Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут”
Кафедра АСОІУ

ЗВІТ

про виконання комп’ютерного практикуму № 4
з дисципліни
“Операційні системи”

Прийняв:
Проф. Сімоненко В. П.

Виконав:
студент 3-го курсу
гр. ІП-52 ФІОТ
Набоков Едуард
Максимович

Київ 2017

ЗМІСТ:

1. ОПИС ІДЕЇ РЕАЛІЗАЦІЇ ФАЙЛОВОЇ СИСТЕМИ

Розроблена файлова систему для пристроїв зберігання інформації блочного типу. Розмір блоку та їх кількість статична. У файловій системі є два типи файлів: звичайні і директорії. Реалізовано однорівневу файлову систему з одного директорією, що містить звичайні файли.

Вміст кожного файлу зберігається в блоках. Для обліку зайнятості блоків використовується біт зайнятості, один біт на один блок. Блоки в структурах файлової системи адресуються за їх порядковими номерами.

Кожен файл, як об'єкт файлової системи, представлений дескриптором. Кількість дескрипторів заздалегідь задається, тому в файловій системі не може бути створено більше певного кількість файлів, навіть якщо є вільне місце. Дескриптор файлу містить таку інформацію: тип файлу (звичайний файл або директорія), кількість посилань на файл, розмір файлу, карта розташування блоків файлу.

Карта розташування блоків має наступну структуру: є кілька прямих посилань на блоки та одне посилання на один блок, що містить прямі посилання на блоки. Номер (позиція) посилання визначає зміщення даних у файлі, посилання нумеруються підряд.

Директорія - це файл, дані якого це масив посилань на файли. Посилання на файл це ім'я файлу і відповідний цьому імені номер дескриптора файлу. Так як ім'я файлу не є частиною дескриптора файлу, то на один файл може бути декілька посилань з різними іменами.

2. ОПИС СТРУКТУР ФАЙЛОВОЇ СИСТЕМИ

- Blocks – клас блоку даних.
    - data – інформація
    - is_used – флаг використання блоку
- filedescriptor – клас дескриптора файла\папки.
    - is_folder – флаг чи являється файл папкою
    - size - розмір файлу
    - links – посилання на Blocks де знаходяться дані файлу
- filesystem – клас, реалізуючий файлову систему.
    - blocks – масив для зберігання даних
    - filedescriptors – список дескрипторыв файлів
    - opened_files – список відкритих файлів
- Main – головний клас програми, де все відбувається.

3. ЛІСТИНГ ПРОГРАМИ

“Block.py”

```
class Block:
    max_block_size = 8
    is_used = None

    def __init__(self):
        self.links = []  # array with int
        self.listOfLinks = {}  # hashmap string , integer
        self.data = ''
        for i in range(self.max_block_size):
            self.data += ' '

    def get_data(self):
        return self.data

    def set_data(self, data):
        self.data = data

    def set_data_with_id(self, id):
        if len(self.links) < self.max_block_size:
            self.links.append(id)
        else:
            print('Sorry, you cannot add more links')

    def get_data_links(self):
        return self.links

    def get_data_list(self):
        return self.listOfLinks

    def set_data_list(self, name, id):
        if len(self.listOfLinks) < self.max_block_size:
            self.listOfLinks[name] = id
        else:
            print('Sorry, you cannot add more links')

    def get_max_block_size(self):
        return self.max_block_size

    def is_used(self):
        return self.is_used
```

```
    def set_is_used(self, is_used):
        self.is_used = is_used
```

“filedescriptor.py”

```
class FileDescriptor:
    max_number_links = 5
    is_folder = None
    size = None

    def __init__(self, is_folder, size):
        self.is_folder = is_folder
        self.size = size
        self.links = []  # array ints

    def get_links(self):
        return self.links

    def add_link(self, link):
        self.links.append(link)

    def get_max_number_links(self):
        return self.max_number_links

    def is_folder(self):
        return self.is_folder

    def set_folder(self, is_folder):
        self.is_folder = is_folder

    def get_number_links(self):
        return len(self.links)

    def get_size(self):
        return self.size

    def set_size(self, size):
        self.size = size
```

