ΗΥ-345 Λειτουργικά Συστήματα
Χειμερινό Εξάμηνο 2022
Άσκηση 3

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

Φροντιστήριο: 15/11/2022
Παράδοση: 06/12/2022

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.

Μια πολιτική χρονοπρογραμματισμού διεργασιών είναι η “Shortest Task First”. Σύμφωνα με την πολιτική αυτή, κάθε διεργασία δηλώνει στον χρονοπρογραμματιστή ποιό είναι το **deadline** της (μέχρι ποιά χρονική στιγμή θα πρέπει να έχει εκτελεστεί) αλλά και πόσο χρόνο υπολογίζεται ότι χρειάζεται για να εκτελεστεί (**estimated runtime**). Για την άσκηση αυτή, υποθέστε ότι το deadline μετριέται σε δευτερόλεπτα (seconds), ενώ το estimated runtime μετριέται σε milliseconds.

Σε κάποια χρονική στιγμή, υπάρχει περίπτωση να πρέπει να εκτελεστούν πολλαπλές διεργασίες. Σε τέτοιες περιπτώσεις ο χρονοπρογραμματιστής δίνει προτεραιότητα ανάλογα με τα δύο αυτά πεδία της κάθε διεργασίας. Περισσότερες πληροφορίες για το πως θα υπολογίζεται η προτεραιότητα κάθε διεργασίας θα σας δοθούν στην επόμενη άσκηση (4η).

Σε αυτήν την άσκηση θα πρέπει να υλοποιήσετε δύο καινούργια system calls, τα οποία θα χρησιμοποιήσετε και στην επόμενη άσκηση του μαθήματος. Πιο συγκεκριμένα, θα χρειαστεί να προσθέσετε στον πυρήνα (kernel) του λειτουργικού συστήματος Linux τα δύο νέα system calls: **“set_task_params”** και **“get_task_params”**. Έπειτα θα τα δοκιμάσετε, χρησιμοποιώντας τον προσομοιωτή (emulator) QEMU, κάνοντας compile τον Linux kernel με τις αλλαγές σας και τρέχοντας το Linux με τον καινούργιο πυρήνα. Τέλος, καλείστε να γράψετε και να τρέξετε ένα user-level demo πρόγραμμα που θα χρησιμοποιεί τα καινούργια system calls. Το πρόγραμμα αυτό θα εκτελεστεί στο Linux λειτουργικό σύστημα που θα φορτώσετε με τον QEMU. Στόχος της άσκησης είναι να εξοικειωθείτε με τον source code του Linux kernel, με τον τρόπο που ορίζονται και υλοποιούνται τα system calls στο Linux, με την μεταγλώττιση του kernel και με τη χρήση ενός προσομοιωτή.

Υλοποίηση

Καλείστε να προσθέσετε την παρακάτω λειτουργικότητα στον Linux kernel:

- Μέσα στον Linux kernel, κάθε διεργασία αναπαριστάται ως ένα `task_struct`. Αυτό το struct περιέχει όλη την απαραίτητη πληροφορία για μια διεργασία (π.χ. process id, parent process, κλπ). Σε αυτό το struct πρέπει να προσθέσετε δύο νέα πεδία:

  ```
  int deadline              /* το deadline της διεργασίας, σε seconds από τώρα */
  int estimated_runtime     /* ο αναμενόμενος χρόνος εκτέλεσης, σε milliseconds*/
  ```

  Οι τιμές των πεδίων αυτών θα αλλάζουν από το system call `set_task_params` ενώ θα μπορούν να επιστραφούν σε ένα πρόγραμμα μέσω του system call `get_task_params`.

- `set_task_params(int deadline, int estimated_runtime)`
  Το system call αυτό, αλλάζει τις τιμές των αντίστοιχων πεδίων στο task_struct της συγκεκριμένης διεργασίας που έκανε το system call. Πρέπει να ελέγχετε ότι οι τιμές των ορισμάτων που δίνονται είναι έγκυρες καθώς επίσης και ότι το estimated_runtime μπορεί να εκπληρωθεί μέσα στο deadline. Το system call πρέπει να επιστρέφει 0 εάν οι πληροφορίες της διεργασίας έχουν αποθηκευτεί επιτυχώς, ενώ θα επιστρέφει `EINVAL` στην περίπτωση σφάλματος.

