Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

8ο εξάμηνο

2017-2018

Εργαστήριο Λειτουργικών Συστημάτων

1η Εργαστηριακή Άσκηση
Επιτήρηση χρήσης πόρων εφαρμογών με Linux Cgroups

Μουζάκης Ανάργυρος-Γεώργιος
Μουζάκης Νικόλαος

Εισαγωγή

Η άσκηση αφορά στην διαχείριση των πόρων του cpu subsystem με χρήση του συστήματος ***linux cgroups***. Συγκεκριμένα, μας δόθηκε ο δαίμονας ***cgmond***, σκοπός του οποίου είναι η διαχείριση των πόρων που προσφέρει ένα data center σε πελάτες. Προκειμένου να επιτευχθεί αυτή η λειτουργία, ο δαίμονας αλληλεπιδρά με τις ιεραρχίες ***cpu*** και ***monitor***. Συγκεκριμένα, χρησιμοποιώντας την ιεραρχία cpu μπορούν να γίνουν κατάλληλες ρυθμίσεις προκειμένου, για τις εφαρμογές που εκτελούνται, να εξασφαλιστεί ότι θα έχουν τουλάχιστον ένα μέρος της ισχύος του επεξεργαστή. Επιπλέον, μέσω της ιεραρχίας monitor ρυθμίζονται θέματα σχετικά με την εισαγωγή εφαρμογών σε cgroups, όταν αυτές ξεκινούν να εκτελούνται, καθώς και με τη διαγραφή τους, όταν ολοκληρώνουν την εκτέλεσή τους.
Ο δαίμονας δεν αλληλεπιδρά απευθείας με τις δύο παραπάνω ιεραρχίες, αλλά μέσω δύο python scripts τα οποία γράψαμε εμείς. Πρόκειται για τα ***mod_policy_cpumin.py*** και ***mod_limit_cpu.py***. Λεπτομερής περιγραφή των λειτουργιών που αυτά επιτελούν ακολουθεί στις επόμενες σελίδες.

mod_policy_cpumin.py

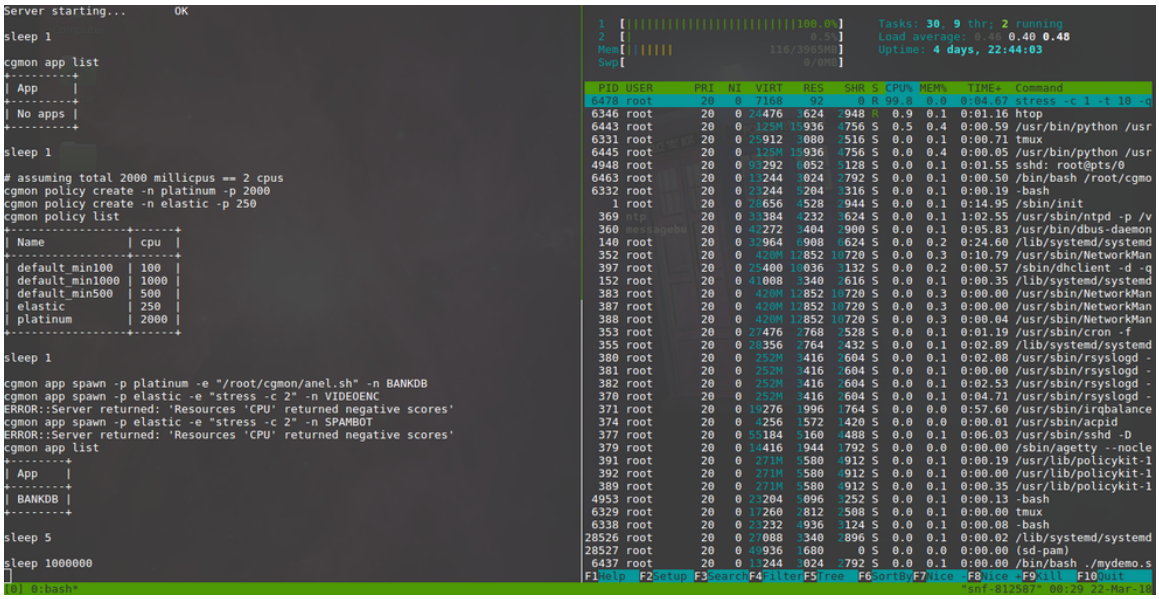
Ο κώδικας αυτού του script είναι ο ακόλουθος:

```
1.  # !/usr/bin/env python
2.  import sys
3.
4.  total=2000 #ta synolika xiliosta
5.
6.  lines=sys.stdin.readlines()
7.  apps=[]
8.  reqs=[]
9.  for line in lines:
10.         args=line.split(":")
11.         apps.append(args[1])
12.         reqs.append(int(args[3].rstrip()))
13.
14. if sum(reqs)>0:
15.         mult=total/(1.0*sum(reqs))
16. else:
17.         mult=1
18. if mult<1:
19.         score=-1
20. else:
21.         score=1
22.
23. sys.stdout.write("score:"+str(score)+"\n")
24.
25. reqs=[i*mult for i in reqs]
26.
27. for req,app in zip(reqs,apps):
28.         shares=int((req/(1.0*total))*10000)
29.         sys.stdout.write("set_limit:" + app + ":cpu.shares:" + str((shares))+"\n")
```

