Φροντιστήριο: 27/11/2015
Παράδοση: 11/12/2015

Implementation of a periodic scheduler for the Linux Operating system.

Σκοπός της άσκησης είναι η υλοποίηση ενός αλγορίθμου χρονοπρογραμματισμού(scheduler) περιοδικών(periodic) διεργασιών. Ο scheduler του kernel 2.6.38.1 είναι ένας Completely Fair Scheduler (CFS). Τυπικά, οι διεργασίες χωρίζονται σε 140 προτεραιότητες(priority). Priority 0-99 είναι οι real-time διεργασίες. Priority 100-139 είναι οι κανονικές διεργασίες. Αυτές οι προτεραιότητες είναι δυναμικές και μια διεργασία μπορεί να αλλάζει προτεραιότητα ανάλογα με τις ανάγκες της. Όλες αυτές οι διεργασίες βρίσκονται σε ένα running queue και κάθε φορά που γίνεται ένα context switch ο scheduler διαβάζει αυτό το queue για να αποφασίσει ποια θα είναι η επόμενη διεργασία που θα τρέξει.

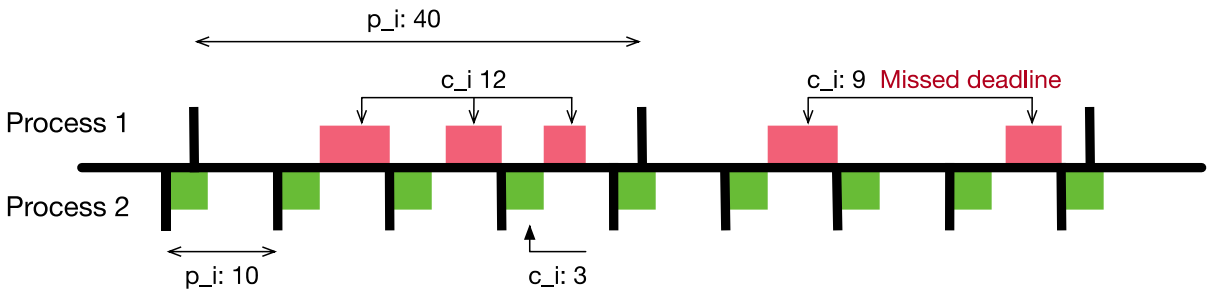
Θεωρία:

Όπως έχουμε ορίσει τις περιοδικές διεργασίες έχουν μία περίοδο p_i και ένα χρόνο εκτέλεσης c_i σε **milliseconds**. Η διεργασία μπορεί να εκτελεστεί στην αρχή της περιόδου i, στο τέλος της περιόδου (t = p_i – c_i) ή οποιαδήποτε χρονική στιγμή ανάμεσα στην αρχή και στο τέλος της περιόδου. Επίσης μπορεί να εκτελεστεί και τμηματικά μέσα στη περίοδό της. Η διεργασία πρέπει να λάβει χρόνο c_i σε κάθε περίοδο από την στιγμή που η περίοδός της ξεκινάει. Αν μία διεργασία δεν καταφέρει να λάβει c_i CPU χρόνου τότε λέμε ότι έχει χάσει ένα deadline. Ο τρόπος που ο scheduler σας θα αποφασίζει ποια από τις periodic διεργασίες θα τρέξει είναι ο **shortest period first**. Σκοπός είναι να διαλέγει αυτή που έχει την μικρότερη περίοδο(p_i) **και** δεν έχει εξαντλήσει το χρόνο της(c_i) ακόμα για το συγκεκριμένο p_i. Όταν έχουμε δύο διεργασίες, μια periodic και **ίδιας προτεραιότητας** ο scheduler θα πρέπει να διαλέγει την periodic διεργασία έναντι της κανονικής.

