

BASH based CGI applications.

Backend and web services testing - postman.

Team project development.

1. On line voting application - this week's lecture example

Activity:

Deploy into your VS the example AJAX application available on lecture 7 – on line voting, it's made of three files.

Thoroughly test the application's frontend.

1.1. Simulating a server unreachable scenario

The frontend was designed to be able to cope with network issues and recover from them, let's test that.

While the frontend is running and showing the current voting standings, let's make the server unreachable, for that purpose type the following command line in your server:

```
iptables -A INPUT -p tcp --dport 443 -j DROP
```

This will block incoming HTTPS traffic, to block HTTP traffic replace the 443 port number by 80 (you may type both command lines).

Check you frontend.

Make the server reachable again by removing the above rules:

```
iptables -F INPUT
```

Check you frontend again, it's expected to have fully recovered the normal operation.

1.2. Testing concurrent accesses to the web service

We already know from the lecture this application has a serious flaw due to the absence of concurrent access control, it may not have been noticed yet because tests so far have been rather modest.

Let's now create a "voting flood" in one candidate to see what happens, the idea is having several PUT requests at the same time and for that purpose we can run the **curl** command in background, the following bash script sends 10 PUT requests in parallel to cast 10 votes on candidate number 4:

```
#!/bin/bash
URL="http://vs129.dei.isep.ipp.pt/cgi-bin/votes"
CANDIDATE=4
for((i=0;i<10;i++))
do
    curl -X PUT ${URL}?c=${CANDIDATE} &
done
```

Create the script, for instance in file **votesFlood**.

Don't forget to change the server name in the URL to match your own.

Grant to the file the execute permission and run it, check the voting standings in the frontend ...

Fraud!! Not all votes are being counted, or even oddest things are happening.

For those with doubts, here is the evidence locking is a must in these cases of concurrent write access.

1.3. Implement a locking mechanism for voting

The issues are rising when several clients are simultaneously executing these three lines of our `/var/www/cgi-bin/votes` backend:

```
(...)  
N_CAND="${QUERY_STRING#c=}"  
STANDING_FILE=${STANDING_FILE_BASENAME}.$N_CAND  
VOTES=$(cat $STANDING_FILE)  
VOTES=$(( ${VOTES}+1 ))  
echo "$VOTES" > $STANDING_FILE  
exit  
(...)
```

While these three lines are being run by one instance of the script, no other instance may be running these same lines, so while running these lines, a lock must be acquired.

As discussed in the lecture, one way to implement a lock over a filesystem is by using the folder creating operation (**mkdir** command):

```
(...)  
N_CAND="${QUERY_STRING#c=}"  
STANDING_FILE=${STANDING_FILE_BASENAME}.$N_CAND  
while ! mkdir ${STANDING_FILE_BASENAME}.$N_CAND.lock ; do sleep 1; done  
VOTES=$(cat $STANDING_FILE)  
VOTES=$(( ${VOTES}+1 ))  
echo "$VOTES" > $STANDING_FILE  
rmdir ${STANDING_FILE_BASENAME}.$N_CAND.lock  
exit  
(...)
```

While the creation of the lock folder fails (because it already exists as it was created by another instance of the script) we wait one second and retry. Once we have acquired the lock, we may safely access and change the shared voting standing. Notice that there's a lock for each candidate, voting in different candidates is not required to be mutually exclusive because there's an independent file to store each candidate votes.

Add these two green lines you backend and test again the "votes flooding" with the previously created script (**votesFlood**), now all votes are expected to be counted, check your frontend.