- `get_task_params(struct d_params * params)`
  Το system call αυτό επιστρέφει μέσω του pointer params τις τιμές που έχουν οριστεί σχετικά με την διεργασία που έκανε το system call. Η μνήμη για το struct αυτό θα δεσμεύεται από το user-level πρόγραμμα που κάλεσε το system call και όχι από τον πυρήνα. Ο πυρήνας απλά θα αλλάζει τις τιμές του struct σύμφωνα με τις αντίστοιχες τιμές που θα βρίσκει στο αντίστοιχο task_struct της διεργασίας και θα επιστρέφει την πληροφορία (by reference) στο user-level πρόγραμμα. Εάν η εκτέλεση του system call είναι επιτυχής, τότε επιστρέφει την τιμή 0. Αν το όρισμα params είναι NULL, το system call πρέπει να επιστρέφει την τιμή `EINVAL`.

<u>Σημειώσεις</u>:

- Για την άσκηση αυτή θα χρησιμοποιήσετε τον Linux kernel 2.6.38.1

- Το `EINVAL` καθώς και άλλα error values βρίσκονται στο αρχείο:
  linux-2.6.38.1/include/asm-generic/errno-base.h

- Το `task_struct` structure ορίζεται στο αρχείο:
  linux-2.6.38.1/include/linux/sched.h

- 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 που βρίσκονται σε αυτή την λίστα, μπορείτε να χρησιμοποιήσετε το macro for_each_process.

- Σε κάθε κλήση του `set_task_params` και του `get_task_params` πρέπει να τυπώνονται σε επίπεδο πυρήνα το όνομα σας, το Α.Μ. σας και το όνομα του system call. Για να το πετύχετε αυτό, χρησιμοποιήστε την συνάρτηση **`printk`**. Με τον ίδιο τρόπο μπορείτε να τυπώνετε ό,τι άλλα μηνύματα θέλετε από τον kernel (π.χ. ένας απλός τρόπος να κάνετε debugging το system call που φτιάχνετε). Μπορείτε να δείτε τα μηνύματα αυτά όταν έχετε φορτώσει το Linux με το συγκεκριμένο kernel, τρέχοντας το dmesg ή εκτελώντας την εντολή “cat /var/log/messages”.

- Θα πρέπει να ορίσετε το `struct d_params` σε ένα αρχείο που θα δημιουργήσετε εσείς στον φάκελο linux-2.6.38.1/include/linux . Το struct αυτό θα περιέχει 2 πεδία που θα αποθηκεύουν τις παραμέτρους μιας διεργασίας.

Demo programs:

Πρέπει να φτιάξετε και να παραδώσετε τουλάχιστον ένα δοκιμαστικό πρόγραμμα το οποίο θα κάνει χρήση των system calls και θα επιδεικνύει τη σωστή τους υλοποίηση. Για παράδειγμα, το πρόγραμμα σας μπορεί αρχικά να καλεί το `set_task_params` και μετά το `get_task_params` για να τυπώσει όλες τις παραμέτρους που όρισε το προηγούμενο system call. Το πρόγραμμα αυτό θα πρέπει να καλύπτει διάφορες πιθανές κλήσεις των system calls και να παρουσιάζει όλες τις πιθανές περιπτώσεις εκτέλεσης (π.χ. σωστή και λανθασμένη δήλωση ορισμάτων).

