

HY-345: Λειτουργικά Συστήματα

Χειμερινό Εξάμηνο 2025

Άσκηση 4

Implementation of the “Least Tolerance First” Scheduling Policy in the Linux Operating System

Φροντιστήριο: 25/11/2025

Παράδοση: 16/12/2025

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.

Least Tolerance First

Στην άσκηση αυτή σας ζητείται να υλοποιήσετε τον αλγόριθμο χρονοπρογραμματισμού “Least Tolerance First” (LTF). Σύμφωνα με τον αλγόριθμο, κάθε διεργασία οφείλει να δηλώσει ποια είναι η προθεσμία **D** (deadline) μέσα στην οποία θέλει να έχει ολοκληρωθεί, καθώς και ποιος είναι ο εκτιμώμενος χρόνος εκτέλεσης της **E** (estimated execution time). Ο αλγόριθμος “Least Tolerance First” δίνει δυναμικά προτεραιότητα σε διεργασίες ανάλογα με τον χρόνο που απομένει μέχρι την προθεσμία και πόσος χρόνος εκτέλεσης απομένει στην διεργασία.

Σύμφωνα με τον αλγόριθμο Least Tolerance First, η κάθε διεργασία χαρακτηρίζεται από μια τιμή **T** (tolerance) η οποία ορίζει ποια διεργασία θα εκτελεστεί πρώτη. Η τιμή **T** ορίζεται ως
$$T = \frac{D - \text{Current Time}}{\text{Remaining Execution Time}}$$
 όπου **D** είναι η προθεσμία της διεργασίας, **Current Time** είναι ο τρέχων χρόνος του συστήματος και **Remaining Execution Time** είναι ο υπολογιστικός χρόνος που απομένει στη διεργασία να εκτελέσει. Ο αλγόριθμος Least Tolerance First επιλέγει να εκτελέσει πάντα την διεργασία με την μικρότερη τιμή **T**.

Σε κάθε απόφαση χρονοπρογραμματισμού ο χρονοπρογραμματιστής υπολογίζει την τιμή **T** για όλες τις διεργασίες που έχουν δηλώσει προθεσμία και χρόνο εκτέλεσης και επιλέγει την διεργασία με την μικρότερη τιμή **T** για να εκτελεστεί. Μικρή τιμή **T** σημαίνει ότι η διεργασία είναι πιο κρίσιμη γιατί πλησιάζει η προθεσμία της οπότε πρέπει να εκτελεστεί άμεσα.

Ο αλγόριθμος Least Tolerance First είναι υπεύθυνος και για τον χρόνο εκτέλεσης της κάθε διεργασίας. Όταν ο αλγόριθμος έχει επιτρέψει σε μια διεργασία να εκτελεστεί για τον χρόνο που εκείνη δήλωσε, τότε ο αλγόριθμος οφείλει να τερματίσει (σκοτώσει) την διεργασία και να τυπώσει το κατάλληλο μήνυμα. 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 Least Tolerance First, τότε ο χρονοπρογραμματιστής θα διακόψει την τρέχουσα διεργασία και θα την αντικαταστήσει με αυτήν με την μεγαλύτερη προτεραιότητα.

Παράδειγμα Εκτέλεσης

Έστω δύο διαφορετικές διεργασίες P_1 και P_2 . Η διεργασία P_1 δηλώνει χρόνο εκτέλεσης 5s και προθεσμία 10s. Η διεργασία P_2 δηλώνει χρόνο εκτέλεσης 4s και προθεσμία 11s. Θεωρούμε ότι και οι δύο διεργασίες ξεκινούν την εκτέλεση τους ταυτόχρονα την χρονική στιγμή 0s. Συνεπώς, ο χρονοπρογραμματιστής θα πρέπει τώρα να αποφασίσει ποια διεργασία θα εκτελεστεί.

1. Την χρονική στιγμή $t=0$
 - ο Η διεργασία P_1 έχει $T = \frac{10-0}{5} = 2$
 - ο Η διεργασία P_2 έχει $T = \frac{11-0}{4} = 2.75$
2. Ο χρονοπρογραμματιστής αποφασίζει ότι η διεργασία P_1 έχει μεγαλύτερη προτεραιότητα οπότε θα την επιλέξει για εκτέλεση.
3. Την χρονική στιγμή $t=1$
 - ο Η διεργασία P_1 έχει $T = \frac{10-1}{4} = 2.25$
 - ο Η διεργασία P_2 έχει $T = \frac{11-1}{4} = 2.5$
4. Ο χρονοπρογραμματιστής αποφασίζει ότι η διεργασία P_1 εξακολουθεί να έχει μεγαλύτερη προτεραιότητα οπότε θα συνεχίσει την εκτέλεση της.
5. Την χρονική στιγμή $t=2$
 - ο Η διεργασία P_1 έχει $T = \frac{10-2}{3} = 2.666$
 - ο Η διεργασία P_2 έχει $T = \frac{11-2}{4} = 2.3$
6. Ο χρονοπρογραμματιστής αποφασίζει πως η διεργασία P_2 έχει τώρα εκείνη μεγαλύτερη προτεραιότητα οπότε πρέπει να πάρει χρόνο στον επεξεργαστή. Συνεπώς, διακόπτει την διεργασία P_1 και την αντικαθιστά με την P_2 .

