Trabalho Prático (T)

Identificação da Disciplina

Código da Disciplina	Nome da Disciplina	Professor	Período
CIC_4MA e EGS_4MA	Redes de Computadores e Internet	Jeremias Moreira Gomes	2023/2

1. Objetivo

O objetivo deste trabalho é capacitar o aluno a estudar e compreender conceitos envolvendo rede de computadores e internet. Utilizando o conhecimento adquiridos nas aulas, o aluno deverá implementar um servidor de compartilhamento de arquivos com princípios básicos da arquitetura P2P (peer-to-peer), utilizando programação em sockets.

Ao cumprir os objetivos do trabalho, o aluno terá adquirido: (*i*) uma compreensão prática dos conceitos e características-chave de redes de computadores e internet; (*ii*) maior aprimoramento das suas habilidades de escrita técnica, organização de informações e comunicação eficaz, uma vez que a documentação de implementação exigirá uma apresentação clara e estruturada das principais características da rede de computadores e internet; e (*iii*) essa experiência proporcionará maior confiança e capacidade para explorar esses conceitos na solução de problemas computacionais.

2. Enunciado

A Interoperabilidade e Desenvolvimento de Protocolos (IDP) é uma organização sem fins lucrativos que tem como objetivo criar e desenvolver padrões abertos para a Internet.

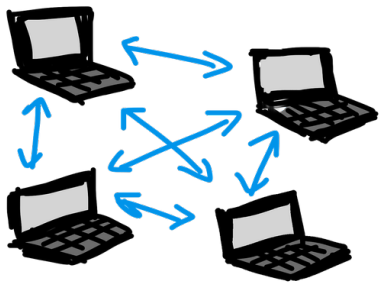
Visando a criação de um novo protocolo de compartilhamento de arquivos, a IDP está contratando novos funcionários que possuam experiência de programação em sockets, ou que sejam capazes de aprender e implementar esse conhecimento.

Como novo contratado da IDP, você recebeu a especificação do protocolo IDP2P, que é um protocolo simplificado de compartilhamento de arquivos. Nesse protocolo, existem dois tipos de nós: **servidor** e **clientes**. Servidores tem a única responsabilidade de conectar clientes. Já clientes podem se conectar a outros clientes e compartilhar arquivos entre si. A seção seguinte contém a especificação completa do protocolo.

Apesar do uso de sistemas P2P ter sido popularizado por distribuição de conteúdo ilegal, como filmes e músicas, a distribuição de conteúdo P2P é usada para implementar aplicativos e modelos de negócios legítimos que se aproveitam de características do protocolo como escalabilidade, tolerância a falhas e descentralização.

3. IDP2P

O protocolo IDP2P é um protocolo simplificado de compartilhamento de arquivos que permite que clientes se conectem a um servidor e compartilhem arquivos entre si. O protocolo é baseado em troca de mensagens,



que são enviadas entre os nós da rede. No funcionamento da rede, existem dois tipos de nós: **servidor**, que é um **único** nó responsável por conectar clientes; e **clientes**, que são nós que podem se conectar a outros clientes e compartilhar arquivos entre si.

3.1. Servidor

O servidor é o nó responsável por conectar clientes. Ele é o primeiro nó a fazer parte da rede e é o único nó que não pode ser desconectado. **Toda a comunicação entre o servidor e os clientes é feita utilizando o protocolo UDP, por meio da porta 54494**.

Ele deve manter uma lista de clientes conectados, que é atualizada sempre que um novo cliente se conecta ou um cliente se desconecta. Além disso, deve-se manter também uma lista de arquivos compartilhados por todos os clientes conectados, controlando quais clientes possuem cada arquivo.

O servidor reage a quatro tipos de mensagens: `REG`, `UPD`, `LST`, e `END`. Se o servidor receber uma mensagem de um tipo diferente desses ou caso a mensagem seja de um tipo válido, porém com formato inválido, o servidor deve responder com a mensagem:

```
ERR INVALID_MESSAGE_FORMAT
```

3.1.1. REG

Mensagem de registro de um novo cliente. O formato da mensagem é[1]

```
REG <SENHA> <PORTA> <ARQUIVOS>
```

SENHA é uma palavra secreta qualquer, composta por letras e números, escolhida pelo cliente.

PORTA é a porta TCP que o cliente estará disponível para enviar arquivos para outros clientes.

ARQUIVOS é uma lista de arquivos que o cliente deseja compartilhar com outros clientes. O formato dessa lista é

```
MD51,NOME1;MD52,NOME2;MD53,NOME3;...;MD5N,NOMEN
```

[1] Os caracteres >< não fazem parte da mensagem.

onde *MD5i* é o hash MD5[2] do arquivo *NOMEi*. O hash do arquivo e seu respectivo nome são separados por vírgula, e múltiplos arquivos são separados por ponto e vírgula.