Παράδειγμα:
Έχουμε δύο διεργασίες, Process 1 & 2 με τα εξής χαρακτηριστικά:
Process 1: p_i: 40 seconds
c_i: 12 seconds

Process 2: p_i: 10 seconds
c_i: 3 seconds

Οι μαύρες κάθετες γραμμές ορίζουν την αρχή-τέλος του κάθε p_i του κάθε process αντίστοιχα. Η εκτέλεση του process 1 είναι με ροζ χρωμα και του process 2 με πράσινο χρώμα. Παρατηρούμε ότι το process 1 δεν παίρνει το χρόνο του συνεχόμενα αλλά διακόπτεται από το process 2 λόγω του policy shorted period first. Στην $1^{η}$ περίοδο του το process 1 καταφέρνει να πάρει τα 12 seconds c_i time αλλά την $2^{η}$ παίρνει μόνο 9 seconds. Αυτό σημαίνει ότι έχασε το deadline της εκτέλεσης του.



Υλοποιήση:

Στόχος της άσκησης είναι να βρείτε την δικιά σας λύση για την υλοποίηση αυτού του scheduler. Ο τρόπος που θα το πετύχετε βρίσκεται στην κρίση σας. Παρακάτω προσφέρονται μερικές συμβουλές ή αναφορές σε ήδη υπάρχων κώδικα του scheduler που μπορεί να σας βοηθήσουν.

- **include/linux/sched.h**
  - struct task_struct;
  - struct rq;
  - struct sched_entity;
  - struct sched_class;
  - RR_TIMESLICE; Το κβάντο χρόνο για τις real-time διεργασίες. Αυτό μπορεί να αλλάξει κατά την εκτέλεση ανάλογα με τις ανάγκες τις διεργασίες.

- **kernel/sched.c**
  - asmlinkage void __sched schedule(void); -- Main schedule function
  - context_switch(rq, prev, next); -- Function to do the actual context switch
  - pre_schedule(rq, prev); -- Ανάλογα το sched_class καλείται και διαφορετική συνάρτηση μέσω ενός function pointer.
  - pick_next_task(rq); -- Η συνάρτηση που διαλέγει την επόμενη διεργασία που θα τρέξει. Διατρέχει την λίστα των διεργασίων που είναι έτοιμες να τρέξουν.
  - post_schedule(rq); -- Παρόμοια με την pre_schedule(rq, prev);

- **kernel/sched_debug.c, kernel/sched_rt.c, kernel/sched_fair.c, kernel/sched_idletask.c**
  Ο κώδικας για το κάθε διαφορετικό είδος διεργασίας του scheduler. Θα πρέπει να βρείτε αυτά τα είδη διεργασιών και να δείτε μήπως κάποιο από αυτά μοιάζει με το δικό σας.

- **include/linux/time.h**
  - Το αρχείο περιέχει αρκετά structs για μετατροπές και προσθέσεις-αφαιρέσεις χρόνων.
  - Struct timespec; -- Εδώ είναι ο ορισμός του field που εμφανίζεται πολλές φορές μέσα στο struct task_struct;

Όπως εξηγείται και στο φροντιστήριο της άσκησης οι διεργασίες στο συγκεκριμένο kernel επιλέγονται βάση προτεραιότητας που ορίζεται σε ένα field του task_struct. Ανάλογα με την προτεραιότητα της κάθε διεργασίας εκτελείται και ένα διαφορετικό είδος scheduling. Είναι δυνατό να χρησιμοποιήσετε αυτή την ιδιότητα προς όφελος σας με την προϋπόθεση ότι ο scheduler θα συνεχίσει να επιλέγει πρώτα τις διεργασίες με μεγαλύτερη προτεραιότητα. Μόνο στην περίπτωση που δύο διεργασίες έχουν την ίδια προτεραιότητα θα πρέπει να διαλέγει την periodic έναντι της κανονικής.

Testing:

Σε αυτή την άσκηση θα παραδώσετε και τον κώδικα που θα γράψετε για να τεστάρετε την σωστή συμπεριφορά του scheduler σας. Παρακάτω παρέχονται μερικές ιδέες για να ελέγξετε ότι ο scheduler λειτουργεί φυσιολογικά.

