 8 GB свободно дисково пространство.

Упътване за задачи 1–3: Примерни системни извиквания:

```
open()   close()   read()   write()   lseek()   scanf()   fgets()
pipe()   dup2()    fork()   exec()    wait()    waitpid()
```

Решения

Задача 1. Сортиране на файл

```
#include <stdio.h>
#include <stdint.h>
#include <err.h>
#include <errno.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
        if (argc != 2) {
                errx(1, "invalid number of arguments");
        }

        int fd;
        fd = open(argv[1], O_RDWR);
        if (fd == -1) {
                err(1, "%s", argv[1]);
        }

        const size_t buf_size = 4096;
        uint8_t buf[buf_size];

        ssize_t read_size = 0;

        unsigned int stats[256] = { 0 };

        do {
                read_size = read(fd, &buf, sizeof(buf));
                if (read_size < 0) {
                        int saved_errno = errno;
                        close(fd);
                        errno = saved_errno;
                        err(1, "%s", argv[1]);
                }

                for (unsigned int i = 0; i < read_size; i++) {
                        stats[buf[i]]++;
                }

        }
        while (read_size > 0);

        lseek(fd, 0, SEEK_SET);

        for (unsigned int i = 0; i < 256; i++) {
                uint8_t c = i;
                while (stats[i] > 0) {
                        if (write(fd, &c, 1) != 1) {
                                int saved_errno = errno;
                                close(fd);
                                errno = saved_errno;
                                err(1, "Error while writing");
                        }
                        stats[i]--;
                }
        }

        close(fd);
```

```
        exit(0);
}
```

Задача 2a. cat / sort / pipe()

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

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

        if (argc != 2) {
                errx(1, "invalid number of arguments");
        }

        int fd[2];

        pipe(fd);

        pid_t   cat_pid;

        if ((cat_pid = fork()) == -1) {
                err(1, "fork cat");
        }

        if (cat_pid == 0) {
                close(fd[0]);

                if (dup2(fd[1], 1) == -1) {
                        err(1, "dup2 cat");
                }

                if (execlp("cat", "cat", argv[1], NULL) == -1) {
                        err(1, "exec cat");
                }
        }

        close(fd[1]);

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

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

        exit(0);
}
```

Задача 2b. Минималистки шел:

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

int main()
```

```
{
    int waitStatus;
    char cmd[32];
    int i = 0 ;

    while (1) {
        write(1,"> ",2);
        while ( ( read ( 0 , &cmd[i] , 1) ) && cmd[i]  != '\n'  ) {
            if ( cmd[i] == ' ' || cmd[i] == '\t') {
                continue;
            }
            else {
                ++i;
            }
        }
        if ( cmd [i] == '\n' ) {
            cmd[i] = '\0';
        }
        if (  strcmp(cmd,"exit")  == 0 ) {
            exit(0);
        } else {
           if (  fork() ) {
                wait(&waitStatus);
                i=0;
              }
            else {
                if ( execlp(cmd,cmd,NULL) < 0 ) {
                    printf("cannot execute %s\n", cmd);
                    exit(1);
                }
            }
        }
    }   /*  while  */
}
```

Задача 3. Извличане на интервали:

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

int main()
{
  int fd1, fd2, fd3;
  fd1 = open("f1", O_RDONLY);
  if (fd1 < 0) err(1, "Error while opening file f1");
  fd2 = open("f2", O_RDONLY);
  if (fd2 < 0) err(1, "Error while opening file f2");
  fd3 = open("f3", O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
  if (fd3 < 0) err(1, "Error while opening file f3");
  uint32_t x[2], a;

  while (read(fd1, x, sizeof(x)) == sizeof(x)) {
//    printf("%d %d \n",x[0],x[1]);
    off_t lpt;
    lpt=lseek(fd2,x[0]*sizeof(a),SEEK_SET);
    if (lpt < 0) err(1, "lseek error for file f1");
    for (uint32_t i = 0; i < x[1]; i++) {
      read(fd2,&a,sizeof(a));
      write(fd3,&a,sizeof(a));
```

```
//      printf("%d ",a);
    }
  };

  close(fd1);
  close(fd2);
  close(fd3);
}
```

Задача 4.

При синхронна входно-изходна операция системното извикване може да доведе до приспиване (блокиране) на потребителския процес, поръчал операцията.

Същевременно, при нормално завършване, потребителският процес разчита на коректно комлектоване на операцията – четене/запис на всички предоставени/поръчани данни във/от входно-изходния канал, или цялостно изпълнение на друг вид операция (примерно, изграждане на TCP връзка).

При асинхронна входно-изходна операция системното извикване не приспива (не блокира) потребителския процес, поръчал операцията.

Същевременно, при невъзможност да се комплектова операцията, ядрото връща управлението на процеса със специфичен код на грешка и друга информация, която служи за определяне на степента на завършеност на операцията.

Потребителският процес трябва да анализира ситуацията и при нужда да направи ново системно повикване по-късно, с цел да довърши операцията.

Използването на асинхронни операции позволява на един процес да извършва паралелна комуникация по няколко канала с различни устройства или процеси, без да бъде блокиран в случай на липса на входни данни, препълване на буфер за изходни данни или друга ситуация, водеща до блокиране.

Типични примери:

(1) Когато ползваме WEB-browser, той трябва да реагира на входни данни от клавиатура и мишка, както и на данните, постъпващи от интернет, т.е. на поне 3 входни канала. Браусерът проверява чрез асинхронни опити за четене по кой от каналите постъпва информация и реагира адекватно.

(2) Сървер в интернет може да обслужва много на брой клиентски програми, като поддържа отворени TCP връзки към всяка от тях. За да обслужва паралелно клиентите, сървърът трябва да ползва асинхронни операции, за да следи по кои връзки протича информация и кои са пасивни.

Когато програмата ползва асинхронни операции и никой от входно-изходните канали не е готов за обмен на данни, тя има нужда от специален механизъм за предоствяне на изчислителния ресурс на останалите процеси. Обикновено в такива случаи програмата се приспива сама за кратък период от време (в UNIX това става с извикване на sleep(), usleep() или nanosleep()).