Εκτέλεση Linux στο QEMU

Οι emulators είναι διαδεδομένοι για πολλούς λόγους. Μας επιτρέπουν να εγκαταστήσουμε και να τρέξουμε ένα λειτουργικό σύστημα σαν απλοί χρήστες σε έναν υπολογιστή που έχει κάποιο άλλο λειτουργικό σύστημα, χωρίς να χρειαστεί να αλλάξουμε κάτι σε αυτό. Αυτόν τον υπολογιστή μπορεί να τον χρησιμοποιούν αρκετοί χρήστες, και κάθε ένας μπορεί να τρέχει διαφορετικό λειτουργικό σύστημα με έναν emulator χωρίς να επηρεάζονται οι υπόλοιποι χρήστες. Ιδιαίτερα όταν θέλουμε να δοκιμάσουμε κάποιες αλλαγές στον kernel ενός λειτουργικού συστήματος, όπως θα κάνουμε σε αυτήν την άσκηση, ο emulator είναι αρκετά χρήσιμος για έναν ακόμα λόγο. Αν λόγω κάποιου προγραμματιστικού λάθους στον kernel το σύστημα καταρρεύσει (π.χ. kernel panic) μπορούμε εύκολα και γρήγορα να ξεκινήσουμε ξανά το λειτουργικό σύστημα με κάποια αλλαγή (debugging) χωρίς να επηρεαστεί το βασικό λειτουργικό σύστημα του υπολογιστή (host operating system).

Ο QEMU υπάρχει ήδη εγκατεστημένος στα μηχανήματα του τμήματος (man qemu για περισσότερες πληροφορίες). Ο QEMU emulator μπορεί να δημιουργήσει και να διαβάσει έναν εικονικό δίσκο (virtual disk image) και σε αυτόν μπορούμε να εγκαταστήσουμε ένα οποιοδήποτε λειτουργικό σύστημα (π.χ. από ένα εικονικό cd rom). Για τους σκοπούς της άσκησης έχουμε εγκαταστήσει για σας ένα απλό Linux OS (ttylinux distribution) για αρχιτεκτονική 32-bit x86 (i386) σε ένα virtual disk image που θα πρέπει να χρησιμοποιήσετε για την άσκηση σας.

Εσείς θα πρέπει αρχικά να αντιγράψετε αυτό το disk image (63 MB) από την περιοχή του μαθήματος (~hy345/qemu-linux/hy345-linux.img) σε έναν κατάλογο στο /spare του μηχανήματος που χρησιμοποιείτε (π.χ. /spare/[username]), ώστε να μην έχετε πρόβλημα χώρου (π.χ. quota exceeded) με την περιοχή σας.

```
$ cp ~hy345/qemu-linux/hy345-linux.img /spare/[username]/
```

Προσέξτε να έχετε τα κατάλληλα permissions στον κατάλογο αυτό ώστε να έχετε μόνο εσείς πρόσβαση:

```
$ chmod 700 /spare/[username]
```

Το παραπάνω disk image έχει εγκατεστημένο το Linux OS που θα χρησιμοποιήσετε. Περιέχει το root filesystem (/) στο οποίο υπάρχουν τα βασικά προγράμματα και tools του συστήματος. Οπότε, χρησιμοποιώντας το image αυτό μπορείτε να δοκιμάσετε να ξεκινήσετε αυτό το Linux OS με τον QEMU emulator, απλά με την παρακάτω εντολή:

```
$ qemu-system-i386 -hda hy345-linux.img
```

Η παράμετρος `-hda hy345-linux.img` κάνει τον QEMU να χρησιμοποιεί το αρχείο hy345-linux.img σαν virtual disk image, το οποίο θα φαίνεται σαν το device /dev/hda στο emulated OS (guest operating system). Τρέχοντας την παραπάνω εντολή θα δείτε να ξεκινάει το Linux OS που προσομοιώνουμε. Όταν σας ζητήσει να κάνετε login χρησιμοποιήστε το account με username "*user*" και password "*csd-hy345*". Επίσης μπορείτε να κάνετε login και με το account του "*root*" με password "*hy345*". Για να βγείτε από τον QEMU και να κλείσετε το Guest OS χρησιμοποιήστε την εντολή poweroff.

