Σιγλίδης Γιάννης (03112069)
Κωνσταντινίδης Ορέστης (03113670)
Ομάδα Α22

Εργαστήριο Λειτουργικών Συστημάτων :

Επιτήρηση χρήσης πόρων εφαρμογών
με Linux Cgroups

1η Άσκηση για το μάθημα Εργαστήριο Λειτουργικών Συστημάτων.
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Διδάσκοντες : Ν. Κοζύρης, Π. Τσανάκας

Ζητούμενα της Άσκησης

1. Όπως φαίνεται στον κώδικα που παραθέτουμε παρακάτω, τα προγράμματα που γράψαμε (σε γλώσσα C), συμφωνούν με τις προδιαγραφές που δίδονται στις αντίστοιχες ενότητες ώστε να μπορούν να χρησιμοποιηθούν από τον cgmond. Εκτός από την παρουσίαση της άσκησης μας στο εργαστήριο όπου εξεταστήκαμε και επιδείξαμε την λειτουργικότητά τους, μπορεί κάποιος εύκολα να δει ότι τα προγράμματα αυτά διαβάζουν προβλεπόμενα την είσοδο και παράγουν έξοδο με τις συγκεκριμένες συντακτικές ιδιότητες που απαιτούνται από την εκφώνηση της άσκησης.

   Πιο συγκεκριμένα, το πρόγραμμα cgmon-policy, στην γραμμή 23, διαβάζει από τον πίνακα A, όπου αποθηκεύεται προσωρινά η είσοδος, το πρότυπο που υποδεικνύει η εκφώνηση, θεωρώντας ότι δεν μπορεί να είναι λανθασμένο μιας και είναι έξοδος του δέμονα.

   Στην περίπτωση μη αποδεκτής αίτησης cpu-χρόνου, το παραπάνω πρόγραμμα στην γραμμή 32, παράγει ως έξοδο την τιμή $score$ με την προβλεπόμενη μορφή εξόδου και όπως φαίνεται εύκολα από την συνθήκη της δομής $if(<$συνθήκη$>)$, αρνητική τιμή, η οποία υποδεικνύει στον δαίμονα ότι η εξεταζόμενη αίτηση πρέπει να απορριφθεί, λόγω περιορισμού πηγών (resources).

   Αν η αίτηση γίνει δεκτή (υπάρχουν διαθέσιμες πηγές (resources)), τότε στην γραμμή 46 τυπώνεται η τιμή του $score$, που προφανώς είναι θετική, γιατί αλλιώς θα πέφταμε στην περίπτωση που περιγράφηκε παραπάνω. Τέλος, ξαναδιαβάζουμε την είσοδο (βρίσκεται στον πίνακα A) και δίνουμε την κατάλληλη (ζητούμενη) έξοδο, κανονικοποιημένη ως προς τις πραγματικές πηγές (resources).

   Το πρόγραμμα cgmon-limit, ελέγχει την ενέργεια που ζητάει ο δαίμονας, ταιριάζοντας το κατάλληλο πρότυπο. Τυπώνει την κατάλληλη εντολή στο string με όνομα command και στην συνέχεια μέσω της εντολής system την εκτελεί μέσω εντολών shell του λειτουργικού.

2. Αυτό εξασφαλίζεται σε δύο σημεία: Πρώτον, από το πρόγραμμα cgmon-policy γραμμή 51, όπου υπολογίζονται και δίνονται ως έξοδο τα κανονικοποιημένα cpu.shares. Οπότε εξασφαλίζεται η ενημέρωση του δαίμονα, και κατ' επέκταση του προγράμματος cgmon-limit για τις ανάγκες της κάθε εφαρμογής. Δεύτερον, και ακόμα πιο σημαντικό, στο ίδιο πρόγραμμα όπως αναφέρθηκε στο προηγούμενο ερώτημα, ελέγχεται η δυνατότητα ικανοποίησης της κάθε αίτησης και άρα εξασφαλίζεται η ύπαρξη πηγών (resources). Τρίτον και τελευταίο, στην γραμμή 47 του προγράμματος cgmon-limit, εξασφαλίζεται ότι η τιμή cpu.shares τυπώνεται στο cgroup της εφαρμογής στο κατάλληλο path ώστε να υλοποιηθεί η δέσμευση του cpu-χρόνου που αιτείται.

3. Αυτό απαντήθηκε ήδη στο πρώτο ερώτημα, όπου δόθηκαν και οι απαραίτητες κατευθύνσεις, ώστε να ελέγξει κάποιος εύκολα ότι το πρόγραμμά cgmon-policy ελέγχει την ικανότητα ικανοποίησης της αίτησης από το σύστημα και σε περίπτωση αρνητικής απάντησης, και μόνον τότε, επιστρέφεται αρνητική τιμή του $score$.

