МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С. П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

Отчет по дополнительному заданию
«Операционные системы»

Студент: Дерганов Г.Д.
Группа: 6302-090301D

Проверил: Востокин С. В.

Самара 2023

ЗАДАНИЕ

Разработать систему автоматического управления временными пользователями в среде TLJH (остановка сессии и удаления пользователя по заданным тайм аутам).

Желтым указана информация, которая должна быть изменена.

<user> - имя пользователя

ПОДГОТОВКА

Необходимые требования:

1. Установленная система ubuntu-подобная система (https://ubuntu.com)
2. Установлен клиент сервера TLJH (https://tljh.jupyter.org/en/latest/install/custom-server.html)

TLJH будет поднят локально. Зайти можно по локальному IP. Например, через браузер по localhost будет доступен интерфейс для взаимодействия со средой.

УСТАНОВКА ПРОГРАММЫ

Для установки программы требуется выполнить следующие шаги:

1. Установить все необходимые библиотеки для python3 (psutil)
2. Положить в какую-либо директорию два скрипта (delete_users.py, delete_sessions.py). Например, в /home/<user>/tljh/ (листинг программ в конце отчета).

   ! Важно изменить token в программе на собственный токен, который можно получить пользователю с правами администратора во вкладке Token.
3. Создать файл конфигурации /opt/tljh/config.json с содержимым:

```
{
"time_out_user": 100,
"time_out_sessions": 100
}
```

4. Создать два системных сервиса (службы), с помощью которых программы смогут работать в фоне с перезапуском в случае критических ошибок, а также запускаться при входе в систему (в случае если они не были остановлены при выходе из системы).
   Для это необходимо создать два файла .service в /etc/systemd/system/.
   Например delete-users.service, delete-sessions.service.
   Содержимое службы delete-users.service

```
[Unit]
Description=Delete users
After=syslog.target
After=network.target

[Service]
Type=idle
ExecStart=/usr/bin/python3 /home/<user>/tljh/delete_users.py
Restart=always
RestartSec=15

[Install]
WantedBy=multi-user.target
```

   Аналогичное содержимое и для службы delete-sessions.service

```
[Unit]
Description=Delete sessions
After=syslog.target
After=network.target

[Service]
Type=idle
ExecStart=/usr/bin/python3 /home/<user>/tljh/delete_sessions.py
Restart=always
RestartSec=15

[Install]
WantedBy=multi-user.target
```

   После этого необходимо загрузить и включить их:

   sudo systemctl daemon-reload
   sudo systemctl enable delete-users.service
   sudo systemctl enable delete-sessions.service

5. Создать файл в jupyter notebook в сессии от администратора. Например User-management.ipynb со следующим содержимым (файл User-management.ipynb содержится в архиве прикрепленным к отчету):

```
# Автоматическое управление временными пользователями
#### Скрипт для остановки сессии и удаления пользователя по заданным тайм аутам (время неактивной
```

работы пользователя, время активной работы сеанса). Останавливает сессии и удаляет пользователей с обычными правами (не администраторов)
!!! Перед запуском основных функций, запустить функции обертки и консольные вызовы
Текущие значения
time_out_user, time_out_sessions = get_time_outs()
print(f"Время неактивной работы пользователя {time_out_user}")
print(f"Время длительности сеанса {time_out_sessions}")
Установка значений
new_time_out_user = int(input('Введите время неактивной работы пользователя, после которого он будет удален (сек):'))
new_time_out_sessions = int(input('Введите время длительности сеанса, после которого сеанс принудительно завершится (сек):'))