Μεταγλώττιση του Linux kernel

Το επόμενο βήμα είναι να αλλάξετε τον Linux kernel, να τον μεταγλωττίσετε (compile), και να φτιάξετε ένα καινούργιο kernel image με το οποίο θα μπορείτε να ξεκινήσετε το guest Linux OS με τον QEMU, αντί για το original kernel image που υπάρχει στο disk image που σας δίνουμε. Θα μπορούσατε να αλλάξετε και να μεταγλωττίσετε τον kernel μέσα από το guest OS. Για ευκολία όμως θα δουλέψετε στο host OS, δηλαδή κατευθείαν στο μηχάνημα του τμήματος που χρησιμοποιείτε για την άσκηση. Αρχικά θα χρειαστείτε τον source code του Linux kernel 2.6.38.1 για να κάνετε τις απαιτούμενες αλλαγές και να τον κάνετε compile. Οπότε, θα πρέπει να αντιγράψετε τον source code από την περιοχή του μαθήματος στον κατάλογο που χρησιμοποιείται στο /spare/[username] του μηχανήματος.

```
$ cp ~hy345/qemu-linux/linux-2.6.38.1-patched.tar.bz2 /spare/[username]/
```

Αφού αντιγράψετε και κάνετε decompress τον source code του Linux kernel 2.6.38.1, μπορείτε να κάνετε ότι αλλαγές απαιτούνται για την υλοποίηση των καινούργιων system calls. Έπειτα υπάρχουν δύο απλά βήματα που πρέπει να ακολουθήσετε για να κάνετε compile τον αλλαγμένο kernel και να φτιάξετε ένα καινούργιο kernel image: configure και make. Υπάρχουν διάφοροι τρόποι για να κάνει κάποιος configure τον Linux kernel (π.χ. make menuconfig, make config, κτλ). Τελικά το configuration του kernel γράφεται στο αρχείο .config μέσα στον κατάλογο linux-2.6.38.1. Επειδή υπάρχουν πολλές επιλογές για το configuration του Linux kernel, σας δίνουμε εμείς έτοιμο ένα configuration που είναι συμβατό με το Linux OS που έχουμε εγκαταστήσει στο disk image. Θα αντιγράψετε από την περιοχή του μαθήματος το configuration file ώστε να το χρησιμοποιήσετε για τον kernel που θα φτιάξετε.

Το μόνο που πρέπει να αλλάξετε στο .config αρχείο είναι η παράμετρος CONFIG_LOCALVERSION. Εκεί πρέπει να προσθέσετε μια κατάληξη για το όνομα (version) του καινούργιου kernel που θα φτιάξετε, έτσι ώστε να είστε σίγουροι ότι χρησιμοποιείτε τον δικό σας kernel όταν θα τον φορτώσετε με τον QEMU (και όχι τον original kernel). Επίσης, αν επαναλάβετε την διαδικασία περισσότερες φορές, θα μπορείτε να ξεχωρίζετε τα διαφορετικά revisions του kernel που έχετε δοκιμάσει, αλλάζοντας το kernel version πριν από κάθε μεταγλώττιση. Οπότε στο CONFIG_LOCALVERSION θα πρέπει να βάλετε το username σας, και αν θέλετε και ένα revision number. Το version του kernel θα μπορείτε να το δείτε όταν ξεκινάει το OS, ή αφού έχετε κάνει login με την εντολή:

```
$ uname -a
```

Ο gcc που υπάρχει στα μηχανήματα της σχολής είναι έκδοσης 10.2.1. Όμως με αυτό το version, δεν θα καταφέρουμε να κάνουμε compile επιτυχώς τον Linux kernel. Γι΄αυτό, έχουμε προσθέσει ένα gcc compiler παλαιότερης έκδοσης στην περιοχή του μαθήματος, τον οποίο θα πρέπει να χρησιμοποιήσετε σε αυτή την άσκηση. Οπότε, πριν κάνετε compile τον kernel θα εκτελέσετε τις εντολές:

```
$ export PATH="/home/misc/courses/hy345/gcc-4.9.2-standalone/bin/:$PATH"
$ export
PATH="/home/misc/courses/hy345/gcc-4.9.2-standalone/libexec/gcc/x86_64-unk
nown-linux-gnu/4.9.2/:$PATH"
```