Το εν λόγω script αλληλεπιδρά με την ιεραρχία cpu και εκτελείται κάθε φορά που γίνεται spawn μία καινούρια εφαρμογή. Στόχος της λειτουργίας του είναι να ελέγχει αν είναι δυνατή η εισαγωγή της καινούριας εφαρμογής. Για αυτό το λόγο, υπολογίζει τα χιλιοστά που απαιτούν συνολικά οι εφαρμογές που ήδη εκτελούνται και προσθέτει σε αυτά εκείνα της καινούριας. Εφόσον το άθροισμα που θα προκύψει είναι μικρότερο από 2000 (το σύστημα όπου υλοποιούνται οι ασκήσεις έχει δύο επεξεργαστές, με τον καθένα να προσφέρει 1000 χιλιοστά), η καινούρια εφαρμογή μπαίνει και αυτή στο σύστημα για εκτέλεση. Τέλος, τα απαιτούμενα χιλιοστά των υπό εκτέλεση εφαρμογών μετατρέπονται σε cpu.shares. Ο κώδικας είναι δομημένος έτσι

ώστε η είσοδος και η έξοδος να συμφωνούν με τις προδιαγραφές που αναφέρονται στην εκφώνηση της άσκησης.

Εδώ πρέπει να αναφερθούν δύο πράγματα. Το πρώτο έχει να κάνει με τη συμπεριφορά εφαρμογών που ζητούν παραπάνω χιλιοστά από αυτά που προσφέρει ένας μεμονωμένος επεξεργαστής. Συγκεκριμένα, αν μία εφαρμογή απαιτεί 2000 χιλιοστά επεξεργαστικής ισχύος, θα δρομολογηθεί, εφόσον δεν εκτελείται κάποια άλλη. Όμως, εφόσον ο κώδικας της εφαρμογής δεν είναι παραλληλοποιήσιμος (πχ να γίνεται χρήση πολλαπλών threads κλπ), στην πράξη, η εφαρμογή θα αξιοποιεί μόνο 1000 χιλιοστά, αφού δεν θα μπορεί να εκτελείται παράλληλα σε δύο επεξεργαστές. Παρότι, όμως, δεν αξιοποιούνται πλήρως οι δυνατότητες του συστήματος, δεν θα μπορούν να εκτελεστούν άλλες εφαρμογές, αφού τα συνολικά απαιτούμενα χιλιοστά θα προκύπτουν περισσότερα από 2000. Μία τέτοια περίπτωση φαίνεται παρακάτω.

