



Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών  
Υπολογιστών Ε.Μ.Π.

ΕΡΓΑΣΤΗΡΙΟ ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ  
2015-2016

ΑΣΚΗΣΗ 1η

Χρυσούλα Βαρηά  
ΑΜ: 03112105  
Ευαγγελία-Σοφία Γεργατσούλη  
ΑΜ: 03112064  
Εξάμηνο: 8ο

## Τα εκτελέσιμα policy και limit

- *policy*

Το εκτελέσιμο policy καλείται από τον cgroup κάθε φορά που δημιουργείται ή τερματίζεται κάποια εφαρμογή. Για όλες τις εφαρμογές, τις οποίες δέχεται το policy, υπολογίζεται το άθροισμα των συνολικών απαιτούμενων χιλιοστών της cpu. Αν το άθροισμα αυτό υπερβαίνει τις 2000 μονάδες, δηλαδή τις συνολικές διαθέσιμες μονάδες των 2 cpus, τότε το score είναι αρνητικό και ίσο με  $-\frac{sum}{2000}$  (όπου sum το άθροισμα των χιλιοστών cpu που απαιτούν όλες οι εφαρμογές), αλλιώς είναι θετικό. Αρνητικό score σημαίνει ότι η εφαρμογή, η οποία δημιουργήθηκε δε θα γίνει δεκτή, ενώ αντίθετα θετικό score δηλώνει ότι η εφαρμογή αυτή θα προστεθεί στο σύνολο των εφαρμογών, οι οποίες έχουν πρόσβαση στη cpu. Τα μεγέθη cpu.shares ταυτίζονται με την τιμή εισόδου value αντίστοιχα για κάθε εφαρμογή.

- *limit*

Αυτό το εκτελέσιμο αναλαμβάνει να διαχειριστεί τα cgroups και τα χιλιοστά που δίνουμε σε κάθε διεργασία ανάλογα με τις οδηγίες που θα λάβει από το εκτελέσιμο policy. Πιο συγκεκριμένα, δημιουργεί και διαγράφει φακέλους στην ιεραρχία των cgroups (/sys/fs/cgroups/) για τη δημιουργία νέου, ή την κατάργηση κάποιου cgroup όταν πεθάνει η διεργασία που άνηκε σε αυτό, ή μεταφερθεί σε κάποιο άλλο group (στην συγκεκριμένη άσκηση, επειδή έχουμε μόνο 1 εφαρμογή ανα cgroup, αυτό θα διαγραφεί μόνο όταν ο cgroup διαπιστώσει ότι όλες οι διεργασίες της εφαρμογής έχουν πεθάνει). Αλλάζει επίσης το αρχείο cpu.shares, καταγράφοντας τα χιλιοστά που θέλουμε να έχει η εφαρμογή που ανήκει στο συγκεκριμένο cgroup και το αρχείο tasks, για την προσθήκη (ή την μετακίνηση) μιας διεργασίας από ένα cgroup σε ένα άλλο.

## Παράδειγμα εκτέλεσης

Δημιουργήθηκαν 3 ελαστικές (stress) με απαίτηση για 50 μονάδες η κάθε μια (2.5%), και 2 ανελαστικές\* (notElast.py και notEl) με απαίτηση για 800 (40%) και 1000 (50%) μονάδες αντίστοιχα. Στις εικόνες 1 και 2 φαίνεται η χρήση της cpu στην περίπτωση που καμία δεν κοιμάται, και σε περίπτωση που κοιμάται η notEl. Παρατηρούμε ότι σε κάθε περίπτωση τηρούνται τα ζητούμενα όρια, ενώ λόγω της ύπαρξης ελαστικών εφαρμογών, η cpu δεν θα μένει ποτέ αδρανής.

\*\* Οι ανελαστικές είναι ένα εκτελέσιμο C και ένα εκτελέσιμο python, τα οποία κάποιο μέρος του χρόνου εκτελούν κάποιες πράξεις (τετραγωνική ρίζα και ύψωση σε δύναμη) και το υπόλοιπο κοιμούνται.

1	[   100.0%]	Tasks: 44, 9 thr; 9 running
2	[   100.0%]	Load average: 6.32 2.81 1.85
Mem	[                 ]	Uptime: 1 day, 02:41:20
Swp	[     ]	
		440/2010MB
		0/0MB

  

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
6032	root	20	0	4076	680	608	R	97.4	0.0	0:58.52	/root/working/notEl
6048	root	20	0	336M	320M	4100	R	86.5	15.9	1:41.23	python /root/working/notElast.py
6020	root	20	0	7172	92	0	R	2.9	0.0	0:09.92	stress -c 2
6021	root	20	0	7172	92	0	R	2.9	0.0	0:09.93	stress -c 2
6003	root	20	0	7172	92	0	R	2.9	0.0	0:09.98	stress -c 2
6004	root	20	0	7172	92	0	R	2.9	0.0	0:10.06	stress -c 2
6056	root	20	0	7172	88	0	R	2.9	0.0	0:09.83	stress -c 2
6055	root	20	0	7172	88	0	R	2.9	0.0	0:09.83	stress -c 2
6057	root	20	0	24624	3680	2860	R	0.0	0.2	0:00.24	htop

