

Практическая работа №17

Работа с файловой системой и процессами

1 Цель работы

1.1 Научиться разрабатывать приложения для работы с ОС

2 Литература

2.1 Жуков, Р. А. Язык программирования Python. Практикум : учебное пособие /Р. А. Жуков. – Москва : ФОРУМ : ИНФРА-М, 2022. - URL:<https://znanium.com/read?id=395908>. – Режим доступа: для зарегистрир. пользователей. – Текст: электронный. – гл.3.

3 Подготовка к работе

3.1 Повторить теоретический материал (см. п.2).

3.2 Изучить описание практической работы.

4 Основное оборудование**4.1** Персональный компьютер.

5 Задание

5.1 Разработать интерактивный файловый менеджер, отвечающий как за создание, удаление, изменение файлов, так и выполняющий операции поиска - по маске, расширению, дате, размеру;

5.2 Разработать диспетчер задач с выводом в виде списка, позволяющий как запускать программы, так и принудительно их завершать;

5.3 Разработать программу по автоматизации работы с файловой системой, на вход которой поступает файл формата JSON, в котором указываются команды и её аргументы, необходимые для выполнения;

6 Порядок выполнения работы

6.1 Запустить Python IDLE и выполнить все задания из п.5.

6.2 Ответить на контрольные вопросы.

7 Содержание отчета

7.1 Титульный лист

7.2 Цель работы

7.3 Ответы на контрольные вопросы

7.4 Вывод

8 Контрольные вопросы

8.1 Что такое «многопоточность»?

8.2 Какие средства реализации многопоточных приложений имеются в Python?

8.3 Для чего применяются семафоры в многопоточных приложениях?

8.4 Для чего используется модуль thread?

9 Приложение

Для работы с файловой системой в Python используется модуль `os`. Для подключения модуля необходимо воспользоваться директивой `import os`

Наиболее часто используемые функции из модуля `os` перечислены в таблице 1. Полный перечень доступен по ссылке [\[https://docs.python.org/3/library/os.html\]](https://docs.python.org/3/library/os.html)

Таблица 1 - Функции модуля `os`

Название	Описание
<code>listdir</code>	возвращает список, содержащий имена файлов и директорий в каталоге, заданном путем <code>path</code>
<code>chdir</code>	изменяет текущий рабочий каталог
<code>getcwd</code>	вернет строку, представляющую текущий рабочий каталог
<code>mkdir</code>	создает каталог с именем <code>path</code> с режимом доступа к нему <code>mode</code>
<code>makedirs</code>	рекурсивно создает все промежуточные каталоги, если они не существуют
<code>remove</code>	удаляет путь <code>path</code> к файлу
<code>rmdir</code>	удаляет каталоги рекурсивно.
<code>rename</code>	переименовывает файл или каталог с именем <code>src</code> в <code>dst</code>
<code>renames</code>	рекурсивно переименовывает пустые директории или переименовывает конечный файл
<code>replace</code>	переименовывает файл или пустой каталог с исходным именем <code>src</code> в <code>dst</code>

rmdir	удаляет путь к каталогу path
path.exists	проверяет существует ли указанный путь
path.isdir	возвращает True, если путь ведет к каталогу
path.isfile	возвращает True, если путь ведет к файлу

Рассмотрим пример программы, создающей директорию, если она не существует

```
import os
dir = input('введите имя директории: ')
if not os.path.exists(dir):
    os.mkdir(dir)
print(f'директория {dir} существует - {os.path.exists(dir)}')
```

Результат работы программы представлен на рисунке 1

```
введите имя директории: 1
директория 1 существует - True
```

Рисунок 1 - Создание и проверка наличия директории

Рассмотрим работу ещё одного модуля - subprocess, который позволяет создавать новые процессы. При этом он может подключаться к стандартным потокам ввода/вывода/ошибок и получать код возврата.

С помощью subprocess можно, например, выполнять любые команды из скрипта. И в зависимости от ситуации получать вывод или только проверять, что команда выполнилась без ошибок.

