HY-345: Λειτουργικά Συστήματα
Χειμερινό Εξάμηνο 2022
Άσκηση 4

Implementation of the “Shortest Task First” Scheduling Policy in the Linux Operating System

Φροντιστήριο: 29/11/2022
Παράδοση: 21/12/2022

Εισαγωγή

Ο πυρήνας του λειτουργικού συστήματος Linux περιέχει έναν χρονοπρογραμματιστή (**Scheduler**) ο οποίος αποφασίζει ποια θα είναι η επόμενη διεργασία η οποία θα εκτελεστεί στον επεξεργαστή του υπολογιστή (CPU). Ο χρονοπρογραμματιστής παίρνει αποφάσεις σύμφωνα με την πολιτική χρονοπρογραμματισμού (**Scheduling Policy**) η οποία βοηθάει στην αποδοτική χρήση του επεξεργαστή. Στην άσκηση αυτή θα υλοποιήσετε μια νέα πολιτική χρονοπρογραμματισμού ως μέρος του λειτουργικού συστήματος Linux.

Shortest Task First

Στην άσκηση αυτή σας ζητείται να υλοποιήσετε τον αλγόριθμο χρονοπρογραμματισμού “Shortest Task First”. Σύμφωνα με τον αλγόριθμο, κάθε διεργασία οφείλει να δηλώσει ποια είναι η προθεσμία **D** (deadline) που θα πρέπει να έχει ολοκληρωθεί καθώς και ποιος είναι ο εκτιμώμενος χρόνος εκτέλεσης **T** (estimated runtime) που χρειάζεται για να ολοκληρωθεί.

Σύμφωνα με τον αλγόριθμο “Shortest Task First”, για κάθε διεργασία θα υπολογίσει τον απομένων χρόνο εκτέλεσης **R** (remaining time). Η τιμή αυτή υπολογίζεται ως το estimated runtime T μείον τον χρόνο που έχει καταναλώσει ήδη μια διεργασία. Εάν η τιμή αυτή είναι μη-θετική, τότε η διεργασία έχει εκτελεστεί για τον χρόνο που είχε δηλώσει αρχικά και ο χρονοπρογραμματιστής την τερματίζει τυπώνοντας το κατάλληλο μήνυμα. Διαφορετικά, ο αλγόριθμος υπολογίζει το περιθώριο **S** κάθε διεργασίας ώς $S = D - R$. Στην συνέχεια, επιλέγει να εκτελέσει την διεργασία με το μικρότερο περιθώριο S.

Ο αλγόριθμος αυτός ελέγχει και την προθεσμία κάθε διεργασίας. Εάν μια διεργασία δεν καταφέρει να εκτελεστεί για όλο τον χρόνο εκτέλεσης που έχει δηλώσει αρχικά και περάσει η προθεσμία που είχε δηλώσει, τότε ο χρονοπρογραμματιστής θα πρέπει να την τερματίζει και να τυπώνει το κατάλληλο μήνυμα. Τέλος, ο αλγόριθμος είναι **preemptive**. Αυτό σημαίνει ότι εάν υπάρχει μια διεργασία που έχει μεγαλύτερη προτεραιότητα σύμφωνα με την πολιτική Shortest Task First, τότε ο χρονοπρογραμματιστής θα διακόψει την τρέχουσα διεργασία και θα την αντικαταστήσει με αυτήν με την μεγαλύτερη προτεραιότητα.

Παράδειγμα Εκτέλεσης

Έστω δύο διαφορετικές διεργασίας P_0 και P_1.

1. Η διεργασία P_0 ξεκινάει να εκτελείται την χρονική στιγμή 0 και ορίζει προθεσμία D σε 30 δευτερόλεπτα από τώρα ενώ ο αναμενόμενος χρόνος εκτέλεσης είναι 10 δευτερόλεπτα.
2. Καθώς δεν υπάρχει άλλη διεργασία να εκτελεστεί ο χρονοπρογραμματιστής την επιλέγει αυτομάτως να εκτελεστεί στον επεξεργαστή.
3. Μετά από 5 δευτερόλεπτα εκτέλεσης, δημιουργείται η διεργασία P_1 και ορίζει προθεσμία D σε 20 δευτερόλεπτα και χρόνο εκτέλεσης 5 δευτερόλεπτα. Ο χρονοπρογραμματιστής θα πρέπει τώρα να αποφασίσει ποια διεργασία θα εκτελεστεί.
4. Ο χρονοπρογραμματιστής θα υπολογίσει τις αντίστοιχες τιμές για κάθε διεργασία.
    - Για την διεργασία P_0 ισχύει $R_0 = 10 - 5 = 5$ και $S_0 = 30 - 5 = 25$.
    - Για την διεργασία P_1 ισχύει $R_1 = 5 - 0 = 5$ και $S_1 = 20 - 5 = 15$.
5. Ο χρονοπρογραμματιστής αποφασίζει ότι η διεργασία P_1 πρέπει να πάρει πλέον χρόνο στον επεξεργαστή. Συνεπώς, διακόπτει την διεργασία P_0 και την αντικαθιστά με την P_1.