Εικόνα 1

1	[   100.0%]	Tasks: 44, 9 thr; 8 running
2	[   100.0%]	Load average: 6.10 2.65 1.79
Mem	[                 ]	Uptime: 1 day, 02:41:09
Swp	[     ]	
		440/2010MB
		0/0MB

  

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
6048	root	20	0	336M	320M	4100	R	99.8	15.9	1:31.46	python /root/working/notElast.py
6021	root	20	0	7172	92	0	R	17.1	0.0	0:08.98	stress -c 2
6003	root	20	0	7172	92	0	R	17.1	0.0	0:09.01	stress -c 2
6055	root	20	0	7172	88	0	R	16.6	0.0	0:08.88	stress -c 2
6004	root	20	0	7172	92	0	R	16.6	0.0	0:09.09	stress -c 2
6056	root	20	0	7172	88	0	R	16.6	0.0	0:08.87	stress -c 2
6020	root	20	0	7172	92	0	R	16.6	0.0	0:08.96	stress -c 2
6057	root	20	0	24624	3680	2860	R	0.5	0.2	0:00.22	htop
5539	evagerg	20	0	93256	3612	2692	S	0.5	0.2	0:00.21	sshd: evagerg@pts/0
6032	root	20	0	4076	680	608	S	0.0	0.0	0:53.03	/root/working/notEl

Εικόνα 2

## Χειρισμός ελαστικών-ανελαστικών

Στον κώδικα μας δεν υπήρξε κάποιος διαφορετικός χειρισμός στις ελαστικές και τις ανελαστικές εφαρμογές με αποτέλεσμα όταν δεν δίνονταν καθόλου ελαστικές, υπήρχε περίπτωση η cru να μένει αδρανής (αν όλες μαζί τύχαινε να μην χρειάζονται την cru). Μια ιδέα/πιθανή πολιτική αντιμετώπισης του παραπάνω προβλήματος είναι η εξής:

- Κάνουμε την παραδοχή ότι ελαστικές διεργασίες είναι όσες απαιτούν 50 ή λιγότερα ελάχιστα χιλιοστά της cru.
- Στην περίπτωση δημιουργίας μιας εφαρμογής, το εκτελέσιμο policy θα ελέγχει το άθροισμα των ελάχιστων απαιτούμενων χιλιοστών των ανελαστικών εφαρμογών.
  - Αν η νέα εφαρμογή είναι ελαστική μπορούμε να την προσθέσουμε στο σύνολο των ήδη υπαρχουσών εφαρμογών με την προϋπόθεση ότι το συνολικό άθροισμα των απαιτούμενων χιλιοστών δεν ξεπερνά κάποιο άνω όριο (για παράδειγμα τις 3000 μονάδες, έτσι ώστε να αποφεύγονται περιπτώσεις λιμοκτονίας ή μεγάλης καθυστέρησης τερματισμού/έξυπνής των εφαρμογών).
  - Αν η νέα εφαρμογή είναι ανελαστική, θα μπαίνει μόνο σε περίπτωση που το άθροισμα των μοναδων της μαζί με τις προυπάρχουσες ανελαστικές δεν ξεπερνάει το 2000
- Στην περίπτωση διαγραφής κάποιας εφαρμογής, τροποποιούμε τις τιμές των cru.shares έτσι ώστε να εξασφαλίζονται τα ελάχιστα χιλιοστά για τις ανελαστικές και στις ελαστικές να προστίθενται τα επιπλέον χιλιοστά, τα οποία ελευθερώθηκαν από την εφαρμογή, αν το συνολικό απαιτούμενο άθροισμα είναι μικρότερο του 2000. Με αυτόν τον τρόπο τα ελεύθερα χιλιοστά της cru θα κατανέμονται στις ελαστικές και όχι στις ανελαστικές εφαρμογές.

Τέλος, προκειμένου να μπορούμε να κατανέμουμε τα ελεύθερα χιλιοστά μόνο στις ελαστικές εφαρμογές, θα πρέπει να γνωρίζουμε πότε κάποια εφαρμογή χρησιμοποιεί ή όχι τη cru (για παράδειγμα αν κάποια εφαρμογή είναι σε sleep τα χιλιοστά της θέλουμε να αξιοποιηθούν μόνο από τις ελαστικές και όχι από όλες τις εφαρμογές). Για να γίνει αυτό θα πρέπει με κάποιο τρόπο να παρατηρούμε πότε μια εφαρμογή χρησιμοποιεί τη cru ή όχι και συγκεκριμένα:

- Όταν παρατηρηθεί ότι κάποια ανελαστική καταναλώνει λιγότερο από το minimum που έχει ζητήσει, θα μοιράζουμε το ελεύθερο κομμάτι της cru μόνο στις ελαστικές (αλλάζοντας τα cru.shares τους).
- Όταν η ανελαστική διεργασία ξυπνήσει, θα επαναφέρουμε τα cru.shares των ελαστικών όπως ήταν αρχικά ώστε οι ανελαστικές να παίρνουν ακριβώς όσο ζήτησαν και το άθροισμα να μην ξεπερνάει το 2000 (οπότε θα έπαιρναν κάποιο ποσοστό αυτού που ζήτησαν).

