cgroups

1) *Τι κάνουν:*

Ορίζουν την ***ελάχιστη*** προσβαση σε πόρο που παίρνει μια διεργασία ***αν τη ζητήσει.*** Πχ αν έχει share=100 και στην πραγματικότητα χρειάζεται μόνο 50, θα χρησιμοποιεί 50 και τα υπόλοιπα θα μοιραστούν στις άλλες διεργασίες. Αν ζητάει 200 τότε εξαρτάται από τι ζητούν οι υπόλοιπες.

2) *Πώς υπολογίζουμε shares:*

Τα hierarchies έχουν δομή filesystem. Σε κάθε φάκελο το αρχείο shares λέει τι ποσοστό από το συνολικό άθροισμα θα πάρει το συγκεκριμένο cgroup/φάκελος από τους πόρους του γονέα του.

3) *Πως χρησιμοποιούμε cgroups:*

Πρώτα δημιουργούμε το hierarchy:

mkdir /sys/fs/cgroup/mygroup

Μετά κάνουμε attach το subsystem(s):

mount –t cgroup –o cpu mygroup /sys/fs/cgroup/mygroup

Δημιουργούμε τυχόν cgroup μέσα στο hierarchy:

mkdir /sys/fs/cgroup/mygroup/c1

Αλλαγή παραμέτρων:

echo 512 > …/c1/cpu.shares

Προσθήκη διεργασίας σε cgroup :

echo 12445 > …/c1/tasks

Για να αλλάξουμε cgroup μιας διεργασίας απλά την προσθέτουμε στο καινούριο cgroup (πχ στον γονέα της) Η αφαίρεση από την προηγούμενη cgroup γίνεται αυτόματα. Για παράδειγμα (θεωρώντας πως είμαστε στο κατάλληλο directory):

echo 12445 > ../tasks

Σβήσιμο cgroup (γίνεται μόνο αν δεν έχει παιδιά και αν το tasks είναι άδειο)

rmdir .../c1

4) *Παραδείγματα*

Θέμα 1 (Επαναληπτική 2016)

α) i) *cgroup3 = 300 cgroup7 = 200*

ii) *P1=100/600*200/300*1/2=1/18*

P2=1/18

*P3=100/600*100/300*1/2=1/36*

P4=1/36

P5=200/600=1/3

*P6=300/600*200/300*1/2=1/6*

P7=1/6

*P8=300/600*100/300=1/6*

β) Δεδομένα: T,T1,T2,A1,M1

Ζητούμενο: A2

$$M1=\frac{T1*A1+T2*A2}{T} \rightarrow A2=\frac{T*M1-T1*A1}{T2}$$

2) Άρα πρέπει $0<\frac{T*M1-T1*A1}{T2}<1$

Αν δεν ισχύει αυτό τυπώνει προειδοποίηση και θέτει το A2 στην κοντινότερη τιμή.

1)Αφού αποφασιστεί το A2 έχουμε:

Χρόνος=0

mkdir A

echo pidA > A/tasks

mkdir B

echo pidB > B/tasks

echo 1000(A1) > A/cpu.shares*

echo 1000(1-A1) > B/cpu.shares*

Χρόνος=T1

echo 1000(A2) > A/cpu.shares*

echo 1000(1-A2) > B/cpu.shares*

γ) Σε python:

```
# !/usr/bin/env python
import sys
total=int(sys.stdin.readline().rstrip())
pids=[]
mins=[]
times=[]
lines=sys.stdin.readlines()
for line in lines:
        args=line.split(":")
        pids.append(args[0])
        mins.append(args[1])
        times.append(args[2].rstrip())

if sum(mins)>total:
        sys.stdout.write("ERROR not enough CPU")
        sys.exit()

sum_mins=sum(mins)
sum_times=sum(times)
n=len(lines)

if n==1:
        sys.stdout.write(str(pids[0])+":"+str(mins[0])+"/n")

else:
        for i in range(n):
                new_share=min[i]+1/(n-1)*(1-times[i]/sum_times)*(total-
sum_mins)
                sys.stdout.write(str(pids[i])+":"+str(new_share)+"/n")
```

Άσκηση 1

Το ***mod_policy_cpumin.py*** 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.

```
# !/usr/bin/env python
import sys

total=2000 #ta synolika xiliosta

lines=sys.stdin.readlines()
apps=[]
reqs=[]
for line in lines:
        args=line.split(":")
        apps.append(args[1])
        reqs.append(int(args[3].rstrip()))

if sum(reqs)>0:
        mult=total/(1.0*sum(reqs))
else:
        mult=1
if mult<1:
        score=-1
else:
        score=1
```

```
sys.stdout.write("score:"+str(score)+"\n")

reqs=[i*mult for i in reqs]

for req,app in zip(reqs,apps):
        shares=int((req/(1.0*total))*10000)
        sys.stdout.write("set_limit:" + app + ":cpu.shares:" + str((shares))
+"\n")
```

Το *mod_limit_cpu.py* 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.

```
# !/usr/bin/env python
import sys
import os

lines = sys.stdin.readlines()

for line in lines:
        args=line.split(":")
        if args[0]=='create':
                os.system('mkdir -p /sys/fs/cgroup/cpu/monitor/'
+args[3].rstrip('\n'))
        elif args[0]=='remove':
                os.system('rmdir /sys/fs/cgroup/cpu/monitor/'
+args[3].rstrip('\n'))
        elif args[0]=='add':
                os.system('echo '+args[4].rstrip('\n')+' >
/sys/fs/cgroup/cpu/monitor/'+args[3]+'/tasks')
        elif args[0]=='set_limit':
                os.system('echo '+args[5].rstrip('\n')+' >
/sys/fs/cgroup/cpu/monitor/'+args[3]+'/cpu.shares')
```