Τροποποιήσεις στον πυρήνα του Linux

Για την άσκηση αυτή, θα χρησιμοποιήσετε τον κώδικα της άσκησης 3 ως βάση. Επιπλέον, θα χρησιμοποιήσετε τον emulator QEMU καθώς και το virtual disk image που χρησιμοποιήσατε στην άσκηση 3. Οδηγίες σχετικά με την μεταγλώττιση του Linux Kernel και την χρήση του virtual disk image μπορείτε να βρείτε στην εκφώνηση της προηγούμενης άσκησης.

Για την άσκηση αυτή χρειάζεται να τροποποιήσετε τον πυρήνα του λειτουργικού συστήματος και να υλοποιήσετε την νέα πολιτική χρονοπρογραμματισμού. Για την σωστή λειτουργία της πολιτικής αυτής θα γίνει χρήση των system calls που υλοποιήσατε στην προηγούμενη άσκηση. Η κυριότερη συνάρτηση του Linux Scheduler καθώς και το σημείο εισόδου είναι η συνάρτηση `void __sched schedule(void)` στο αρχείο `kernel/sched.c`.

Τα βήματα που πρέπει να ακολουθήσετε είναι:

1. Ο scheduler εντοπίζει τις διεργασίας που χρησιμοποιούν την πολιτική μας ελέγχοντας ποιες από αυτές έχουν ορίσει προθεσμία D και εκτιμώμενος χρόνος εκτέλεσης T.
    - Οι διεργασίες αυτές έχουν προτεραιότητα σε σχέση με τις υπόλοιπες που τρέχουν στο σύστημα. Εάν δεν υπάρχει κάποια “προνομιούχα” διεργασία ο scheduler δεν χρειάζεται να αλλάξει συμπεριφορά και γι’ αυτό χρησιμοποιεί τον προκαθορισμένο αλγόριθμο χρονοπρογραμματισμού.
2. Για κάθε μια από “προνομιούχες” διεργασίες, υπολογίζει πόσο χρόνο έχουν εκτελεστεί ήδη στον επεξεργαστή.
3. Για κάθε μια από τις διεργασίας υπολογίζει τον απομένων χρόνο εκτέλεσης R. Εάν κάποια διεργασία έχει αρνητική τιμή R την τερματίζει.
4. Για κάθε μια από τις διεργασίες υπολογίζει το περιθώριο S. Η διεργασία που έχει την χαμηλότερη τιμή S επιλέγεται από τον αλγόριθμο για να εκτελεστεί.
    - Σε αυτή την περίπτωση, ο χρονοπρογραμματιστής θα διακόψει την διεργασία χαμηλότερης προτεραιότητας που τρέχει ήδη στον επεξεργαστή ώστε να εκτελεστεί η διεργασία που επέλεξε.

Υλοποίηση

Στόχος της άσκησης είναι υλοποιήσετε τον ζητούμενο αλγόριθμο χρονοπρογραμματισμού. Η υλοποίηση που θα κάνετε βρίσκεται στην κρίση σας και δεν υπάρχει μια σωστή υλοποίηση. Ακολουθούν κάποιες χρήσιμες δομές και συναρτήσεις που μπορεί να σας βοηθήσουν.