“filesystem.py”

```
import filedescriptor as fd
import block as db


class FileSystem:
    quantity_blocks = 8
    max_file_count = 4
    file_count = None

    file_descriptors = {}  # hashmap Integer descriptor
```

```
    openedFiles = []

    def __init__(self):
        self.folder = fd.FileDescriptor(True, 0)
        self.datablocks = [db.Block() for _ in range(self.quantity_blocks)]

    def get_opened_files(self):
        return self.openedFiles

    def get_file_descriptors(self):
        return self.file_descriptors

    def get_datablocks(self):
        return self.datablocks

    def get_max_file_count(self):
        return self.max_file_count

    def get_folder(self):
        return self.folder

    def get_file_count(self):
        return self.file_count

    def set_file_count(self, count):
        self.file_count = count
```

“main.py”

```
import os
import filedescriptor
import filesystem
import block
import re


class Main:

    c = ''
    file_system = None
    descriptor = None

    def __init__(self):
        self.choices = {
            'mount': self.mount,
            'unmount': self.unmount,
            'filestat': self.filestat,
            'create': self.create,
            'ls': self.ls,
            'open': self.open,
            'close': self.close,
            'read': self.read,
            'write': self.write,
            'link': self.link,
            'unlink': self.unlink,
```

```
            'truncate': self.truncate,
            'print': self.print,
        }

    def main(self):
        self.menu()

        choice = input('Enter: ')

        if self.choices.get(choice):
            method = self.choices.get(choice)
            method()

    def mount(self):
        filename = 'temp.txt'
        if os.path.exists(filename):
            self.open(filename)
        else:
            self.file_system = file_system.file_system()

        print(f'File system was mounted. {len(self.file_system.get_datablocks())} blocks')

    def unmount(self, filename):
        self.save(filename)
        print('File system was UN-mounted')

    def filestat(self, id):
        current_file_descriptoriptor = self.file_system.get_file_descriptors().get(id)
        print(f'{current_file_descriptoriptor.get_number_links()} Links:
{current_file_descriptoriptor.get_links()}')

    def create(self, filename):
        file_descriptor = self.get_file_descriptoriptor_by_name(filename)
        if self.file_system.get_file_count() < self.file_system.get_max_file_count():
            n = self.file_system.get_file_descriptors().size()
            descriptor = filedescriptor.filedescriptor(True, 0)

            block = self.get_at_least_one_free_block()

            self.file_system.get_folder().addLink(block)

            self.file_system.get_blocks()[block].set_is_used(True)
            self.file_system.get_blocks()[block].set_data(filename, n)

            print('New file created')
        else:
            print('There is no additional file_descriptoriptors')

    def ls(self):
        for i, link in enumerate(self.file_system.get_folder().getLinks()):
            adrOfBlock = link.get(i)
            h = self.file_system.get_blocks()[adrOfBlock].getDataList()
            print(h)

    def open(self, name):
        file_descriptor = self.get_file_descriptoriptor_by_name(name)
        if file_descriptor != 999:
            self.file_system.get_opened_files().add(file_descriptor)
            print(f'{name} was closed. file_descriptor = {file_descriptor}')
```

```
        else:
            print("This file system doesn't exist in the file system")

    def close(self, name):
        file_descriptor = self.get_file_descriptoriptor_by_name(name)
        if file_descriptor in self.file_system.get_opened_files():
            self.file_system.get_opened_files().pop()
            print(f'{name} was closed. file_descriptor = {file_descriptor}')
        else:
            print('This file was closed')

    def read(self, buf, name):
        file_descriptor = self.get_file_descriptoriptor_by_name(name)
        buf = buf.split('/')
        size = buf.contains(file_descriptor)
        if file_descriptor in self.file_system.get_opened_files():
            d = self.file_system.get_file_descriptors().get(file_descriptor)
            blockSize = self.file_system.get_blocks()[0].get_max_block_size()
            block = d.getLinks()
            data = ''
            for i in range(blockSize):
                data += self.file_system.get_blocks()[block.get(size / blockSize + i)].getData()

        print('Result: ', data)

    def write(self, buf, name):
        if os.path.exists(name):
            file_descriptor = self.get_file_descriptoriptor_by_name(name)
            data = buf
            d = self.file_system.get_file_descriptors().get(file_descriptor)
            block = self.get_at_least_one_free_block()
            blockSize = self.file_system.get_blocks()[0].get_max_block_size()
            localDisp, blockDisp = 0, 0
            required_space = self.file_system.get_blocks().length - block
            size = blockSize % 20
            datasize_blocks = (((localDisp + size) + (size - 1)) / blockSize)
            d.setSize(datasize_blocks)