- Δημιουργείστε ένα test πρόγραμμα που να είναι κάπως έτσι:
  Λαμβάνει σαν παραμέτρους το p_i και το c_i και μετά εκτελεί ένα ατέρμονο loop.

```
set_period_parameters ( -1, argument_p_i, argument_c_i );
while (1) {
        x++;
}
```

Ορίστε σαν p_i 1000 και c_i 200 και ξεκινήστε την 1η test διεργασία. Λογικά με μία μόνο διεργασία δεν θα πρέπει να χάνετε deadlines. Κρατώντας την 1η test διεργασία ενεργή(να τρέχει) ξεκινήστε και άλλες με παραμέτρους p_i/c_i όπως 500/2000, 400/1500, 200/1000. Σε κάποια φάση κάποια από όλες τις διεργασίες θα αρχίσει να χάνει deadlines και θα μπορέσετε να το δείτε διαβάζοντας το dmesg όπου θα εκτύπωνετε κάθε φορά που μια διεργασία χάνει ένα deadline ή εκτελώντας το δικό σας system call για να διαβάσετε τα missed deadlines.

- Δημιουργείστε αρκετές διεργασίες ή δημιουργείτε αρκετό CPU load μέσω αριθμητικών ή άλλων πράξεων μέχρι το σημείο που οι περιοδικές σας διεργασίες αρχίσουν να χάνουν deadline. Μετά εκτυπώστε χρησιμοποιώντας το σωστό system call τον αριθμό των missed deadlines.
- Το συγκεκριμένο linux kernel λειτουργεί με μεταβλητού χρόνου quantum ανά διεργασία. Συνήθως αυτό είναι 100ms. Μπορείτε να ορίσετε διεργασίες που το period time τους είναι μικρότερο από αυτό και να δείτε αν χάνουν deadlines.

Σημειώσεις:

Παρακάτω υπάρχει μια λίστα από links τα οποία μπορεί να σας βοηθήσουν πολύ για να βρείτε την λύση. Προϋπόθεση για αυτό είναι να τα διαβάσετε.

Ο χρόνος p_i & c_i δεν μοιράζεται μεταξύ μιας διεργασίας και των θυγατρικών της. Είναι δυνατό μια διεργασία να ορίσει αυτές τις τιμές για τις θυγατρικές της αλλά δεν θα μοιραστεί το c_i χρόνο της με αυτές.

Ο GCC compiler προσπαθεί να βελτιώσει τον κώδικα που δημιουργεί. Στην περίπτωση των test προγραμμάτων σας αυτό μπορεί να σημαίνει ότι ο compiler θα αγνοεί το while loop. Βάζοντας σαν flag το –O0 μπορείτε να απενεργοποιήσετε τα optimizations. Αυτό δεν χρειάζεται να το κάνετε κατά το compilation του kernel σας. Μόνο στα test προγράμματα σας.

Για τον τρόπο λειτουργίας του VM και του QEMU καθώς και άλλες σχετικές πληροφορίες μπορείτε να δείτε την εκφώνηση της προηγούμενης άσκησης.

Η άσκηση αυτή υποθέτει ότι έχει γίνει η υλοποίηση της άσκησης 3. Σε περίπτωση που δεν έχετε υλοποιήσει την προηγούμενη άσκηση δεν είναι υποχρεωτικό να ακολουθήσετε το προηγούμενο μοντέλο αλλά θα το προτείναμε.

Links:

- Completely Fair Scheduler
- *kernel.org CFS scheduler - Check 6. SCHEDULING CLASSES*
- Tuning the Task Scheduler – Read all but focus on 14.4
- Columbia University - Linux Scheduler
- Linux Data Structures
- Understanding the structure task_struct
- A study on Linux kernel scheduler 2.6.32
- Preemption on Computing

https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt

Cool reads:

- Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment
- Inside the Linux 2.6 Completely Fair Scheduler

Παραδωτέα:

Αφού κάνετε αυτήν την άσκηση θα πρέπει να παραδώσετε τα παρακάτω:

1. Το καινούργιο kernel image, δηλαδή το αρχείο linux-2.6.38.1/arch/x86/boot/bzImage.
2. Όλα τα αρχεία που τροποποιήσατε ή δημιουργήσατε στον source code του Linux kernel 2.6.38.1 για να υλοποιήσετε τα system calls. Δηλαδή όλα τα αρχεία .c, .h, Makefile κτλ που κάνατε κάποια αλλαγή, ή δημιουργήσατε εσείς. Μην παραδώσετε αρχεία που δεν χρειάστηκε να τα τροποποιήσετε για την υλοποίησή σας.
3. Τον source code από όλα τα test προγράμματα που γράψατε και τρέξατε μέσα στο guest Linux OS για να δοκιμάσετε τα system calls και τον scheduler που υλοποιήσατε. Και επίσης ότι header files χρησιμοποιήσατε για type και function definitions (π.χ. το unist.h). Δηλαδή τα αρχεία .c, .h και Makefile και ότι άλλο αρχείο δημιουργήσατε στο guest OS για να δοκιμάσετε το scheduler (εκτός από τα executables).
4. Ένα README file στο οποίο να περιγράφετε συνοπτικά (αλλά περιεκτικά και ξεκάθαρα) όλα τα βήματα που ακολουθήσατε για την δημιουργία των καινούργιων system calls. Επίσης πρέπει να σχολιάσετε τι παρατηρήσατε από τα test προγράμματα που τρέξατε. Αν έχετε κάνει κάτι διαφορετικό ή παραπάνω από όσα αναφέρουμε στην εκφώνηση της άσκησης σε οποιοδήποτε βήμα μπορείτε επίσης να το αναφέρετε στο README. Καλό θα ήταν το README να είναι από 20 μέχρι 30 γραμμές.

Μπορείτε να φτιάξετε έναν κατάλογο με τα τροποποιημένα source code αρχεία του kernel (αν θέλετε θα είναι καλό να κρατήσετε την δομή καταλόγων που υπάρχει στον linux kernel), έναν κατάλογο με τα test προγράμματα και header files από το guest OS, και τέλος να τους μεταφέρετε σε ένα κατάλογο μαζί το bzImage και το README file για να παραδώσετε την άσκηση με τον γνωστό τρόπο.

Προσοχή:

1. ΔΕΝ χρειάζεται να παραδώσετε το disk image (hy345-linux.img) ακόμα και αν αυτό έχει τροποποιηθεί (όντως, το disk image μπορεί να αλλάξει όσο χρησιμοποιείτε το guest OS, σαν ένας κανονικός δίσκος, αλλά δεν χρειάζεται να το παραδώσετε).
2. ΔΕΝ χρειάζεται επίσης να παραδώσετε κάποιο αρχείο με ολόκληρο τον source code του Linux kernel, πρέπει να σημειώσετε και να παραδώσετε μόνο τα αρχεία που τροποποιήσατε ή δημιουργήσατε. Το kernel image (bzImage), τα header files, και τα test προγράμματα που θα παραδώσετε θα πρέπει να είναι αρκετά ώστε η άσκησή σας να μπορεί να τρέξει με το αρχικό disk image και το QEMU, έτσι ώστε να φαίνετε η σωστή υλοποίηση του ζητούμενου scheduler.

Παρατηρήσεις:

1. Η άσκηση είναι ατομική. Τυχόν αντιγραφές μπορούν να ανιχνευθούν εύκολα από κατάλληλο πρόγραμμα και θα μηδενιστούν. Συμπεριλάβετε το όνομα σας και το λογαριασμό σας (account) σε όλα τα αρχεία.

2. Γράψτε ένα αρχείο README, το πολύ 30 γραμμών, με επεξηγήσεις για τον τρόπο υλοποίησης του scheduler.
3. Κατασκευάστε ένα αρχείο Makefile, έτσι ώστε πληκτρολογώντας make all να γίνεται η μεταγλώττιση (compilation) των προγραμμάτων που χρησιμοποιούν τα system calls και να παράγονται τα εκτελέσιμα αρχεία. Επίσης πληκτρολογώντας make clean να καθαρίζονται όλα τα περιττά αρχεία, και να μένουν μόνο τα αρχεία που χρειάζονται για τη μεταγλώττιση.
4. Τοποθετήστε σε ένα κατάλογο όλα τα αρχεία προς παράδοση για την άσκηση 3. Παραδώστε τα παραπάνω αρχεία χρησιμοποιώντας το πρόγραμμα turnin (πληκτρολογήστε turnin assignment_4@hy345 directory_name από τον κατάλογο που περιέχει τον κατάλογο directory_name με τα αρχεία της άσκησης).
5. Σε πολλές περιπτώσεις τα ονόματα των αρχείων είναι ενδεικτικά. Μπορείτε να χρησιμοποιήσετε όποια σας βολεύουν.