File	Entity	Description
include/linux/sched.h	struct task_struct	Process descriptor. Each process is represented as such a struct. It offers all the information about one particular task (i.e. process) such as pid, state, parent process, children, opened files, etc.
	struct rq	Runqueue. It is the main data structure in process scheduling. It manages active processes by holding the tasks that are in a runnable state at any given moment of time.
	struct sched_entity	CFS works with more general entities than tasks. This struct contains attributes for accounting run time of processes.
	struct sched_class	The current Linux scheduler has been designed with an extensible hierarchy of modules in mind. These modules encapsulate scheduling policy details. Scheduling classes are implemented through the sched_class structure, which contains hooks to the functions that implement the policy.
kernel/sched.c	schedule(void)	Main function of the Linux scheduler. Responsible for implementing the process scheduling functionality.
	void context_switch(...)	Performs the actual context switch operation by switching from the old task_struct to the new one.
	pick_next_task(...)	Selects task_struct of the next process that will run on the processor. Iterates over the list of processes in the runnable state.

Επιπλέον, μπορείτε να εξετάσετε το αρχείο `kernel/sched_rt.c` αλλά και το `kernel/sched_fair.c` που υλοποιούν το Real-Time Scheduling Class και τον Completely Fair Scheduler αντιστοίχως. Οι υλοποιήσεις τους μπορεί να σας βοηθήσουν να καταλάβετε πως λειτουργεί ο Linux scheduler αλλά και πως γίνεται το process management. Τέλος, στο αρχείο include/linux/time.h υπάρχουν διάφορα structs για μέτρηση χρόνου καθώς και διάφορες συναρτήσεις για μετατροπές μεταξύ μονάδων μέτρησης.

Παρατηρήσεις

- Φροντίστε στην υλοποίηση σας να μην γίνονται **starve** οι υπόλοιπες διεργασίες του συστήματος. Αυτό μπορείτε να το επιτύχετε είτε θέτοντας κατάλληλες παραμέτρους μέσω των system calls είτε εναλλάσσοντας μεταξύ policies που επιλέγονται στον scheduler.

- Για την άσκηση αυτή θα χρησιμοποιήσετε τον Linux kernel 2.6.38.1 Μπορείτε να χρησιμοποιήσετε το Elixir platform για να περιηγηθείτε στον κώδικα του Linux Kernel.

- Για να πάρετε το task_struct της τρέχουσας διεργασίας που έκανε το system call μπορείτε κοιτάξτε στο αρχείο `arch/x86/include/asm/current.h`

- Ο Linux kernel αποθηκεύει τις αναλυτικές πληροφορίες για όλες τις τρέχουσες διεργασίες σε μία λίστα από task_struct objects. Για να προσπελάσετε όλες τις διεργασίες του συστήματος στη λίστα αυτή, μπορείτε να χρησιμοποιήσετε το macro `for_each_process`.

- Χρησιμοποιήστε την συνάρτηση printk για να ελέγξετε τη σωστή λειτουργία της υλοποίησης σας. Για να δείτε τα μηνύματα αυτά μπορείτε να χρησιμοποιήσετε το `dmesg` ή να εκτελέσετε την εντολή "`cat /var/log/messages`".

Demo Programs

Πρέπει να φτιάξετε και να παραδώσετε τουλάχιστον ένα δοκιμαστικό πρόγραμμα το οποίο θα κάνει χρήση των system calls της προηγούμενης άσκησης και θα επιδεικνύει την σωστή υλοποίηση του νέου αλγόριθμου χρονοπρογραμματισμού. Ενδεικτικά, αναφέρονται κάποιες περιπτώσεις που μπορείτε να ελέγξετε.

1. Δημιουργήστε ένα απλό πρόγραμμα το οποίο καλεί το system call `set_task_params` (από την προηγούμενη άσκηση) και ορίζει τις παραμέτρους χρονοπρογραμματισμού. Ελέγξτε ότι η υλοποίηση σας διαχειρίζεται σωστά τις παραμέτρους αυτές και μπορεί να υπολογίζει σωστά το περιθώριο S όπως ορίζει ο αλγόριθμος.
2. Δημιουργήστε ένα απλό πρόγραμμα το οποίο ορίζεις τις παραμέτρους του και βεβαιωθείτε ότι η δική σας υλοποίηση καλείται σωστά όταν εντοπίζει μια τέτοια διεργασία.
3. Δημιουργήστε πολλαπλά processes που ορίζουν διαφορετικές παραμέτρους η κάθε μια και βεβαιωθείτε ότι η υλοποίηση σας δουλεύει σωστά και επιλέγει σωστά ποια διεργασία θα εκτελεστεί πρώτα ανάλογα με την προτεραιότητα τους.
    - Οι διεργασίες σας πρέπει να απαιτούν την χρήση του επεξεργαστή (spinning) και να μην βρίσκονται σε sleep state. Αυτό μπορείτε να το πετύχετε με την χρήση ενός loop και αριθμητικών πράξεων.
4. Δημιουργείστε πολλαπλά processes και ορίσετε έτσι τις παραμέτρους ώστε μια διεργασία να μην προλάβει να εκτελεστεί μέσα στην προθεσμία της. Ελέγξτε ότι όταν περάσει η προθεσμία της ο χρονοπρογραμματιστής την τερματίζει.
5. Για κάθε δοκιμαστικό πρόγραμμα που θα δημιουργήσετε, χρησιμοποιήσετε τα κατάλληλα prints τα οποία θα δείχνουν τις παραμέτρους που έχει ορίσει η κάθε διεργασία καθώς και πότε ξεκινάει να κάνει spin. Τα prints αυτά θα σας βοηθήσουν να καταλάβετε εάν η υλοποίηση σας παίρνει τις σωστές αποφάσεις.