            d.setFolder(False)
            print('Was written to')
        else:
            print('This file is close')

    def link(self, name_one, name_two):
        block = self.get_at_least_one_free_block()
        self.file_system.get_folder().getLinks().add(block)
        file_descriptor = self.get_file_descriptoriptor_by_name(name_one)
        self.file_system.get_blocks()[block].setUsed(True)
        self.file_system.get_blocks()[block].setData(name_two, file_descriptor)

        print(name_one + " linked with " + name_two)

    def unlink(self, name_one, name_two):
        if os.path.exists(name_one) and os.path.exists(name_two):
            file_descriptor = self.get_file_descriptoriptor_by_name(name_one)
            self.file_system.get_file_descriptors()\
                .get(file_descriptor).setSize(self.file_system.get_file_descriptors()
                .get(file_descriptor).getSize() - 1)

            for linkers in self.file_system.get_folder().getLinks():
```

```
                linkers.remove()

            print('Was unlinked')
        else:
            print('Does not exist')

    def truncate(self, name):
        file_descriptor = self.get_file_descriptoriptor_by_name(name)
        blockSize = self.file_system.get_blocks()[0].get_max_block_size()
        d = self.file_system.get_file_descriptors().get(file_descriptor)
        block = d.getLinks().get(d.getLinks().size() - 1)
        if d.getLinks():
            block = d.getLinks().get(0)

        fileData = ''
        for i, link in enumerate(d.getLinks()):
            fileData += self.file_system.get_blocks()[block + i].getData()

        datasize_blocks = (fileData.length() + (blockSize + 1)) / blockSize
        if datasize_blocks:
            curBlock = block
            for block in reversed(fileData):
                block[curBlock].setUsed(True)
            self.file_system.get_blocks()[curBlock].setData(curBlock)

        self.file_system.get_blocks()[block].setData(fileData)

    def print(self, name):
        for i, block in enumerate(self.file_system.get_blocks()):
            print(f'{i + 1} {file_system.get_blocks()[i].getData()}')

    def menu(self):
        print("Make choice: ")
        print("mount [fs_name] - mount file system [fs_system] from file (if file not exist - create
new)")
        print("umount - save FS to file")
        print("filestat [file_id] - get info about file file_descriptoriptor")
        print("ls - show list of files and their id in FS")
        print("create [name] - create new file")
        print("open [name] - open file")
        print("close [file_descriptor] - close opened file")
        print("read [file_descriptor] [seek] [size] - read information from file in range [seek seek +
size]")
        print("write  [file_descriptor] [seek] [size] - write information into file in range [seek seek +
size]")
        print("link [file] [link_name] - create link with name [link_name] to [file]")
        print("unlink  [link_name] - delete link with name [link_name]")
        print("truncate [file_name] [size] - change size of file to [size]")
        print()

    def getNumberOfSpace(self, name):
        n = 0
        for a in name:
            if a == ' ':
                n += 1

        return n

    def get_file_descriptoriptor_by_name(self, name):
```

```
        file_descriptor = 999
        for i, link in enumerate(self.file_system.get_folder().get_links()):
            adrOfBlock = link.get(i)
            h = self.file_system.get_blocks()[adrOfBlock].getDataList()

            if h.containsKey(name):
                file_descriptor = h.get(name)
                break

        return file_descriptor

    def get_at_least_one_free_block(self):
        c = 0
        for i in range(self.file_system.get_folder()):
            if self.file_system.get_datablocks()[i].is_used():
                c += 1
            else:
                break
        return c

    def open(self, s):
        with open(s, 'r') as opened:
            self.file_system = opened.read()

    def save(self, s):
        with open(s, 'w') as opened:
            opened.write(self.file_system)


if __name__ == '__main__':
    main = Main()
    main.main()
```

4. ПРИКЛАД ВИКОНАННЯ ПРОГРАМИ

```
TypeError: create() missing 1 required positional argument: 'filename'
(petproject_movies) 192:filesystem eduardnabokov$ python main.py
Make choice:
mount [fs_name] - mount file system [fs_system] from file (if file not exist - create new)
umount - save FS to file
filestat [file_id] - get info about file file_descriptoriptor
ls - show list of files and their id in FS
create [name] - create new file
open [name] - open file
close [file_descriptor] - close opened file
read [file_descriptor] [seek] [size] - read information from file in range [seek seek + size]
write  [file_descriptor] [seek] [size] - write information into file in range [seek seek + size]
link [file] [link_name] - create link with name [link_name] to [file]
unlink  [link_name] - delete link with name [link_name]
truncate [file_name] [size] - change size of file to [size]

Enter: create
New file create
(petproject_movies) 192:filesystem eduardnabokov$
```

Робота програми