O formato da mensagem de resposta é

```
OK <N>_REGISTERED_FILES
```

onde *N* é o número de arquivos que o cliente compartilha com outros clientes.

3.1.2. UPD

Mensagem de atualização de arquivos e porta de um cliente. O formato da mensagem é:

```
UPD <SENHA> <PORTA> <ARQUIVOS>
```

SENHA é a palavra secreta, composta por letras e números, utilizada pelo cliente, quando o mesmo se registrou.

PORTA é a porta TCP que o cliente estará disponível para enviar arquivos para outros clientes.

ARQUIVOS é uma lista de arquivos que o cliente deseja compartilhar com outros clientes. O formato dessa lista é:

```
MD51,NOME1;MD52,NOME2;MD53,NOME3;...;MD5N,NOMEN
```

Há dois tipos de mensagens de resposta, o primeiro quando a senha está correta:

```
OK <N>_REGISTERED_FILES
```

e o segundo quando a senha está incorreta:

```
ERR IP_REGISTERED_WITH_DIFFERENT_PASSWORD
```

3.1.3. LST

Mensagem de listagem de arquivos. O formato da mensagem é:

```
LST
```

O formato da mensagem de resposta é

```
MD51,NOME1,IP1:PORTA1,IP2:PORTA2,...,IPN:PORTAN;MD52,NOME2,IP1:PORTA1,...
```

que é uma string única, sem espaços, onde cada arquivo é separado por ponto e vírgula. Cada arquivo é composto pelo hash MD5 do arquivo, seu nome, e uma lista de IPs e portas, separados por dois pontos, de clientes que possuem o arquivo.

[2] A linguagem python possui, por padrão, a biblioteca `hashlib` que calcula a função de dispersão MD5 (e várias outras) para conjuntos de bytes.

3.1.4. END

Mensagem de desconexão de um cliente, informando ao servidor que o cliente não estará mais disponível para compartilhar arquivos. O formato da mensagem é:

```
END <SENHA> <PORTA>
```

em que *SENHA* é a palavra secreta, composta por letras e números, utilizada pelo cliente, quando o mesmo se registrou, e *PORTA* é a porta TCP que o cliente estava disponível para enviar arquivos para outros clientes.

Há dois tipos de mensagens de resposta, o primeiro quando a senha está correta:

```
OK CLIENT_FINISHED
```

e o segundo quando a senha está incorreta:

```
ERR IP_REGISTERED_WITH_DIFFERENT_PASSWORD
```

3.2. Cliente

O cliente é o nó responsável por se conectar a outros clientes e compartilhar arquivos entre si. Ele pode se conectar a outros clientes e baixar arquivos compartilhados por eles. **Toda a comunicação entre clientes é feita utilizando o protocolo TCP, por meio da porta que o cliente informou ao servidor**.

O cliente, responde a um tipo de mensagem: `GET`. Se o servidor receber uma mensagem inválida ou incorreta, ele deve apenas ignorar a mensagem e encerrar a conexão com o outro cliente.

3.2.1. GET

Mensagem de requisição de um arquivo. O formato da mensagem é:

```
GET <MD5>
```

em que *MD5* é o hash MD5 do arquivo que o cliente deseja baixar, pertencente a um dos clientes conectados.

A mensagem de resposta, para uma mensagem de requisição válida, é o conjunto de bytes do arquivo solicitado.

4. Implementação

A implementação do protocolo IDP2P deve ser feita em Python 3.8 ou superior. As soluções devem ser divididas entre servidor e cliente, e devem ser implementadas em arquivos separados.

O arquivo principal do servidor (pode ser que haja mais de um arquivo) deve se chamar `servidor.py`. A execução do servidor deve ser feita por meio do seguinte comando:

```
python3 servidor.py
```

Já o cliente possui um funcionamento mais complexo, pois será a partir dele que o usuário irá interagir com a rede. O arquivo principal do cliente (pode ser que haja mais de um arquivo) deve se chamar `cliente.py`. A execução do cliente deve ser feita por meio do seguinte comando:

```
python3 cliente.py <IP> <DIRETORIO>
```

onde *IP* é o endereço IP do servidor e *DIRETORIO* é o diretório que possui arquivos os quais o cliente deseja compartilhar com outros clientes, além de ser o local onde os arquivos baixados serão salvos.