Έτσι, λέτε στο σύστημα να κοιτάξει στο συγκεκριμένο path για να βρει τον compiler, επομένως θα βρει πρώτα την έκδοση που είναι εγκατεστημένη στη περιοχή του μαθήματος. Για να βεβαιωθείτε ότι χρησιμοποιείτε το σωστό gcc (4.9.2), αρκεί να τρέξετε:

$ which gcc ή **$ gcc --version**

Τέλος, για να γίνει compile ο kernel με τις δικές σας αλλαγές, θα τρέξετε:

$ make ARCH=i386 bzImage

Το καινούργιο kernel image (bzImage) που θα δημιουργηθεί από τη μεταγλώττιση, θα είναι το αρχείο **linux2.6.38.1/arch/x86/boot/bzImage**. Αφού τον καινούργιο kernel δεν θα τον χρησιμοποιήσετε στο host OS, δεν χρειάζεται να κάνετε `make install`.

Συνοπτικά τα βήματα που πρέπει να ακολουθήσετε είναι:

```
$ cp ~hy345/qemu-linux/linux-2.6.38.1-patched.tar.bz2 /spare/[username]/
$ cd /spare/[username]
$ tar -jxvf linux-2.6.38.1-patched.tar.bz2
$ cd linux-2.6.38.1

# Edit kernel source code to implement the new system calls

$ cp ~hy345/qemu-linux/.config . # Mind the dot at the end!

# Edit .config, find CONFIG_LOCALVERSION="-hy345", and append to the kernel's
version name your username and a revision number

$ export PATH="/home/misc/courses/hy345/gcc-4.9.2-standalone/bin/:$PATH"
$ export
PATH="/home/misc/courses/hy345/gcc-4.9.2-standalone/libexec/gcc/x86_64-u
nknown-linux-gnu/4.9.2/:$PATH"
$ make ARCH=i386 bzImage
```

Ρύθμιση παραμέτρων QEMU

Το επόμενο βήμα είναι να χρησιμοποιήσετε το καινούργιο kernel image, το οποίο βρίσκεται στο αρχείο linux2.6.38.1/arch/x86/boot/bzImage για να ξεκινήσετε το Linux χρησιμοποιώντας το QEMU. Θα χρησιμοποιήσετε ξανά το virtual disk image που σας δώσαμε, αλλά θα δώσετε επίσης στο QEMU το kernel image που θα τρέξει.

$ qemu-system-i386 -hda hy345-linux.img -append "root=/dev/hda" -kernel linux-2.6.38.1/arch/x86/boot/bzImage

Με το `-kernel linux-2.6.38.1/arch/x86/boot/bzImage` το QEMU θα ξεκινήσει με το καινούργιο kernel image. Με το `-append "root=/dev/hda"` το QEMU θα κάνει mount

το root filesystem από το /dev/hda, που είναι το disk image που φορτώνετε όπως και πριν. Αφού κάνετε login, με την εντολή `uname -a` βλέπετε την έκδοση του kernel που τρέχει το σύστημα.

Remote working

Αν δουλεύετε remotely σε κάποιο μηχάνημα του τμήματος, για να ξεκινήσετε το QEMU στο remote μηχάνημα θα πρέπει να συνδεθείτε με X11 forwarding από τον δικό σας υπολογιστή. Το X11 forwarding μπορεί να ενεργοποιηθεί όταν συνδέεστε στο μηχάνημα:

```
$ ssh [username]@[host].csd.uoc.gr -Y
```

Δοκιμάστε να τρέξετε την εντολή xterm και θα πρέπει να σας ανοίξει το ένα παράθυρο xterm. Εφόσον λειτουργεί το xterm, μπορείτε να χρησιμοποιήσετε το QEMU. Εναλλακτικά, μπορείτε απλά να τρέχετε το QEMU χωρίς γραφικό περιβάλλον (πολύ λιγότερο lag) με την βιβλιοθήκη ncurses:

```
$ qemu-system-i386 -hda hy345-linux.img –curses
```

Εναλλακτικά μπορείτε να αντιγράψετε το kernel image (αφού έχετε αλλάξει και έχετε κάνει compile τον Linux kernel σε κάποιο μηχάνημα του τμήματος) και το disk image, και έπειτα να τρέξετε το QEMU (αφού το εγκαταστήσετε) τοπικά στον υπολογιστή σας. Εάν χρησιμοποιείτε putty πρέπει να ενεργοποιήσετε την επιλογή “Enable X11 forwarding” στην καρτέλα Connection και μετά SSH.

