Домашнее задание №3.

Общее описание работы

Существует достаточно большой перечень утилит, которые позволяют анализировать реестр WIndows. Однако, в Вашей работе вы должны самостоятельно разработать программное обеспечение для анализа составляющих операционных систем. Программное обеспечение может быть реализовано как в виде консольной утилиты, так и в виде графического приложения. Программное обеспечение можно разделить на два отдельных приложения

В работе необходимо выполнить 2 части.

Часть 1. Реализация считывания реестра

В программном обеспечении необходимо предусмотреть возможность:

1. считывания реестра при непосредственном запуске утилиты на компьютере
2. считывания реестра при указании на путь до кустов реестра, которые сдампили
3. (опционально) утилита сама сможет выполнить дамп кустов реестра по типу

Часть 2. Считывание данных из реестра

После подключения реестра к программному обеспечению необходимо выполнить анализ и вывести следующую информацию (варианты отображения выбираете сами - в консоль, на экран или в файл - в любо случае информация должны быть представлены в аккуратном структурированном виде)

1. Вся информация о пользователях
2. Вся информация о сетевом стеке
3. Вся информация об операционной системе
4. Вся информация об оборудовании
5. Вся информация о BIOS
6. Информация о программном обеспечении

Пример кода

```
from winreg import OpenKeyEx, QueryValueEx, HKEY_LOCAL_MACHINE, HKEY_USERS, QueryInfoKey,
EnumKey, KEY_READ, EnumValue
import time

const_question = ['1. Вывести информацию о всех пользователях', '2. Вывести информацию о OS',
                  '3. Вывести информацию о BIOS', '4. Вывести информацию о сетевом стеке',
                  '5. Вывести информацию о HARDWARE', '6. Выход']
reserved_users = {'S-1-5-18': 'Local System', 'S-1-5-19': 'Local Service', 'S-1-5-20': 'Network'}
info_users = 1
info_os = 2
info_bios = 3
info_network = 4
info_hardware = 5

const_user_value = {'Volatile Environment': ['APPDATA', 'USERNAME', 'USERDOMAIN', 'LOCALAPPDATA',
'USERPROFILE'], \
                    'Environment': 'all', \
                    r'Control Panel\International': ['LocaleName', 'sShortDate', 'sShortDate', 'sNativeDigits'], \
                    r'Control Panel\Desktop': ['WallPaper']}
const_user_key = {'SOFTWARE': 'all'}

const_os_value = {r'SYSTEM\CurrentControlSet\Control\ComputerName\ComputerName': ['ComputerName'], \
                  r'SYSTEM\CurrentControlSet\Control\Windows': ['ShutdownTime'],
                  r'SOFTWARE\Microsoft\Windows NT\CurrentVersion': ['ProductName', 'EditionID', 'DisplayVersion', \
                                                    'CurrentBuild', 'UBR', 'InstallDate',
                                                    'RegisteredOwner'],
                  r'SYSTEM\CurrentControlSet\Control\TimeZoneInformation': ['TimeZone']}

const_os_key = {'SOFTWARE': 'all'}

const_bios_value = {r'HARDWARE\DESCRIPTION\System': ['SystemBiosVersion', 'SystemBiosDate',
'VideoBiosVersion'],
                    r'HARDWARE\DESCRIPTION\System\BIOS': ['BIOSVendor', 'BIOSVersion', 'BIOSReleaseDate'],
                    r'SYSTEM\CurrentControlSet\Control\SystemInformation': ['BIOSVersion', 'BIOSReleaseDate']}

const_network_value = {
   r'SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\{32a0d524-54cb-44fd-91e0-82c4a74dc41c}': [
      'DhcpIPAddress', 'DhcpNameServer', \
      'DhcpServer', 'DhcpSubnetMask', \
```

```
        'Domain', 'DhcpDefaultGateway']}
const_hardware_value = {
    r'SYSTEM\CurrentControlSet\Control\SystemInformation': ['SystemManufacturer', 'SystemProductName'],
    r'SYSTEM\CurrentControlSet\Control\Class\{4d36e968-e325-11ce-bfc1-08002be10318}\0000': ['DriverDesc']}

const_change = {r'HARDWARE\DESCRIPTION\System\CentralProcessor': ('ProcessorNameString', 'Identifier', \
                                                  'VendorIdentifier', '~MHz'),
                r'HARDWARE\DEVICEMAP\Scsi': ('Identifier', 'SerialNumber')}

const_change_network = {r'SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces': ('DhcpIPAddress',
'DhcpNameServer', 'DhcpServer', 'DhcpSubnetMask', 'Domain', 'DhcpDefaultGateway')}
class Monitor:
    def __init__(self, info_value, info_key):
        self._info_value = info_value
        self._info_key = info_key

    def write_file(self, file_name, operation, write_str):
        with open(file_name, operation, encoding='utf8') as file:
            file.write(write_str)

    def info_str_current(self, const_registry, const_name, user=''):
        info_str = ''
        for value in self._info_value:
            if user in value:
                info_str += self.info_current(const_registry, value, 1, EnumValue, const_name)
        for key in self._info_key:
            if user in key:
                info_str += self.info_current(const_registry, key, 0, EnumKey, const_name)
