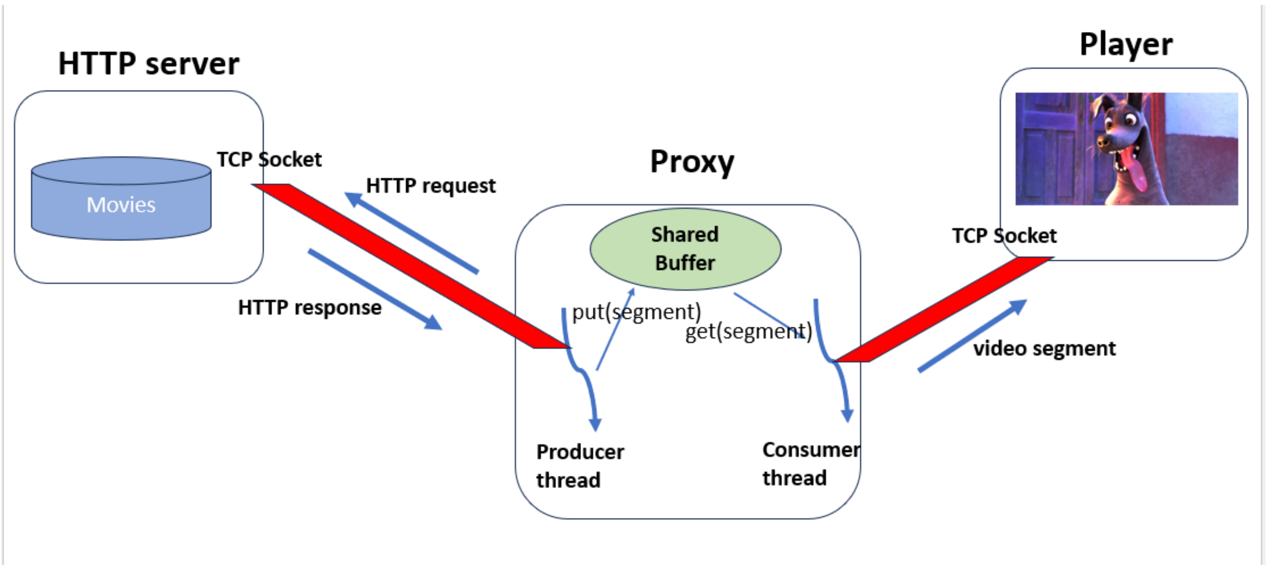
NOVA SCHOOL OF SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

Video streaming and HTTP partial requests (TPC4 – Redes de Computadores)

This assignment concerns the problem of video streaming in a networking environment where available bandwidth and roundtrip times are changing. The application to be built is inspired by the DASH approach, is composed by three processes that can run in distinct machines:

- **Content server**: it is a standard http server that contains movies to be played. As in the DASH approach, for the same movie more than one version is available; versions are called *tracks* and are described in a file called *manifest.txt*. Videos are in MP4 format. Each track is divided in *segments* all with the same duration
- **Player**: shows the movie. Receives the byte stream through a socket TP.
- **Proxy**: Has two parts each executed by a thread; the two threads share a queue or buffer.
    - <u>Thread producer</u>: gets segments from the content server using a TCP socket TC and the HTTP protocol and puts them in the queue.
    - <u>Thread consumer</u>: gets segments from the queue and writes its content to socket TP

The following picture summarizes the organization of the solution:

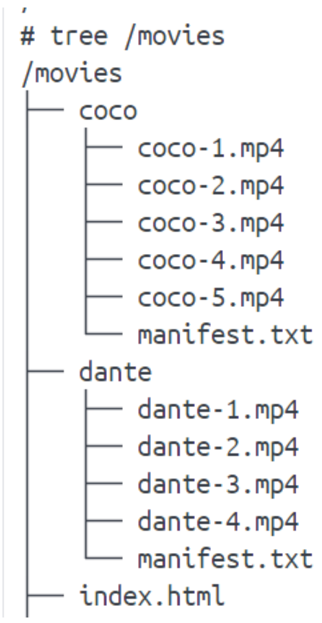


2. Content Server

The content server is a standard HTTP server like Apache or Internet Information Server. It stores the files with the videos in a folder called *movies*. It is implemented by a Docker container prepared some time ago by Prof. Sérgio Duarte. It can be run using the following command:

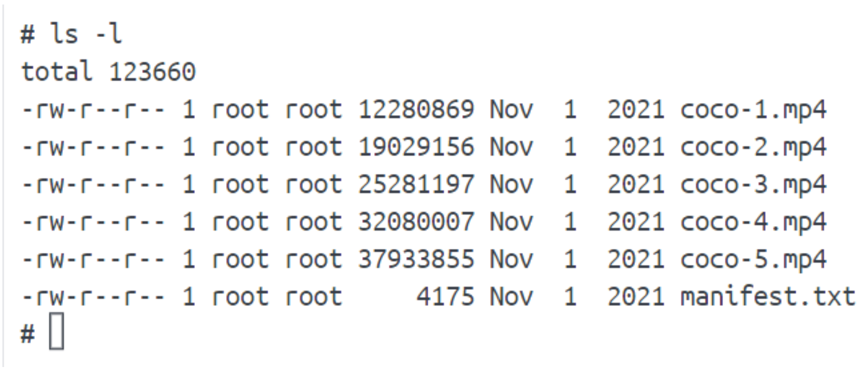
```
docker run -ti -p 9999:8080 smduarte/rc2021-tp1-test1
```

As defined in the start command above, the web server is accessible through the URL http://localhost:9999/ in the host machine.



```
# tree /movies
/movies
├── coco
│   ├── coco-1.mp4
│   ├── coco-2.mp4
│   ├── coco-3.mp4
│   ├── coco-4.mp4
│   ├── coco-5.mp4
│   └── manifest.txt
├── dante
│   ├── dante-1.mp4
│   ├── dante-2.mp4
│   ├── dante-3.mp4
│   ├── dante-4.mp4
│   └── manifest.txt
├── index.html
```

The server contains two small movies *coco* and *dante*. As seen in the output of the tree command above, there are 5 tracks for the movie *coco* and 4 tracks for movie *dante*. As exemplified for the coco movie, coco-1.mp4 corresponds to the lowest resolution and coco-5 has the better quality.



```
# ls -l
total 123660
-rw-r--r-- 1 root root 12280869 Nov  1  2021 coco-1.mp4
-rw-r--r-- 1 root root 19029156 Nov  1  2021 coco-2.mp4
-rw-r--r-- 1 root root 25281197 Nov  1  2021 coco-3.mp4
-rw-r--r-- 1 root root 32080007 Nov  1  2021 coco-4.mp4
-rw-r--r-- 1 root root 37933855 Nov  1  2021 coco-5.mp4
-rw-r--r-- 1 root root     4175 Nov  1  2021 manifest.txt
# 
```

The beginning of the *manifest.txt* file is shown below:

```
# more manifest.txt
coco
5
coco-1.mp4
video/mp4; codecs="avc1.42C015, mp4a.40.2"
593614
3003
50
0 1237
1237 270231
271468 257957
529425 145848
675273 147430
822703 155045
977748 207149
1184897 185679
1370576 304930
1675506 317544
1993050 297852
2290902 312885
2603787 296302
2900089 197729
3097818 260614
3358432 238584
```

The header of the file contains two lines:

1. Movie name
2. Number of tracks

Each track is described by the following lines:

1. File name for the mp4 file containing the track.
2. Codec used to play the fragment
3. Bit rate.
4. Duration.
5. Number of fragments. For all the tracks this number is the same (50).
6. *Number of segments* lines with 2 columns:
    a. File offset for the first byte of the segment.
    b. Number of bytes of the segment.

For our problem only lines 1, 5 and 6 to 55 matter.