Μεταφορά αρχείων από το Guest OS στο Host OS

Για να μεταφέρετε αρχεία από το guest OS (που τρέχετε με το QEMU) στο host OS (που κάνετε την βασική σας υλοποίηση) και αντίστροφα, μπορείτε να χρησιμοποιήσετε το πρόγραμμα scp. Μέσα από το guest OS μπορείτε να προσπελάσετε το host OS με την (virtual) IP address 10.0.2.2. Για παράδειγμα, για να μεταφέρετε το αρχείο test1.c από το guest OS στο host OS στην περιοχή σας σε έναν κατάλογο hy345 μπορείτε να εκτελέσετε μέσα από το QEMU (δηλαδή από το Guest OS) την εντολή:

```
$ scp test1.c [username]@10.0.2.2:~/hy345
```

Το [username] είναι το username που έχετε στα μηχανήματα του τμήματος. Θα χρειαστεί να δώσετε το password που έχετε στα μηχανήματα του τμήματος για να ολοκληρωθεί η αντιγραφή με το scp. Αντίστοιχα, για να αντιγράψετε από το host OS (π.χ. ένα μηχάνημα του τμήματος) το αρχείο test1.c από τον κατάλογο hy345 που είναι στην περιοχή σας στο Linux OS που τρέχει στο QEMU, θα τρέξετε μέσα από το QEMU την εντολή:

```
$ scp [username]@10.0.2.2:~/hy345/test1.c .
```

Προσοχή στην τελεία μετά το test1.c. Δεν είναι τυπογραφικό λάθος.

Προσθήκη νέου system call

Γενικές πληροφορίες για τα βήματα που πρέπει να ακολουθήσετε και τα αρχεία που πρέπει να αλλάξετε ή να δημιουργήσετε για να προσθέσετε ένα system call στο Linux kernel 2.6 μπορείτε να βρείτε εδώ. Ο παρακάτω οδηγός περιγράφει συνοπτικά πως είναι δομημένο ένα system call στο Linux kernel και πως μπορείτε να προσθέσετε ένα νέο. Υπάρχουν τρία βασικά βήματα για να υλοποιήσετε ένα καινούργιο system call στον Linux kernel:

1. Να προσθέσετε ένα καινούργιο system call number στον πυρήνα που να αντιστοιχεί στο δικό σας system call.

2. Να προσθέσετε ένα entry στο system call table του kernel με το system call number του δικού σας system call. Αυτό το entry θα καθορίσει ποια συνάρτηση του πυρήνα θα εκτελεστεί όταν συμβεί ένα trap με το δικό σας system call number (όταν δηλαδή καλεστεί το system call από user level και ο έλεγχος μεταβεί στον kernel για την εκτέλεση του συγκεκριμένου system call).

3. Πρέπει να προσθέσετε κώδικα στον kernel που να υλοποιεί την λειτουργικότητα που θα προσφέρει το system call. Πρέπει επίσης να προσθέσετε τα κατάλληλα header files, για να ορίσετε καινούργιους τύπους και structs που χρησιμοποιεί το system call για να μεταφέρει πληροφορία μεταξύ kernel και user space. Ακόμα, θα πρέπει να αντιγράφετε arguments και αποτελέσματα μεταξύ kernel space και user space χρησιμοποιώντας τις αντίστοιχες συναρτήσεις που υπάρχουν στον kernel.