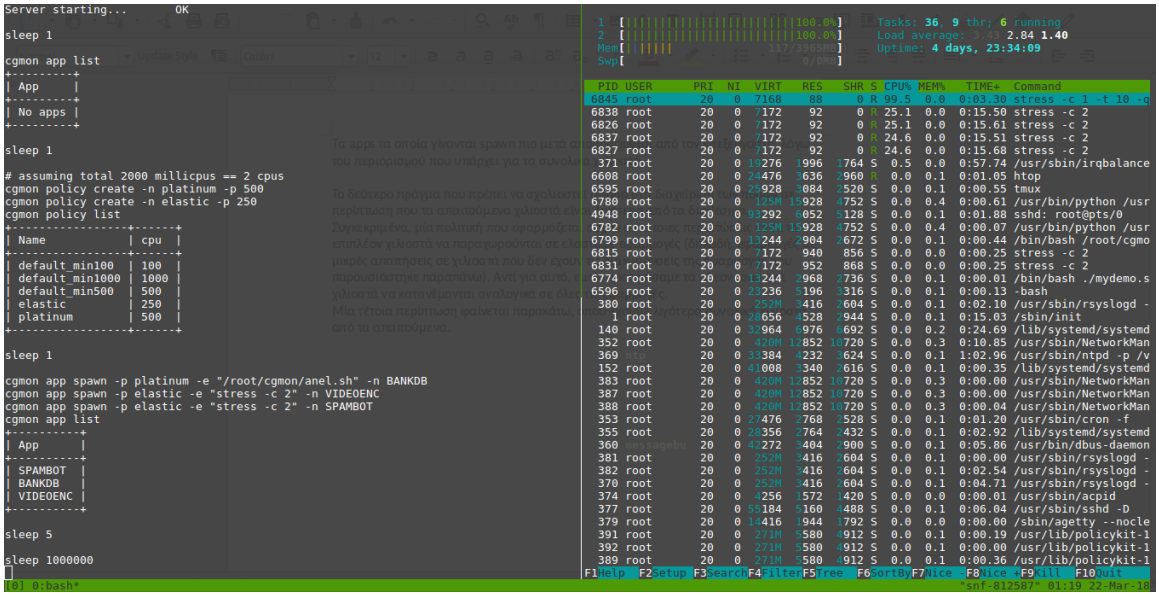


Συγκεκριμένα, πρώτα γίνεται spawn μία εφαρμογή με ελάχιστα απαιτούμενα χιλιοστά 2000. Ο κώδικας αυτής της εφαρμογής είναι:

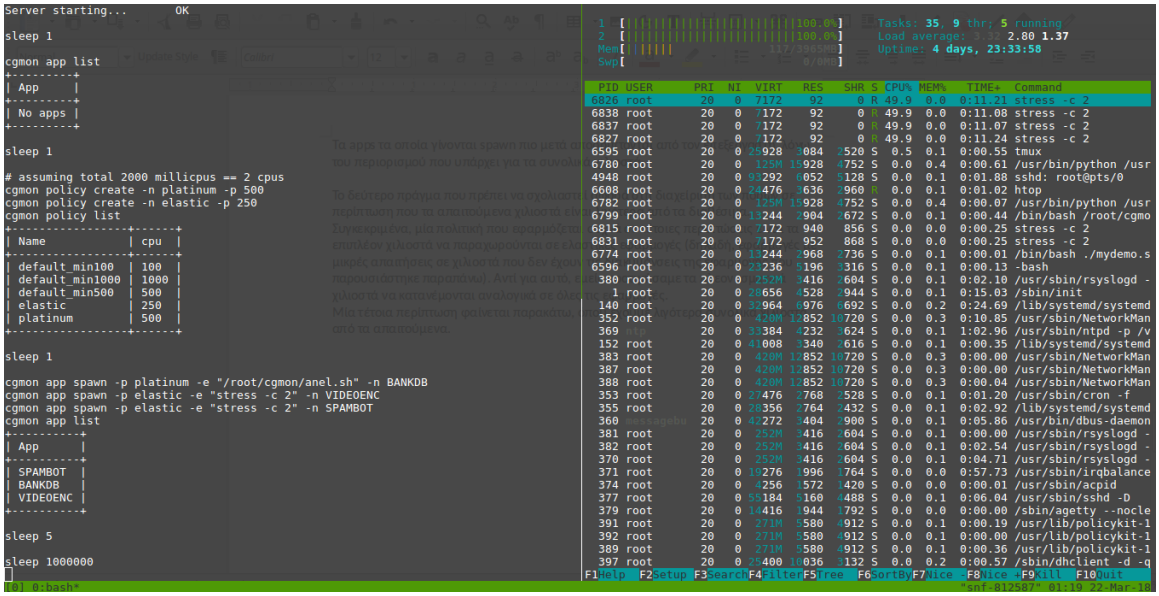
1. *# !/bin/bash*
2.
3. *while true*
4. *do*
5. *stress -c 1 -t 10 -q*
6. *sleep 10*
7. *done*