Ao executar o cliente, este deverá realizar as seguintes ações:

- Descobrir uma porta TCP disponiível para aceitar conexões de outros clientes.
- Decidir qual será sua própria senha para se registrar no servidor.
- Listar os arquivos que deseja compartilhar com outros clientes.
- Enviar uma mensagem de registro para o servidor.
- Paralelamente:
  - Aceitar conexões TCP de outros clientes, para envio de arquivos, na porta informada ao servidor.
  - Interagir com o usuário, recebendo comandos de:
    - Listagem de arquivos disponíveis na rede.
    - Download de um arquivo.
    - Desconexão do cliente.

Como o funcionamento do cliente possui atividades que devem ser executadas paralelamente, é necessário o uso de *threads*. Como o foco do trabalho é o protocolo de compartilhamento de arquivos, será disponibilizado um arquivo `cliente_base.py` que possui uma implementação básica de um cliente que executa as duas atividades paralelas (TCP e UDP) necessárias.

O arquivo `cliente_base.py` pode ser acessado no link:

```
https://gist.github.com/j3r3mias/334627c46a7fc721b6f50aa00c4832c6
```

Ao final deste documento, há uma versão compacta do mesmo arquivo.

Na parte de interação com o usuário, a opção de listagem de arquivos deve mostrar a lista de arquivos disponíveis por nome. A Figura 1 contém um exemplo de listagem.

Já para baixar um arquivo, o usuário deverá escolher o número do índice do arquivo que deseja baixar.

4.1. Sumário

Ao final do trabalho, o aluno deverá entregar a implementação do protocolo, tanto o cliente quanto o servidor.

5. Metodologia

Este trabalho será dividido em três partes: (*i*) Implementação do Servidor; (*ii*) Implementação do Cliente; e (*iii*) Relatório Técnico.

```
[55017] j3r3mias@tardis:trabalho|main > python3 cliente.py 192.168.0.225 arquivos-03
[*] Procurando porta disponível...
[!] Porta 20000 indisponível!
[*] Porta 20001 disponível!
[+] Utilizando porta 20001.
[*] Senha do cliente: f2940e687645369f
MENU:
    1 - Registrar no servidor
    2 - Listar arquivos
    3 - Baixar arquivo
    4 - Remover registro do servidor
    5 - Sair

Escolha uma opção: 2
[*] Arquivos disponíveis:
[*]   1 -   3 peers - idp.png    - ['192.168.0.144:20000', '192.168.0.31:20000', '192.168.0.144:20001']
[*]   2 -   3 peers - pudim.jpg  - ['192.168.0.144:20000', '192.168.0.31:20000', '192.168.0.144:20001']
[*]   3 -   2 peers - arquivo-texto-01.txt - ['192.168.0.31:20001', '192.168.0.31:20000']
[*]   4 -   1 peers - arquivo-texto-02.txt - ['192.168.0.31:20001']
[*]   5 -   1 peers - arquivo-texto-03.txt - ['192.168.0.31:20001']

MENU:
    1 - Registrar no servidor
    2 - Listar arquivos
    3 - Baixar arquivo
    4 - Remover registro do servidor
    5 - Sair

Escolha uma opção:
```

Figura 1. Exemplo de listagem de arquivos

5.1. Implementação do Servidor (45 pts)

Nesta etapa, os alunos deverão implementar o servidor do protocolo IDP2P. O servidor deve ser capaz de realizar e responder corretamente às mensagens de registro, atualização, listagem e desconexão de clientes.

5.2. Implementação do Cliente (45 pts)

Nesta etapa, os alunos deverão implementar o cliente do protocolo IDP2P. O cliente deve ser capaz de se conectar no servidor, receber conexões de outros clientes, para enviar arquivos, e interagir com o usuário, recebendo respostas de requisições do servidor e baixando arquivos de outros clientes, quando solicitado.

5.3. Documentação (10 pts)

Nesta etapa, os alunos deverão documentar a implementação do servidor e do cliente, de forma a permitir que outras pessoas possam compreender o funcionamento do protocolo e a implementação realizada. A documentação deverá ser concisa e objetiva, não deixando de ser clara, contendo no máximo **duas páginas**.

6. Detalhes Acerca do Trabalho

6.1. Restrições deste Trabalho

Esta atividade poderá ser realizada em até **três integrantes.**

6.2. Datas Importantes

Esse trabalho poderá ser submetido a partir de 06/11/2023 (segunda-feira) - 12:00h até 12/12/2023 (terça-feira) - 23:55h.

6.3. Por onde será a entrega o Trabalho?

O trabalho deverá ser entregue via Ambiente Virtual (https://ambientevirtual.idp.edu.br/), na disciplina de Redes de Computadores e Internet. Na disciplina haverá uma atividade chamada “Trabalho Prático (T)”, para submissão do trabalho.