Для подключения модуля необходимо воспользоваться директивой `import subprocess`

Главной функцией этого модуля является `run()` - запускает команду с аргументами, ждет завершения команды, а затем возвращает экземпляр `CompletedProcess()` с результатами работы. Полный перечень доступен по ссылке [\[https://docs.python.org/3/library/subprocess.html\]](https://docs.python.org/3/library/subprocess.html)

Рассмотрим пример программы, запускающую командную строку `import subprocess`
`subprocess.run('cmd.exe')`

В результате должна запускаться командная строка (рисунок 2).

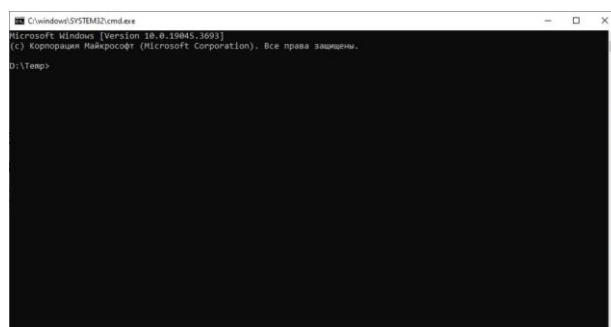


Рисунок 2 - Запуск командной строки

По умолчанию функция `run()` возвращает результат выполнения команды на стандартный поток вывода. Если нужно получить результат выполнения команды, надо добавить аргумент `stdout` и указать ему значение `subprocess.PIPE`

```
import subprocess
result = subprocess.run('dir', shell=True, stdout=subprocess.PIPE) print(result.stdout)
```

В результате будет выведено содержимое текущей директории (рисунок 3)

```
b' \x92\xae\xac \xa2 \xe3\xe1\xe2\xe0\xae\xae\xae\xae\xae \xa5 D \xa8\xac\xa5\xa5
\xe2 \xac\xa5\xe2\xaa\xe3 work\r\n \x91\xa5\xe0\xa8\xa9\xad\xeb\xa9 \xad\xae\xac
\xa5\xe0 \xe2\xae\xac\xa0: 3347-501F\r\n\r\n \x91\xae\xae\xae\xae\xae \xa6\xac\xae
e\xa5 \xaf\xa0\xaf\xaa\xa8 D:\Temp\_2\r\n\r\n29.11.2023 13:50 <DIR>
.\r\n29.11.2023 13:50 <DIR> ..\r\n29.11.2023 13:58
1 1.txt\r\n29.11.2023 13:58 1 2.txt\r\n
4\xa0\xa9\xab\xae\xa2 2 \xa1\xa0\xa9\xe2\r\n 2 \xaf\x
a0\xaf\xae\xa8 289\xff519\xff521\xff792 \xa1\xa0\xa9\xe2 \xe1\xa2\xae\xa1\xae\x
a4\xad\xae\r\n'
```

Рисунок 3 - Вывод содержимого директории

Как видно из рисунка 3 перед строкой вывода указана буква `b`. Это означает, что модуль вернул вывод в виде байтовой строки. Для перевода её в `unicode` есть два варианта:

- выполнить `decode` полученной строки;

- указать аргумент `encoding`. Декодируем в кодировку `cp866`

```
result = subprocess.run('dir', shell=True, stdout=subprocess.PIPE)
print(result.stdout.decode('cp866'))
```

Результат работы программы представлен на рисунке 4

```
Том в устройстве D имеет метку work
Серийный номер тома: 3347-501F

Содержимое папки D:\Temp\_2


29.11.2023 13:50 <DIR> .
29.11.2023 13:50 <DIR> ..
29.11.2023 13:58 1 1.txt
29.11.2023 13:58 1 2.txt
29.11.2023 13:58 2 файлов 2 байт
2 папок 289 519 521 792 байт свободно
```

Рисунок 4 - Вывод содержимого директории

Можно проанализировать завершилась ли команда успешно

```
result = subprocess.run(['ping', '8.8.8.8']) print('alive') if result.returncode==0 else  
print('unreachable')
```

На рисунке 5 показан результат работы программы



unreachable

Рисунок 5 - Результат работы программы

Пакеты были потеряны, поскольку маршрутизатор на пути следования пакетов блокирует icmp. Обратимся к хосту в локальной сети. Результат показан на рисунке 6

```
import subprocess  
result = subprocess.run(['ping', '10.0.0.91']) print('alive') if result.returncode==0 else  
print('unreachable')
```



alive

Рисунок 6 - Результат работы программы