Ανά 10 δευτερόλεπτα, ο ένας εκ των δύο επεξεργαστών επιβαρύνεται από την εφαρμογή. Μετά ακολουθεί μία περίοδος 10 δευτερολέπτων, στη διάρκεια των οποίων η εφαρμογή παραμένει αδρανής. Αυτή η συμπεριφορά, σε συνδυασμό με τις απαιτήσεις σε χιλιοστά, προσδίδουν στην εν λόγω εφαρμογή όλα τα στοιχεία μίας ανελαστικής εφαρμογής.
Τα apps τα οποία γίνονται spawn πιο μετά απορρίπτονται από τον επεξεργαστή, λόγω του περιορισμού που υπάρχει για τα συνολικά χιλιοστά.

Το δεύτερο πράγμα που πρέπει να σχολιαστεί αφορά στη διαχείριση των πόρων σε περίπτωση που τα απαιτούμενα χιλιοστά είναι λιγότερα από τα διαθέσιμα. Συγκεκριμένα, μία πολιτική που εφαρμόζεται συχνά σε τέτοιες περιπτώσεις είναι τα επιπλέον χιλιοστά να παραχωρούνται σε ελαστικές εφαρμογές (δηλαδή, εφαρμογές με μικρές απαιτήσεις σε χιλιοστά που δεν έχουν τις διακυμάνσεις της εφαρμογής που παρουσιάστηκε παραπάνω). Αντί για αυτό, εμείς προτιμήσαμε τα πλεονασματικά χιλιοστά να κατανέμονται αναλογικά σε όλες τις εφαρμογές.
Δύο τέτοιες περιπτώσεις φαίνονται παρακάτω, όπου έχουμε λιγότερα συνολικά χιλιοστά από τα απαιτούμενα.



Σε αυτή την περίπτωση η ανελαστική εφαρμογή απαιτεί 500 χιλιοστά, ενώ υπάρχουν 2 ελαστικές, καθεμία εκ των οποίων απαιτεί 250. Το αποτέλεσμα είναι πως στην μεν πρώτη δίνεται το 100% ενός επεξεργαστή, οι δε άλλες μοιράζονται τον άλλο, με την καθεμία να παίρνει από 50% (στο htop βλέπουμε 4 των 25%, επειδή κάθε εφαρμογή κάνει spawn δύο workers που απασχολούν τη cpu).



Εδώ η ανελαστική εφαρμογή είναι σε περίοδο αδράνειας, οπότε οι 2 ελαστικές έχουν η καθεμία από 1 επεξεργαστή που απασχολούν κατά 100%.

mod_limit_cpu.py

Ο κώδικας αυτού του script είναι ο ακόλουθος:

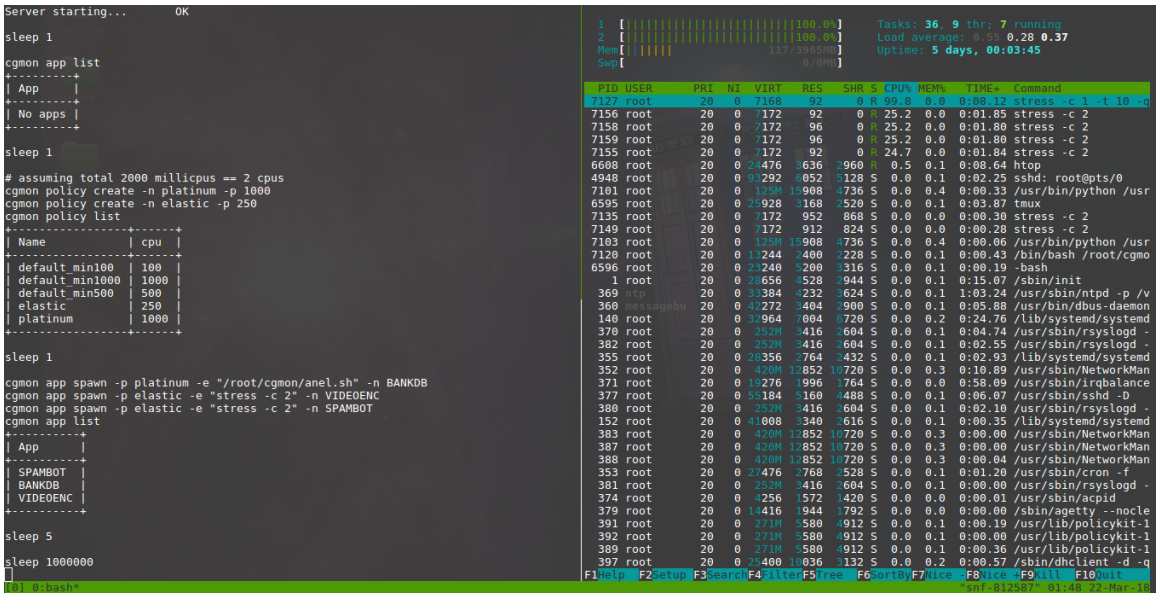
```
1.  # !/usr/bin/env python
2.  import sys
3.  import os
4.
5.  lines = sys.stdin.readlines()
6.
7.  file=open("/root/cgmon/LOG.txt","w")
8.
9.  for line in lines:
10.     args=line.split(":")
11.     if args[0]=='create':
12.         os.system('mkdir -p /sys/fs/cgroup/cpu/monitor/' +args[3].rstrip('\n'))
13.         file.write('mkdir -p /sys/fs/cgroup/cpu/monitor/' +args[3].rstrip('\n')+'\n')
14.     elif args[0]=='remove':
15.         os.system('rmdir /sys/fs/cgroup/cpu/monitor/' +args[3].rstrip('\n'))
16.         file.write('rmdir /sys/fs/cgroup/cpu/monitor/' +args[3].rstrip('\n')+'\n')
17.     elif args[0]=='add':
18.         os.system('echo '+args[4].rstrip('\n')+' >
    /sys/fs/cgroup/cpu/monitor/'+args[3]+'/tasks')
19.         file.write('echo '+args[4].rstrip('\n')+' >
    /sys/fs/cgroup/cpu/monitor/'+args[3]+'/tasks'+'\n')
20.     elif args[0]=='set_limit':
21.         os.system('echo '+args[5].rstrip('\n')+' >
    /sys/fs/cgroup/cpu/monitor/'+args[3]+'/cpu.shares')
22.         file.write('echo '+args[5].rstrip('\n')+' >
    /sys/fs/cgroup/cpu/monitor/'+args[3]+'/cpu.shares'+'\n')
23.     else:
24.         file.write('empty!\n')
25. file.close()
```

Αυτό το script αλληλεπιδρά με την ιεραρχία monitor, προκειμένου να εφαρμόσει τις ρυθμίσεις που παρήγαγε το προηγούμενο script. Παραπάνω φαίνονται οι εντολές που εκτελούνται σε καθεμία από τις πέντε περιπτώσεις που αναφέρονται στην εκφώνηση. Η πρώτη περίπτωση αντιστοιχεί στη δημιουργία ενός νέου cgroup. Η δεύτερη στη διαγραφή μίας εφαρμογής από ένα cgroup. Η τρίτη στην προσθήκη μίας εφαρμογής σε ένα υπάρχον cgroup. Η τέταρτη στην ρύθμιση των cpu.shares μιας εφαρμογής. Όταν γίνεται spawn μία εφαρμογή, εκτελούνται οι ενέργειες που αντιστοιχούν στην τρίτη και στην τέταρτη περίπτωση και, ενδεχομένως, στην πρώτη. Όταν ολοκληρώνεται μία εφαρμογή έχουμε την δεύτερη περίπτωση.