Παράδοση

Αφού κάνετε την άσκηση θα πρέπει να παραδώσετε τα παρακάτω:

1. Το καινούργιο kernel image που προέκυψε από τη μεταγλώττιση, δηλαδή το αρχείο linux-2.6.38.1/arch/x86/boot/bzImage.

2. Όλα τα αρχεία που χρειάστηκε να τροποποιήσετε ή να δημιουργήσετε στον source code του Linux kernel για να υλοποιήσετε τα system calls. Αυτό σημαίνει ότι θα παραδώσετε όλα τα αρχεία .c, .h, Makefile, κλπ στα οποία κάνατε οποιαδήποτε αλλαγή ή δημιουργήσατε εσείς. Προσοχή, **μην** παραδώσετε αρχεία που δεν χρειάστηκε να τα τροποποιήσετε για την υλοποίησή σας (π.χ. όλο το υπόλοιπο source tree του kernel).

3. Τον κώδικα από όλα τα test προγράμματα που γράψατε και τρέξατε μέσα στο guest Linux OS για να δοκιμάσετε τα system calls που υλοποιήσατε. Επίσης, ότι header files χρησιμοποιήσατε για type και function definitions αλλά και ότι Mekefiles χρειάζονται για την μεταγλώττιση των προγραμμάτων αυτών. Δεν χρειάζεται να παραδώσετε τα executable αρχεία.

4. Ένα README file στο οποίο να περιγράφετε συνοπτικά (αλλά περιεκτικά και ξεκάθαρα) όλα τα βήματα που ακολουθήσατε για την προσθήκη και υλοποίηση των νεων system calls. Επίσης, πρέπει να σχολιάσετε τι παρατηρήσατε από τα test προγράμματα που τρέξατε. Αν έχετε κάνει κάτι διαφορετικό ή παραπάνω από όσα αναφέρονται στην εκφώνηση της άσκησης σε οποιοδήποτε βήμα μπορείτε επίσης να το αναφέρετε στο README. Λόγω της πολυπλοκότητας της άσκησης αυτής, προτείνεται να αναφέρετε στο README και όποιες προσπάθειες υλοποίησης κάνατε ακόμα κι αν αυτές δεν σας οδήγησαν κάπου ή δεν δούλευαν σωστά.

5. Μπορείτε να φτιάξετε έναν κατάλογο με τα τροποποιημένα αρχεία του kernel (αν θέλετε θα είναι καλό να κρατήσετε και την δομή των αρχείων μέσα στον πυρήνα) καθώς και έναν κατάλογο με τα test προγράμματα και header files από το guest OS.

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 αλλά δεν χρειάζεται να το παραδώσετε.

2. **ΔΕΝ** χρειάζεται 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 source και header files καθώς 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 φαίνεται η σωστή υλοποίηση της άσκησης.

Resources

Μπορείτε να χρησιμοποιήσετε τις παρακάτω πηγές για να καταλάβετε καλύτερα πως δουλεύει ο Linux Scheduler καθώς και διάφορα components και συναρτήσεις του.

[1] Completely Fair Scheduler
[2] Kernel Scheduler Documentation
[3] Columbia University - Linux Scheduler
[4] Linux Processes and Scheduling
[5] Linux Data Structures
[6] Process Management and Process Descriptor
[7] A study on Linux Kernel Scheduler version 2.6.32
[8] A complete guide to Linux process scheduling
[9] Kernel Scheduling Classes Documentation
[10] Completely Fair Scheduler Internals