Παράδειγμα:

Έστω ότι θέλουμε να προσθέσουμε ένα system call με όνομα **dummy_sys** το οποίο παίρνει ένα όρισμα από το user-level πρόγραμμα που το κάλεσε και συγκεκριμένα έναν ακέραιο αριθμό. Το dummy_sys system call θα τυπώνει απλά αυτόν τον αριθμό που δόθηκε σαν όρισμα και θα επιστρέφει το διπλάσιο του στο user-level πρόγραμμα. Θα ακολουθήσουμε τα παρακάτω βήματα:

1. Ανοίγουμε το αρχείο **linux-2.6.38.1/arch/x86/include/asm/unistd_32.h** με κάποιον editor, βρίσκουμε τα system call numbers για τα system calls που υπάρχουν ήδη στον kernel, και μετά το τελευταίο system call number (π.χ. 340 στον δικό μας kernel) προσθέτουμε μία γραμμή με τον επόμενο αριθμό.

   ```
   #define __NR_dummy_sys 341
   ```

   Επίσης αυξάνουμε το NR_syscalls κατά ένα (π.χ. από 341 σε 342 στον δικό μας

kernel). Έτσι ορίσαμε τον αριθμό 341 για το system call dummy_sys. Αυτός ο αριθμός θα χρησιμοποιηθεί μετά από ένα trap ώστε να βρεί ο kernel στο system call table την κατάλληλη συνάρτηση (system call function pointer) που υλοποιεί το system call.

2. Ανοίγουμε το αρχείο **linux-2.6.38.1/arch/x86/kernel/syscall_table_32.S** και εκεί προσθέτουμε στην τελευταία γραμμή το όνομα της συνάρτησης που υλοποιεί το καινούργιο system call.

```
.long sys_dummy_sys /* 341 */
```

3. Στο τρίτο βήμα θα ορίσουμε το system call dummy_sys. Στο τέλος του αρχείου **linux-2.6.38.1/include/asm-generic/syscalls.h** προσθέτουμε το function prototype του system call.

```
asmlinkage long sys_dummy_sys(int arg0);
```

4. Αν έχετε να προσθέσετε type definitions πρέπει να φτιάξετε και ένα header file στον φάκελο **linux- 2.6.38.1/include/linux/** το οποίο θα κάνετε include όπου χρειάζεται. Για το παράδειγμα αυτό κάτι τέτοιο δεν είναι αναγκαίο.

5. Έπειτα, φτιάχνουμε ένα καινούργιο αρχείο στον φάκελο **linux-2.6.38.1/kernel/** το οποίο θα περιέχει την υλοποίηση του system call. Στο παράδειγμα αυτό το αρχείο θα είναι το linux-2.6.38.1/kernel/dummy_sys.c και θα περιέχει τον κώδικα:

```
#include <linux/kernel.h>

asmlinkage long sys_dummy_sys(int arg0) {
    printk("Called dummy_sys with argument: %d\n", arg0);
    return ((long)arg0 * 2);
}
```

6. Αν έχετε arguments που περνάνε by reference από user space σε kernel space θα πρέπει να τα αντιγράψετε αφού καλέσετε το `access_ok()`. Η αντιγραφή μπορεί να γίνει καλώντας την συνάρτηση `copy_from_user()`. Αντίστοιχη διαδικασία θα πρέπει να κάνετε για να αντιγράψετε τα δεδομένα πίσω στο user space χρησιμοποιώντας τις συναρτήσεις `access_ok()` και `copy_to_user()`.

7. Τέλος, θα πρέπει να αλλάξετε το αρχείο **linux2.6.38.1/kernel/Makefile** για να συμπεριλάβει και να κάνει compile το καινούργιο source code αρχείο προσθέτοντας μια γραμμή στο κατάλληλο σημείο:

```
obj-y += dummy_sys.o
```

Παρατηρήσεις:

- Είναι σημαντικό να δείτε πως έχουν υλοποιηθεί κάποια παρόμοια system calls που υπάρχουν ήδη στον Linux kernel (π.χ. gettimeofday, times, getpid)

- Μπορείτε να μάθετε και να ακολουθήσετε το coding style του Linux Kernel κατά την υλοποίηση των system calls: https://www.kernel.org/doc/Documentation/process/coding-style.rst

- Το Elixir Cross Reference θα σας βοηθήσει να περιηγηθείτε στον source code του Linux kernel. Η αναζήτηση του (Search Identifier) πιθανώς να σας φανεί χρήσιμη για να εντοπίσετε συναρτήσεις και δομές στα διάφορα αρχεία του source code. Προσέξτε να έχετε επιλέξει την σωστή έκδοση του kernel που χρησιμοποιούμε στην άσκηση. https://elixir.bootlin.com/