config = {"time_out_user": new_time_out_user, "time_out_sessions": new_time_out_sessions}
save_config(config)
Перезапуск скрипта
restart()
Запуск скрипта
start()
Остановка скрипта
stop()
Посмотреть статус работы скрипта
print_status()
Функции обертки
import json
import os
file_path = '/opt/tljh/config.json'

```
def restart():
    daemon_reload()
    # user
    stop_delete_users()
    enable_delete_users()
    start_delete_users()
    #session
    stop_delete_sessions()
    enable_delete_sessions()
    start_delete_sessions()

def start():
    start_delete_users()
    start_delete_sessions()

def stop():
    stop_delete_users()
    stop_delete_sessions()

def print_status():
    s = status_delete_users()
    print('\n'.join(s))
    print('-----------------')
    s = status_delete_sessions()
    print('\n'.join(s))

def get_time_outs():
    config = load_config()
    if config is None:
        # Создаем новый конфиг, если файл не найден
        config = {"time_out_user": 600, "time_out_sessions": 600}
        save_config(config)
    time_out_user = config["time_out_user"]
    time_out_sessions = config["time_out_sessions"]
    return time_out_user, time_out_sessions
```

```
def load_config():
    try:
        file_descriptor = os.open(file_path, os.O_RDONLY)
        with os.fdopen(file_descriptor, 'r') as file:
            config = json.load(file)
        return config
    except FileNotFoundError:
        print(f"Файл конфигурации {file_path} не найден.")
        return None

def save_config(config):
    file_descriptor = os.open(file_path, os.O_WRONLY | os.O_CREAT | os.O_TRUNC, 0o666)
    with os.fdopen(file_descriptor, 'w') as file:
        json.dump(config, file, indent=2)
### Консольные вызовы
def daemon_reload():
    !sudo systemctl daemon-reload

def enable_delete_users():
    !sudo systemctl enable delete-users.service

def start_delete_users():
    !sudo systemctl start delete-users.service

def stop_delete_users():
    !sudo systemctl stop delete-users.service

def status_delete_users():
    s = !sudo systemctl status delete-users.service
    return s


def enable_delete_sessions():
    !sudo systemctl enable delete-sessions.service

def start_delete_sessions():
    !sudo systemctl start delete-sessions.service

def stop_delete_sessions():
    !sudo systemctl stop delete-sessions.service

def status_delete_sessions():
    s = !sudo systemctl status delete-sessions.service
    return s
```

ЛИСТИНГ ПРОГРАММ

delete_users.py:

```
import requests
import shutil
import os
from datetime import datetime
from time import sleep
import json
import logging

logging.basicConfig(filename='/home/godji/tljh/history_users.log', level=logging.DEBUG)

url = "localhost" # Домен или айпи
token="87a6c96aaf3f4cad97e2e3b3898df2bc"
```

```
api_url="http://" + url + "/hub/api"

class user:
    def __init__(self, user_name, user_user_is_admin, user_latest_time_online):
        self.user_name=user_name
        self.user_is_admin=user_user_is_admin
        self.user_latest_time_online=user_latest_time_online

    def getName(self):
        return self.user_name

    def getUserIsAdmin(self):
        return self.user_is_admin

    def getUserLatestTimeOnline(self):
        return self.user_latest_time_online

# returns full list of info about users
def getInfoList():
    users_list_req=requests.get(api_url+'/users', headers={'Authorization': f'token {token}', })
    users_list_req.raise_for_status()
    usersInfoList=users_list_req.json()
    return usersInfoList

# returns list of all users
def getUsersList(usersInfoList):
    users_list=[]
    for i in range(len(usersInfoList)):
        user_name=usersInfoList[i]['name']

        user_latest_time_online=usersInfoList[i]['last_activity']
        if ('admin' in usersInfoList[i]['roles']):
            user_is_admin=1
        else:
            user_is_admin=0
            users_list.append(user(user_name, user_is_admin, user_latest_time_online))
    return users_list

# deletes unactive users
def deleteUnactiveUsers(usersList, time_out_user):
    curr_datetime=datetime.utcnow()
    for i in range(len(usersList)):
        if (usersList[i].getUserIsAdmin()!=1) and (str(usersList[i].getUserLatestTimeOnline())!="None"):
            user_time= datetime.strptime(str(usersList[i].getUserLatestTimeOnline()), '%Y-%m-%dT%H:%M:%S.%fz' )
            if (abs(user_time.timestamp()-curr_datetime.timestamp())>time_out_user):
                name=str(usersList[i].getName())
                delete_user_req=requests.delete(api_url+'/users/'+name, headers={'Authorization': f'token {token}', })
                delete_user_req.raise_for_status()
                if os.path.isdir("/home/jupiter-" + name):
                    os.system(f"sudo rm -r /home/jupyter-{name}")
                logging.info(f"deleted {name}!")
    logging.info("-------")
# prints all active users

def printUsers(usersList):
    for i in range(len(usersList)):
        logging.info(f"Name: {usersList[i].getName()}, Has Admin: {usersList[i].getUserIsAdmin()}, Latest Activity:
{usersList[i].getUserLatestTimeOnline()}")

def load_config(file_path="/opt/tljh/config.json"):
    try:
        with open(file_path, "r") as file:
            config = json.load(file)
        return config
    except FileNotFoundError:
        print(f"Файл конфигурации {file_path} не найден.")
```

```
        return None

def save_config(config, file_path="/opt/tljh/config.json"):
    with open(file_path, "w") as file:
        json.dump(config, file, indent=2)

def get_time_outs():
    config = load_config()
    if config is None:
        # Создаем новый конфиг, если файл не найден
        config = {"time_out_user": 600, "time_out_sessions": 600}
        save_config(config)
    time_out_user = config["time_out_user"]
    time_out_sessions = config["time_out_sessions"]
    return time_out_user, time_out_sessions

def main():
    time_out_user, time_out_sessions = get_time_outs()
    logging.info(f"time_out_user: {time_out_user}")
    logging.info(f"time_out_sessions: {time_out_sessions}")

    users_list=getUsersList(getInfoList())
    json_formatted_str=json.dumps(getInfoList(), indent=2)
    logging.info(json_formatted_str)
    while True:
        users_list=getUsersList(getInfoList())
        deleteUnactiveUsers(users_list, time_out_user)
        sleep(10)

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

delete_sessions.py:

```
import os
import time
import psutil
import requests
import logging
import json
from time import sleep

logging.basicConfig(filename='/home/godji/tljh/history_sessions.log', level=logging.DEBUG)

url = "localhost" # Домен или айпи
token = "87a6c96aaf3f4cad97e2e3b3898df2bc" # Токен
hub_url = "http://" + url + "/hub/api"

def get_jupyter_sessions(hub_url, token):
    jupyter_not_admin_sessions = []
    for process in psutil.process_iter(['pid', 'username']):
        username = process.info["username"]
        if username.startswith("jupyter-") and not is_admin_user(username[8:], hub_url, token):
            jupyter_not_admin_sessions.append(process.info['pid'])
            return jupyter_not_admin_sessions

def is_admin_user(username, hub_url, token):
    try:
        response = requests.get(f"{hub_url}/users/{username}", headers={"Authorization": f"token {token}", })
        response.raise_for_status()
        if response.status_code == 200:
            return response.json().get("admin")
        return True
    except Exception as e:
        logging.info(f"Network error: {e}")
    return True
```

```
def load_config(file_path="/opt/tljh/config.json"):
    try:
        with open(file_path, "r") as file:
            config = json.load(file)
        return config
    except FileNotFoundError:
        print(f"Файл конфигурации {file_path} не найден.")
        return None

def save_config(config, file_path="/opt/tljh/config.json"):
    with open(file_path, "w") as file:
        json.dump(config, file, indent=2)

def get_time_outs():
    config = load_config()
    if config is None:
        # Создаем новый конфиг, если файл не найден
        config = {"time_out_user": 600, "time_out_sessions": 600}
        save_config(config)
    time_out_user = config["time_out_user"]
    time_out_sessions = config["time_out_sessions"]
    return time_out_user, time_out_sessions

def delete_sessions(time_out_sessions):
    jupyter_not_admin_sessions = get_jupyter_sessions(hub_url, token)
    if jupyter_not_admin_sessions != None:
        for session_pid in jupyter_not_admin_sessions:
            session_start_time = os.path.getctime(f'/proc/{session_pid}')
        elapsed_time = time.time() - session_start_time
        if elapsed_time > time_out_sessions:
            os.system(f"sudo kill {session_pid}")
            logging.info(f"{session_pid} disabled")

def main():
    time_out_user, time_out_sessions = get_time_outs()
    logging.info(f"time_out_user: {time_out_user}")
    logging.info(f"time_out_sessions: {time_out_sessions}") # Максимальное время сеанса в секундах

    while(True):
        delete_sessions(time_out_sessions)
        sleep(10)

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

ВЫВОД

Был разработан скрипт, управляющий службами по запуску/остановке сессий и удаляющий временных пользователей по заданным таймаутам.

РАБОТА ПРОГРАММЫ

При работе скриптов, каждый из них сохраняет лог файл в директории, где расположены сами скрипты (по примеру /home/user/tljh/history_users.log, history_sessions.log)

В интерактивном окне можно задать значения таймаутов, посмотреть их, перезапустить выполнение служб, запустить, остановить, посмотреть статус их работы (активны или нет).