# Κώδικας

*policy.c:*

```
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char*argv[]){
    FILE *f;
    char line[1024], pol[10], temp_name[40], temp_val[10];
    char **names=NULL;
    char **values=NULL;
    char * p;
    int n_spaces=0,v_spaces=0,i;
    int sum_cpu=0;
    float score;
    char c;
    f = stdin;
    while(1){
        if(fscanf(f, "policy:") <0)
            break;

        i=0;
        c=fgetc(f);
        while(c!=':') {
            temp_name[i]=c;
            i++;
            c=fgetc(f);
        }
        temp_name[i]='\0';

        names =(char**)realloc (names, sizeof(char*) * (++
            n_spaces));

        names[n_spaces-1]=malloc(sizeof(char)*(strlen(
            temp_name)+1));
        strncpy(names[n_spaces-1],temp_name,strlen(temp_name)
            );

        fgetc(f);
        fgetc(f);
        fgetc(f);
        fgetc(f);
        i=0;
        c=fgetc(f);
        while(c!='\n' && c!=EOF){
            temp_val[i]=c;
            i++;
```

```

        c=fgetc (f);
    }
    temp_val[i] = '\0';
    values = (char**)realloc (values, sizeof(char)* (++
        v_spaces));

    values[v_spaces-1]=malloc (sizeof(char)*(strlen(
        temp_val)+1));
    strncpy (values[v_spaces-1],temp_val, strlen (temp_val))
        ;
    sum_cpu=sum_cpu+(int) strtol (values[v_spaces-1],(char
        **)NULL,10);
}

score=sum_cpu /2000.0;
if (score >1)
    score=-score;
printf (" score: %.2f\n", score);
for(i=0; i<(n_spaces); i++)
    printf (" set_limit:%s:cpu.shares:%d\n", names[i], (int)
        )strtol(values[i], (char **)NULL, 10));

free (values);
free (names);

return 0;
}

```

***limit.c:***

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define PATH "/sys/fs/cgroup/cpu/"
char temp[100];
int next_line () {
    int i=0;
    char c;
    c=getchar ();
    for (; c!=EOF && c!='\n'; c=getchar (), i++)
        temp[i]=c;
    temp[i] = '\0';
    return i;
}
int next_word(const char temp[], char arr[], int index)
{
    int i=index, newi=0;
    for (; temp[i] != ':' && temp[i] != '\0'; i++, newi++)
        arr[newi]=temp[i];
}

```

```

        arr[newi] = '\0';
        return newi; // length of word
    }
int main ()
{
    int i, len, act_len, mon_len, app_len, count;
    char action[10], monitor[20], app_name[40], pid[5], value[5],
        command[100], total_path[200];

    while (next_line() != 0) {
        command[0] = '\0';
        total_path[0] = '\0';
        /* Parse word */
        len = strlen(temp);
        count = 0;
        count += next_word(temp, action, 0);

        count += next_word(temp, monitor, count + 1);
        count += 6; // add ":cpu:"

        count += next_word(temp, app_name, count);
        strcat(total_path, PATH);
        strcat(total_path, monitor);
        strcat(total_path, "/");
        strcat(total_path, app_name);
        strcat(total_path, "/");
        if (!strcmp(action, "add")) { // more arguments
            count += next_word(temp, pid, count + 1); /* add
                an process into existing cgroup */
            strcat(command, "echo ");
            strcat(command, pid);
            strcat(command, " >> ");
            strcat(command, total_path);
            strcat(command, "tasks");
            system(command);
        }
        else if (!strcmp(action, "set_limit")) { /* Change cpu.
            shares for app_name */
            count += next_word(temp, value, count + 12); //:
                cpu.shares:
            strcat(command, "echo ");
            strcat(command, value);
            strcat(command, "> ");
            strcat(command, total_path);
            strcat(command, "cpu.shares");
            system(command);
        }
    }
}

```

```

        else if(!strcmp(action, "create")){ /* create
            directory-cgroup */
                strcat(command, "mkdir -p ");
                strcat(command, total_path);
                system(command);
        }
        else{ /*Remove directory-cgroup */
                strcat(command, "rmdir "); /// sys/fs/cgroup/");
                strcat(command, total_path);
                system(command);
        }
    }
    return 0;
}

```

***notElast.py:***

```

import math
import time

while True:
    for x in range(0, 10000000):
        for i in range(0, 100000):
            for y in range(0,1000):
                z=math.sqrt(x)

    time.sleep(10)

```