Η εκτέλεση των διεργασιών συνεχίζει σύμφωνα με τους κανόνες του αλγορίθμου Least Tolerance First μέχρι να μην υπάρχει άλλη διεργασία που να μπορεί να εκτελεστεί (είτε επειδή ολοκλήρωσε τον χρόνο εκτέλεσης της είτε επειδή πέρασε η προθεσμία της).

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 διεργασίες που χρησιμοποιούν την πολιτική LTF ελέγχοντας ποιες από αυτές έχουν ορίσει προθεσμία D και χρόνο εκτέλεσης E.
 - ο 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 διεργασίες υπολογίζει τον απομένον χρόνο εκτέλεσης R. Η χρονική στιγμή αυτή εξαρτάται από τον χρόνο που έχει εκτελεστεί ήδη η διεργασία και τον χρόνο που έχει ζητήσει να εκτελεστεί συνολικά.
4. Εάν η διεργασία έχει ξεπεράσει την προθεσμία της, ο χρονοπρογραμματιστής την τερματίζει. Εάν κάποια διεργασία έχει ήδη ολοκληρώσει τον χρόνο εκτέλεσης που είχε ζητήσει, ο χρονοπρογραμματιστής την τερματίζει.
5. Για κάθε μια από τις διεργασίες υπολογίζει την τιμή T. Η διεργασία που έχει την χαμηλότερη τιμή T επιλέγεται από τον αλγόριθμο για να εκτελεστεί.
 - ο Σε αυτή την περίπτωση, ο χρονοπρογραμματιστής θα διακόψει την διεργασία χαμηλότερης προτεραιότητας που τρέχει ήδη στον επεξεργαστή ώστε να εκτελεστεί η διεργασία που επέλεξε.

Στον αλγόριθμο LTF, μία διεργασία με τιμή T μικρότερη από 1 έχει λιγότερο χρόνο μέχρι την προθεσμία από ότι χρόνο εκτέλεση που της απομένει. Αυτό σημαίνει ότι δεν μπορεί να προλάβει την προθεσμία της. Στα πλαίσια αυτής της άσκησης, μπορείτε να χειριστείτε τέτοιες περιπτώσεις με έναν από τους παρακάτω τρόπους:

1. Ο χρονοπρογραμματιστής τερματίζει κατευθείαν την διεργασία γιατί δεν μπορεί να πετύχει την προθεσμία της.
2. Ο χρονοπρογραμματιστής αγνοεί την συγκεκριμένη διεργασία και επιλέγει κάποια άλλη η οποία μπορεί να έχει μεγαλύτερη τιμή T.
3. Ο χρονοπρογραμματιστής επιλέγει κανονικά την συγκεκριμένη διεργασία γνωρίζοντας ότι δεν θα προλάβει την προθεσμία της και ότι μπορεί να καθυστερήσει κάποια άλλη.

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 υλοποίησή 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 κοιτάξετε 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, μπορείτε να χρησιμοποιήσετε το macro for_each_process.
- Χρησιμοποιήστε την συνάρτηση printk για να ελέγξετε τη σωστή λειτουργία της υλοποίησής 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 την set_proc_info (από την προηγούμενη άσκηση) και ορίζει τις παραμέτρους χρονοπρογραμματισμού. Ελέγξτε ότι η υλοποίησή σας διαχειρίζεται σωστά τις παραμέτρους αυτές και μπορεί να υπολογίζει σωστά την τιμή T όπως ορίζει ο αλγόριθμος LTF.
2. Δημιουργήστε ένα απλό πρόγραμμα το οποίο ορίζει τις παραμέτρους του και βεβαιωθείτε ότι η δική σας υλοποίηση του αλγόριθμου καλείται σωστά όταν εντοπίζει μια τέτοια διεργασία.
3. Δημιουργήστε πολλαπλά processes που ορίζουν διαφορετικές παραμέτρους η κάθε μια και βεβαιωθείτε ότι η υλοποίησή 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. Ορίστε για μια διεργασία συγκεκριμένο χρόνο εκτέλεσης και αφήστε την να εκτελεστεί για τουλάχιστον τόσο χρόνο. Ελέγξτε ότι όταν η διεργασία εκτελεστεί για τον χρόνο που ζητήσε, 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 Makefiles 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 νέων 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](#)