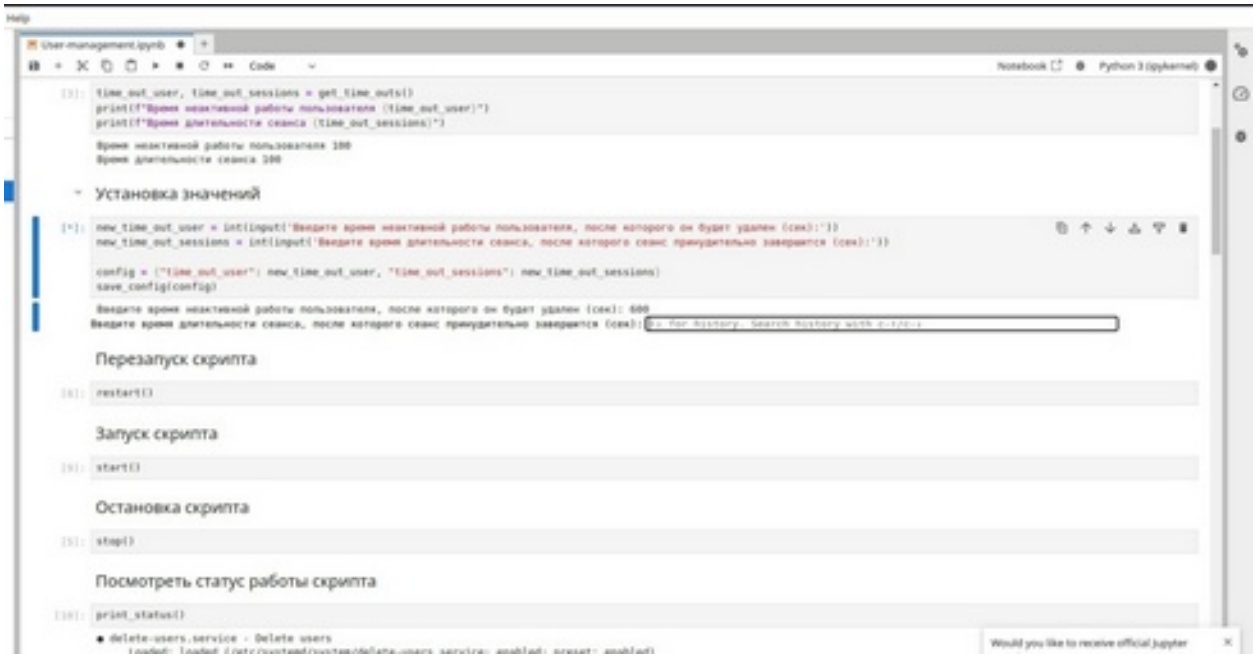

Рисунок 1. Установка значений таймаута в файл конфигурации

УДАЛЕНИЕ ПОЛЬЗОВАТЕЛЯ

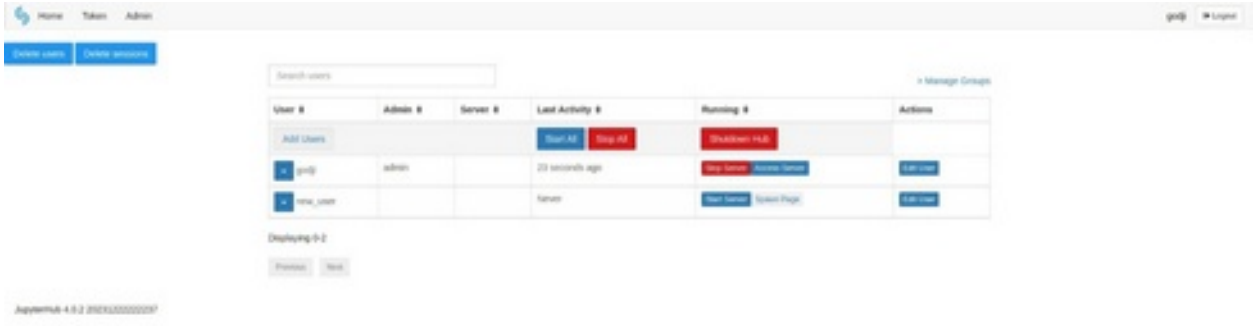

Рисунок 2. Добавили нового пользователя new_user

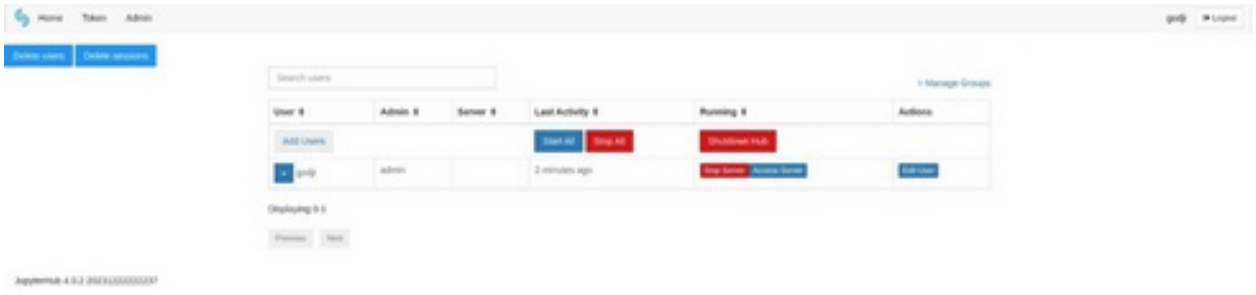

Рисунок 3. Пользователь был удален через заданное кол-во секунд

ОСТАНОВКА СЕССИИ