4. Αυτό το σημείο δεν υλοποιήθηκε αλλά θεωρούμε ότι μπορούμε να δώσουμε μια καλή περιγραφή του πως θα αντιμετωπίζαμε την περίπτωση της κοινής συνύπαρξης ελαστικών και ανελαστικών εφαρμογών. Ο τρόπος που θα υλοποιούσαμε μια κοινή συνύπαρξη τέτοιων εφαρμογών, θα ήταν ό εξής: κρατούμε δύο μεταβλητές sum_1 και sum_2 για ελαστικές και ανελαστικές εφαρμογές και έπειτα ελέγχουμε ότι $sum_1 + sum_2 \leq given\_cpu\_shares$ και αν όχι επιστρέφουμε αρνητικό score και τερματίζουμε το πρόγραμμα. Αλλιώς αφού έχουμε ολοκληρώσει όλη την ανάγνωση της εισόδου ($stdin$), και την καταγραφή της σε ένα αρχείο μαζί με το είδος κάθε εφαρμογής, κλείνουμε το αρχείο, τυπώνουμε το θετικό πλέον score, και έπειτα ξανανοίγοντας το, τώρα για ανάγνωση, ακολουθούμε την παρακάτω διαδικασία.

   Αν μία εφαρμογή είναι ανελαστική τυπώνουμε για αυτήν την τιμή: $\frac{cpu}{given\_cpu\_shares} \cdot 1024$.

   Διαφορετικά, για μία ελαστική εφαρμογή, τυπώνουμε: $\frac{cpu}{sum_1} \cdot \frac{(given\_cpu\_shares - sum_2)}{given\_cpu\_shares} \cdot 1024$.

> Δήλαδη στις ελαστικές εφαρμόγες δίνουμε τον εναπομείναντα χώρο, κανονικοποιημένο στο 1024, ολοκληρωτικά και κατ'αναλογία. Έτσι εξασφαλίζεται ότι, εάν υπάρχουν ελαστικές και ανελαστικές εφαρμογές υπό επιτήρηση, όλοι οι επεξεργαστές του κόμβου χρησιμοποιούνται, και κανένας δεν θα μένει αδρανής. Η ορθότητα του προγράμματος εξασφαλίζεται από το γεγονός ότι στις ελαστικές εφαρμογές, δίνουμε το μέγιστο (αναλογικά) δυνατό, ενώ ταυτόχρονα παρέχουμε στις ανελαστικές ακριβώς το ελάχιστο εγγυημένο.

Παρακάτω παρατίθεται ο πηγαίος κώδικας των δύο προγραμμάτων που υλοποιούν τις λειτουργίες που ζητούνται. Προστεθήκαν αρκετά σχόλια, ώστε να μπορεί να γίνει εύκολα κατανοητός ο σκοπός του κάθε βήματος.

cgmon–policy.c

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

int main() {
        FILE *fp;

        char A[1024];
        char path[1024];
        char app_name[256];
        int cpu;
        int sum=0;

        /* You may have to initialise sum to existing sum */
        int given_cpu_shares=2000;

        /* Creating temporary adress for storing information on a file --[unique for each pid] */
        sprintf(path,"/etc/rex%d",(int)getpid());

        fp=fopen(path,"wr");
        while (scanf("%s\n",A) != -1) {

                sscanf(A,"policy:%[^:]:cpu:%d",app_name,&cpu);
                sum=cpu+sum;
                fprintf(fp,"%s\n",A);

                /* Cpu shares are acceptable? */
                if(sum>given_cpu_shares){

                        /* Pop error message */
                        fclose(fp);
                        printf("score:%f\n",(given_cpu_shares - sum)*1.0);

                        /* Delete file and exit */
                        remove(path);
                        return 0;
                }

        }
        fclose(fp);

        /* Cpu shares are acceptable, reopen the file from the top, readonly */
        fp=fopen(path,"r");

        /* Output score value */
        printf("score:%f\n",(given_cpu_shares - sum)*1.0);

        while (fscanf(fp,"%s\n",A) != -1) {

                sscanf(A,"policy:%[^:]:cpu:%d\n",app_name,&cpu);

                /* Normalisation of the cpu.shares value and output*/
                printf("set_limit:%s:cpu.shares:%d\n",app_name,(cpu*1024)/sum);

        }
        fclose(fp);

        /* Delete file and exit */
        remove(path);
        return 0;

}
```

cgmon–limit.c

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

int main() {

        char monitor[256],app_name[256];
        int pid,csv;
        char A[1024];
        char command[1024];

        while (fscanf(stdin,"%s\n",A)!=-1) {

                /* Create Cgroup for some new app */
                if(sscanf(A,"create:%[^:]:cpu:%s",monitor,app_name) != 0){

                        sprintf(command,"mkdir -p /sys/fs/cgroup/cpu/%s/%s",monitor,app_name);

                        system(command);
                        continue;

                }

                /* Remove Cgroup for some finished app */
                if(sscanf(A,"remove:%[^:]:cpu:%s",monitor,app_name) != 0){

                        sprintf(command,"rmdir /sys/fs/cgroup/cpu/%s/%s",monitor,app_name);

                        system(command);
                        continue;

                }

                /* Add process to an existing Cgroup */
                if(sscanf(A,"add:%[^:]:cpu:%[^:]:%d",monitor,app_name,&pid) != 0){

                        sprintf(command,"echo %d >> /sys/fs/cgroup/cpu/%s/%s/tasks",pid,monitor,
    app_name);

                        system(command);
                        continue;

                }

                /* Set cpu.shares value for the Cgroup of some app */
                if(sscanf(A,"set_limit:%[^:]:cpu:%[^:]:cpu.shares:%d",monitor,app_name,&csv) != 0){

                        sprintf(command,"echo %d > /sys/fs/cgroup/cpu/%s/%s/cpu.shares",csv,monitor
    ,app_name);

                        system(command);
                        continue;

                }

        }

        return 0;

}
```