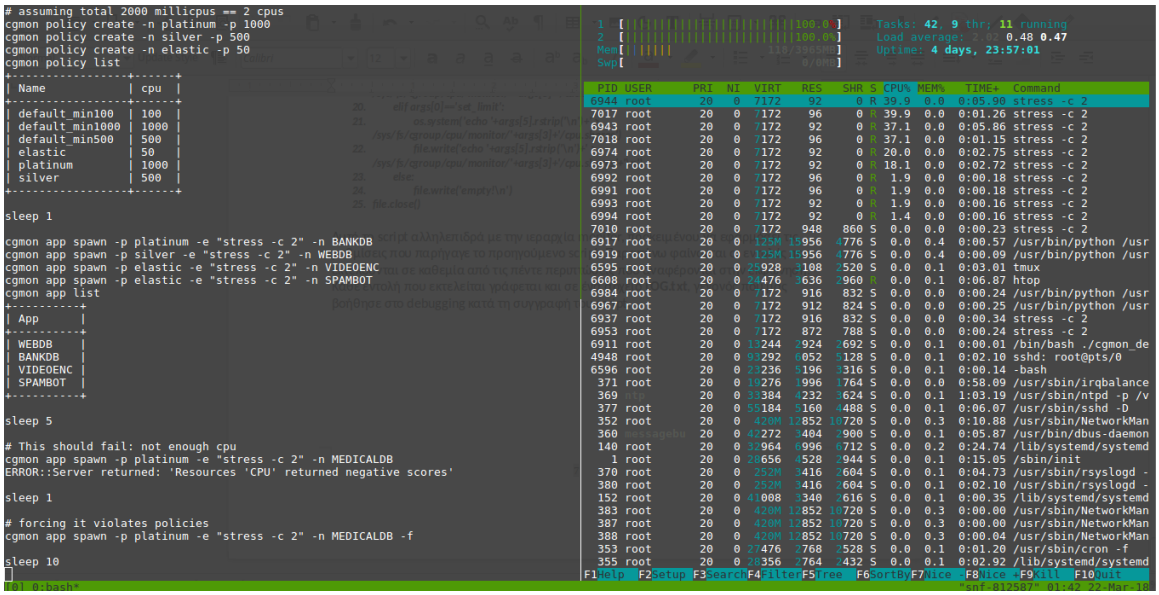
Κάθε εντολή που εκτελείται γράφεται και σε ένα αρχείο ***LOG.txt***, γεγονός που μας βοήθησε στο debugging κατά τη συγγραφή του script. Στο debugging βοήθησε και η ύπαρξη της εντολής ***else: file.write('empty!\n').***

Επιπλέον παραδείγματα

Παρακάτω φαίνονται τα αποτελέσματα μερικών ακόμη δοκιμών. Η πρώτη γίνεται με το δικό μας demo, που χρησιμοποιήθηκε και στα προηγούμενα, με τη διαφορά ότι τώρα δίνουμε 1000 χιλιοστά στην ανελαστική εφαρμογή. Το δεύτερο βασίζεται στο demo που δόθηκε ως βοηθητικό αρχείο.



Εδώ έχουμε άθροισμα χιλιοστών ίσο με το διαθέσιμο, οπότε όλες οι εφαρμογές παίρνουν ακριβώς τα χιλιοστά που ζητάνε.



Εδώ φαίνεται πώς, αρχικά, δεν επιτρέπεται σε μία εφαρμογή να εκτελεστεί, αλλά μετά, χρησιμοποιώντας το flag f, η πολιτική αυτή παραβιάζεται και, αναγκαστικά, αναπροσαρμόζονται οι τιμές των cpu.shares κάθε εφαρμογής.