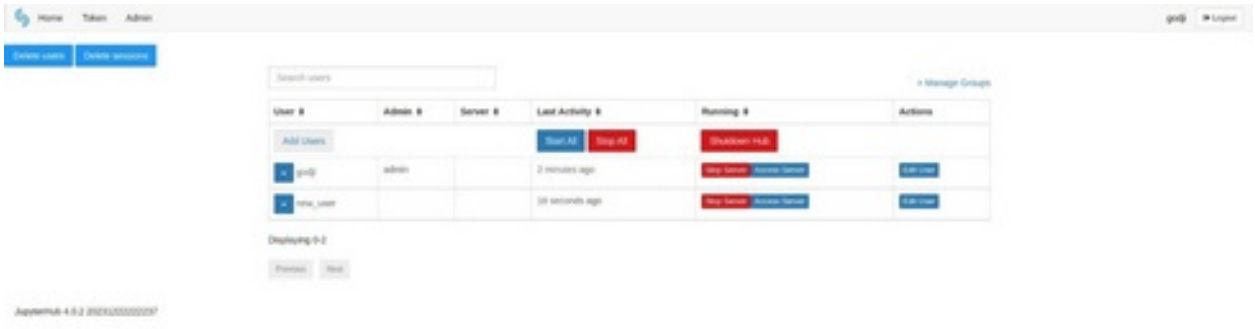

Рисунок 4. Добавлен new_user и запущен сервер

Рисунок 5. Окно входа нового пользователя

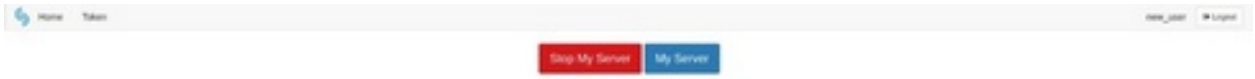

Рисунок 6. Сервер запущен

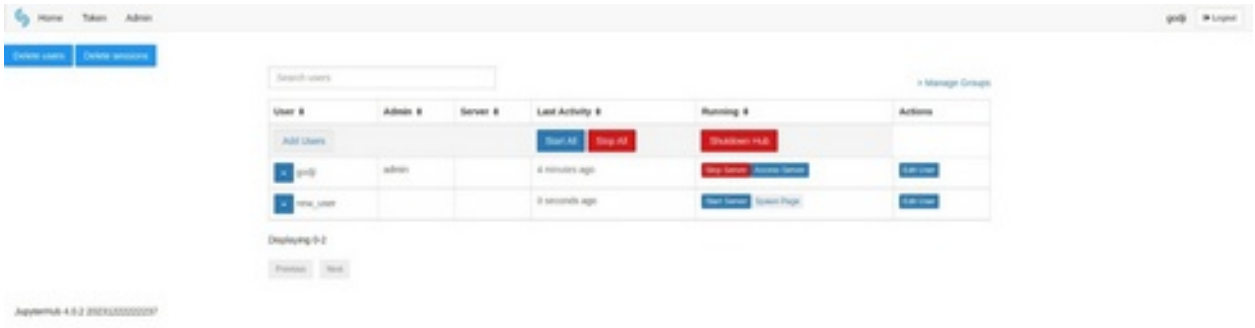

Рисунок 7. Спустя заданное кол-во секунд сессия останавливается

Рисунок 8. Что видит пользователь new_user, когда сервер был остановлен