3. Player

The player is a Python program that launches a movie viewer. The movie viewer suggested is *mplayer* that was developed for Linux but is also available in MacOS and Windows. The *mplayer* program should be installed previously. Please consult http://www.mplayerhq.hu/design7/dload.html for getting binary versions of the tool and installation instructions. Supposing that the viewer code is in file player.py it is invoked simply as *python player.py*. The code of *player.py* is as follows:

```
import socket
import subprocess

# Start a socket listening for connections on 0.0.0.0:8000
#(0.0.0.0 means all interfaces)

server_socket = socket.socket()
server_socket.bind(('0.0.0.0', 8000))
server_socket.listen(0)

# Accept a single connection and make a file-like object out of it
connection = server_socket.accept()[0].makefile('rb')
try:
    # Run a viewer with an appropriate command line. Uncomment the mplayer
    # version if you would prefer to use mplayer instead of VLC
    # For Mac:  alias vlc='/Applications/VLC.app/Contents/MacOS/VLC'
    # cmdline = '/Applications/VLC.app/Contents/MacOS/VLC --demux h264 -'
    cmdline = 'c:/Users/pedro/Bin/MPlayer/mplayer.exe -ni -'
    player = subprocess.Popen(cmdline.split(), stdin=subprocess.PIPE)
    while True:
        # Repeatedly read 1k of data from the connection and write it to
        # the media player's stdin
        data = connection.read(1024)
        if not data:
            break
        player.stdin.write(data)
finally:
    connection.close()
    server_socket.close()
    player.terminate()
```

The Python code launches the viewer with its standard input redirected to a pipe. Then, there is a cycle where the bytes read from the socket are copied to the pipe. The option **'-'** in the viewer indicates that it will display the bytes received in its standard input. It is possible to use other viewers as far as the redirection of input is supported.

Note: You must replace the highlighted line, with the absolute path to *mplayer* program on your computer.

4. Proxy

Finally, the work to be done! The proxy is invoked with the following command line:

```
python proxy.py baseURL movieName track
```

and will play the specified track of the movie indicated. With the server described in 2, the base URL is http://locahost:9999/. You must write the code for the proxy, corresponding to the main program and the two threads mentioned before:

The main program should

- Process the command line.
- Create the sockets necessary to the interaction with player and the content server.
- Create the queue for the cooperation between the two threads.
- Create and launch the producer and consumers threads.
- Wait for termination of the threads and do the cleanup.

Regarding thread management and the producer consumer pattern you can consult https://superfastpython.com/thread-producer-consumer-pattern-in-python/.

The producer thread should follow the steps below:

1. Obtain the *manifest.txt* file of the movie from the content server.
2. Process the string received in step 1 to obtain the number of segments of the track and the start and end offsets of each segment.
3. Obtain the contents of the next segment.
4. Put it in the queue.
5. If not the last segment go to step 3.
6. Return (corresponds to thread termination).

Note about step 1: In this step, you must send a GET HTTP command to the server. A compact way of doing this is to use the *requests* package of Python. See information about the *requests* package in *https://requests.readthedocs.io/en/latest/*.

Note about step 3: In this step, you must send a GET HTTP which specifies a byte range of the file corresponding to the current track. See the example in the lab session of last week and also https://developer.mozilla.org/en-US/docs/Web/HTTP/Range_requests .

You should use the requests package which allows an easy way to include the header line specifying the byte range. See the example below:

```
import requests

url = "http://stackoverflow.com"
headers = {"Range": "bytes=0-100"}          # first 100 bytes

r = requests.get(url, headers=headers)
```

The consumer thread should follow the steps below:

1. Obtain the next video segment from the queue.
2. Send the segment to the player through the socket.
3. If not the last segment go to step 1.
4. Return (corresponds to thread termination).

5. Delivery

The delivery should be done before 10:00 on November,28 2023. The submission has two parts:

- a Google form containing the identification of the students that submit the work and questions about the functionality of the code
- The code developed will be uploaded through Moodle

Details will be sent later.