6.4. O que deverá ser entregue, referente ao trabalho?

Deverá ser entregue a documentação e os códigos produzidos pelos alunos, relacionado a resolução do trabalho.

6.5. Como entregar o trabalho?

A submissão das atividades deverá ser feita em um arquivo único comprimido do tipo zip (`Zip archive data, at least v1.0 to extract`), contendo documentação e um diretório com os códigos elaborados. Além disso, para garantir a integridade do conteúdo entregue, o nome do arquivo comprimido deverá possuir duas informações (além da extensão `.zip`):

- As matrículas dos alunos, separada por hífen.
- O *hash* md5 do arquivo e `.zip`.

Exemplo: `23001100-23011000-23010203-3b1b448e9a49cd6a91a1ad044e67fff2.zip`

Para gerar o md5 do arquivo comprimido, utilize o comando `md5sum` do Linux e em seguida faça o renomeamento utilizando o *hash* coletado.

Segue um exemplo de geração do arquivo final a ser submetido no moodle, para um aluno:

```
[13930] j3r3mias@tardis:trabalho-01 > ls
trabalho-01-apc-jeremias.c
[13931] j3r3mias@tardis:trabalho-01 > zip -r aaa.zip trabalho-01-apc-jeremias.c
  adding: trabalho-01-apc-jeremias.c (deflated 99%)
[13932] j3r3mias@tardis:trabalho-01 > ls
aaa.zip  trabalho-01-apc-jeremias.c
[13933] j3r3mias@tardis:trabalho-01 > ls -lha aaa.zip
-rw-rw-r-- 1 j3r3mias j3r3mias 335 out 15 16:54 aaa.zip
[13934] j3r3mias@tardis:trabalho-01 > md5sum aaa.zip
44cf46f9237cb506ebde3e9e1cd044f9  aaa.zip
[13935] j3r3mias@tardis:trabalho-01 > mv aaa.zip 160068000-44cf46f9237cb506ebde3e9e1cd044f9.zip
[13936] j3r3mias@tardis:trabalho-01 > ls -lha 160068000-44cf46f9237cb506ebde3e9e1cd044f9.zip
-rw-rw-r-- 1 j3r3mias j3r3mias 335 out 15 16:54 160068000-44cf46f9237cb506ebde3e9e1cd044f9.zip
[13937] j3r3mias@tardis:trabalho-01 > md5sum 160068000-44cf46f9237cb506ebde3e9e1cd044f9.zip
44cf46f9237cb506ebde3e9e1cd044f9  160068000-44cf46f9237cb506ebde3e9e1cd044f9.zip
[13938] j3r3mias@tardis:trabalho-01 >
```

6.6. Quem deverá entregar o trabalho?

O Ambiente Virtual disponibiliza uma opção de trabalhos em grupo, onde um membro do grupo realiza a submissão e o arquivo é disponibilizado para ambos. Caso esta opção não esteja disponível, ambos os alunos da dupla deverão submeter o trabalho no Ambiente Virtual onde, **somente um aluno deverá gerar a versão final do trabalho (arquivo zip)**, para que o *hash* do arquivo não corra o risco de ficar diferente.

6.7. Observações importantes

- **Não deixe para fazer o trabalho de última hora**.
- Não serão aceitos trabalhos com atraso.
- **Não deixe para fazer o trabalho de última hora**.
- Cópias de trabalho receberão zero (além disso, plágio é crime).
- **Não deixe para fazer o trabalho de última hora**.

7. Bibliografia

1. KUROSE, James F.; ROSS, Keith W. Redes de computadores, uma abordagem top-down. 2013.

Código do Cliente Base

```
#!/usr/bin/python

import time
import socket
from _thread import *

porta_tcp = None

def configurar_ambiente():
    pass


def descobre_porta_disponivel():
    return 31337


def controle_udp():
    # Codigo do servico UDP
    while True:
        print('Controle UDP funcionando')
        time.sleep(5)


def servico_tcp(client):
    # Codigo do servico TCP
    print('Nova conexao TCP')
    client.send(b'OI')
    client.close()


def controle_tcp():
    global porta_tcp
    _socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    _socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    _socket.bind(('', porta_tcp))
    _socket.listen(4096)
    while True:
        client, addr = _socket.accept()
        start_new_thread(servico_tcp, (client, ))


def inicia_controle_tcp():
    controle_tcp()


def inicia_controle_udp():
    controle_udp()


def main():
    global porta_tcp
    porta_tcp = descobre_porta_disponivel()

    configurar_ambiente()

    start_new_thread(inicia_controle_tcp, ())
    start_new_thread(inicia_controle_udp, ())

    while True:
        time.sleep(60)
        print('Cliente em execucao')


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