        return info_str

    def info_current(self, const_registry, key, number, func, name_const):
        try:
            with OpenKeyEx(const_registry, fr'{key}') as connect_registry:
                info_str = ''
                count = 0
                info_registry_value = QueryInfoKey(connect_registry)[number]
                info_str += '\n'
                info_str += fr'{name_const}\{key}'
                info_str += '\n'
                info_str += '\n'
                for i in range(info_registry_value):
```

```
                value = func(connect_registry, i)
                if type(value) == tuple and (
                        value[0] in self._info_value[key] or self._info_value[key] == 'all'):
                    if self._info_value[key] != 'all' and len(self._info_value[key]) == count:
                        return info_str
                    info_str += f'{value[0]} - {value[1]}\n'
                    count += 1
                elif type(value) == str and (value in self._info_key[key] or self._info_key[key] == 'all'):
                    if self._info_key[key] != 'all' and len(self._info_key[key]) == count:
                        return info_str
                    info_str += value + '\n'
                    count += 1
            return info_str
        except OSError as exp:
            info_str = ''
            info_str += '\n'
            info_str += fr'{name_const}\{key}'
            info_str += '\n'
            info_str += 'None\n'
            info_str += f'{exp}'
            info_str += '\n'
            return info_str

    def dump_info(self, current_str='', user='', info='current'):
        file_name = input('Введите путь до текстового файла: ')
        print()
        print('Данная операция полностью перезапишет существующий файл')
        questions = ['Продолжить', 'Назад']
        for i in range(len(questions)):
            print(f'{i + 1}. {questions[i]}')
        answer = int(input('Введите число: '))
        if answer == 1 and info == 'current':
            self.write_file(file_name, 'w', current_str)
        elif answer == 1 and info == 'all':
            with OpenKeyEx(HKEY_USERS, user) as object_current_user:
                path_registry_user = fr'HKEY_USERS\{user}'
                self.write_file(file_name, 'w', f'~~~{path_registry_user}~~~\n\n')
                self.all_info(file_name, object_current_user, path_registry_user)
        print('\nОперация завершена')


class Users(Monitor):
```

```
def __init__(self, info_users_value, info_users_key):
    super().__init__(info_users_value, info_users_key)
    self.not_reserved_users = {}
    self.dump_question = ['Обновить текущую информацию',
                          'Записать эту информацию в файл',
                          'Записать всю информацию о текущем пользователе в файл (займёт некоторое время)']
    self.const_registry = HKEY_USERS

def console_users_question(self):
    while True:
        print()
        for i, key in enumerate(self.not_reserved_users):
            print(f'{i + 1}. Вывести информацию о user - {key}')
        print(f'{i + 2}. Назад')
        number_user = int(input("Введите число: "))
        sid = list(self.not_reserved_users.keys())
        if number_user == i + 2:
            break
        elif sid[number_user - 1] and number_user - 1 >= 0:
            self.print_value_key_user(sid[number_user - 1])
        for i in range(len(self.dump_question)):
            print(f'{i + 1}. {self.dump_question[i]}')
        print(f'{i + 2}. Назад')
        number_dump = int(input("Введите число: "))
        if number_dump == 1:
            self.not_reserved_users[sid[number_user - 1]] = ''
            self.print_value_key_user(sid[number_user - 1])
        elif number_dump == 2:
            self.dump_info(self.not_reserved_users.get(sid[number_user - 1]))
        elif number_dump == 3:
            self.dump_info(user=sid[number_user - 1], info='all')

def all_info(self, file_name, object_registry_user, path_registry_user):
    try:
        info_object = QueryInfoKey(object_registry_user)
        if info_object[1] != 0:
            info_str = path_registry_user + '\n\n'
            for i in range(info_object[1]):
                value = EnumValue(object_registry_user, i)
                info_str += f'{value[0]} - {value[1]}\n'
            info_str += '\n'
```

```
            super().write_file(file_name, 'a', info_str)
        if info_object[0] != 0:
            for i in range(info_object[0]):
                sub_key_current_object = EnumKey(object_registry_user, i)
                with OpenKeyEx(object_registry_user, sub_key_current_object) as connect_registry:
                    self.all_info(file_name, connect_registry, path_registry_user + fr'\{sub_key_current_object}')
    except OSError as exp:
        print(f'{exp}')
        return ''

def __change_user_value_key(self):
    value_dict = {}
    key_dict = {}
    for user in self.not_reserved_users:
        for value in self._info_value.keys():
            value_dict[fr'{user}\{value}'] = self._info_value[value]
        for key in self._info_key.keys():
            key_dict[fr'{user}\{key}'] = self._info_key[key]
    self._info_value = value_dict
    self._info_key = key_dict

def print_value_key_user(self, user):
    print()
    print(f'SID - {user}')
    if not self.not_reserved_users.get(user):
        self.not_reserved_users[user] = super().info_str_current(self.const_registry, 'HKEY_USERS', user)
        print(self.not_reserved_users.get(user))
    else:
        print(self.not_reserved_users.get(user))

def all_users(self):
    with OpenKeyEx(HKEY_LOCAL_MACHINE, r'SOFTWARE\Microsoft\Windows NT\CurrentVersion\ProfileList')
as user_profile:
        for i in range(QueryInfoKey(user_profile)[0]):
            sid_user = EnumKey(user_profile, i)
            user = reserved_users.get(sid_user)
            if user:
                print(f'{sid_user} - {user}')
            else:
                self.not_reserved_users[sid_user] = ''
                print(sid_user)
```

```
        self.__change_user_value_key()
        self.console_users_question()


class Info_All_OS(Monitor):
    def __init__(self, info_value, info_key, method, change={}):
        super().__init__(info_value, info_key)
        self.info_str = ''
        self.method = method
        self.const_registry = HKEY_LOCAL_MACHINE
        self.dump_question = ['Обновить текущую информацию',
                              'Записать эту информацию в файл']
        if change:
            self._info_value.update(self.__change(self.const_registry, change))

    def __change(self, const_registry, change_registry):
        dict_change = {}

        def value_change(connect_registry, reg_name, change_list):
            for elem in change_list:
                try:
                    QueryValueEx(connect_registry, elem)
                except Exception:
                    return {}
            return {reg_name: list(change_list)}

        def inner(const_registry, change_registry, change_list):
            nonlocal dict_change
            try:
                with OpenKeyEx(const_registry, change_registry) as connect_registry:
                    dict_change.update(value_change(connect_registry, change_registry, change_list))
                    count_keys = QueryInfoKey(connect_registry)[0]
                    if count_keys == 0:
                        return
                    else:
                        for i in range(count_keys):
                            reg_sub_name = EnumKey(connect_registry, i)
                            inner(const_registry, change_registry + rf'\{reg_sub_name}', change_list)
            except Exception:
                return {}

        for reg in change_registry:
```

```
            inner(const_registry, reg, change_registry[reg])
        return dict_change

    def work(self):
        self.print_current_info()
        self.console_question()

    def console_question(self):
        while True:
            for i in range(len(self.dump_question)):
                print(f'{i + 1}. {self.dump_question[i]}')
            print(f'{i + 2}. Назад')
            number_dump = int(input("Введите число: "))
            if number_dump == i + 2:
                break
            elif number_dump == 1:
                self.info_str = ''
                self.print_current_info()
            elif number_dump == 2:
                self.dump_info(current_str=self.info_str)

    def print_current_info(self):
        print()
        print(f'~~~{self.method}~~~')
        if not self.info_str:
            self.info_str = super().info_str_current(self.const_registry, 'HKEY_LOCAL_MACHINE')
            print(self.info_str)
        else:
            print(self.info_str)


def main():
    user = Users(const_user_value, const_user_key)
    os = Info_All_OS(const_os_value, const_os_key, 'OS')
    bios = Info_All_OS(const_bios_value, {}, 'BIOS')
    network = Info_All_OS({}, {}, 'NETWORK', const_change_network)
    hardware = Info_All_OS(const_hardware_value, {}, 'HARDWARE', const_change)
    while True:
        print()
        for i in const_question:
            print(i)
        number = int(input('Введите число: '))
```

```
        if number == len(const_question):
            break
        elif number == info_users:
            user.all_users()
        elif number == info_os:
            os.work()
        elif number == info_bios:
            bios.work()
        elif number == info_network:
            network.work()
        elif number == info_hardware:
            hardware.work()


if __name__ == "__main__":
    main()
```