- Για να τροποποιήσετε αρχεία μέσα από το Guest OS μπορείτε να χρησιμοποιήσετε τον editor Vi, ο οποίος λειτουργεί παρόμοια με τον editor Vim. Περισσότερες πληροφορίες για την χρήση του vi μπορείτε να βρείτε στο: http://linuxfocus.org/~guido/vi/viref.html

Δοκιμή system calls

Στο τελευταίο βήμα της άσκησης θα πρέπει να δοκιμάσετε τα καινούργια system calls. Αφού έχετε κάνει compile με επιτυχία τον kernel με τα system calls που φτιάξατε, και έχετε ξεκινήσει τον QEMU με τον καινούργιο Linux kernel, θα πρέπει να γράψετε κάποια δοκιμαστικά προγράμματα που να χρησιμοποιούν τα `set_task_params` και `get_task_params` στο guest Linux OS.

Μην ξεχνάτε ότι οι δηλώσεις των system calls που θα κάνετε βρίσκονται στον πυρήνα και δεν είναι ορατές σε user-level προγράμματα. Συνήθως, ένα system call καλείται μέσω κάποιας συνάρτησης που τρέχει σε user level και υπάρχει υλοποιημένη σε κάποια βιβλιοθήκη (π.χ. libc). Στην συνέχεια, αυτή η user-level συνάρτηση καλεί το macro `syscall()` με το σωστό system call number του, για να μεταβιβάσει τον έλεγχο στον πυρήνα (trap). Εκεί θα τρέξει ο κώδικας του system call. Αν δεν έχει υλοποιηθεί αυτή η συνάρτηση σε κάποια user-level library (όπως στην δική σας περίπτωση), ένα πρόγραμμα μπορεί να καλέσει το system call που θέλει χρησιμοποιώντας το macro `syscall()` αλλά και τον αριθμό του system call που θέλει να καλέσει. Για παράδειγμα, το παρακάτω κομμάτι κώδικα καλεί το system call με αριθμό 341 και του δίνει σαν όρισμα την τιμή 42.

```
#include <stdio.h>
#include <unistd.h>
#include <errno.h>

#define __NR_dummy_sys 341
```

```
int main(void) {
    printf("Trap to kernel level\n");
    syscall(__NR_dummy_sys, 42);
    printf("Back to user level\n");
    return 0;
}
```

Για την άσκηση αυτή πρέπει να φροντίσετε τα system calls να μοιάζουν με function calls. Μπορείτε να το καταφέρετε αυτό είτε δηλώνοντας macros είτε φτιάχνοντας τα κατάλληλα wrapper functions τα οποία θα δέχονται τα ίδια ορίσματα με τα system calls. Για παράδειγμα:

```
#define __NR_dummy_sys 341

/* Use either a macro */
#define dummy_sys(arg1) syscall(__NR_dummy_sys, arg1)

/* Or a wrapper function */
long dummy_sys(int arg1) {
    return syscall(__NR_dummy_sys, arg1);
}
```

Έτσι, στο δοκιμαστικό σας πρόγραμμα θα μπορείτε απλά να γράψετε `dummy_sys(42)` και να κληθεί το σωστό system call με τα σωστά arguments.

Σημείωση:

Αν έχετε header files με ορισμούς για νέους τύπους και structs, πρέπει να γίνουν κι αυτά include στα demo προγράμματα που θα γράψετε. Για να γίνει αυτό θα πρέπει να τα μεταφέρετε στο Guest OS και ίσως χρειαστεί να δώσετε το path με τα header files αυτά κατά τη διάρκεια της μεταγλώττισης. Συγκεκριμένα, θα χρειαστεί να ορίσετε το `struct d_params` ώστε να είναι ορατό σε user level. Για τον λόγο αυτό, μπορείτε να προσθέσετε τον ορισμό για το struct αυτό μαζί με τα macros ή wrapper functions των system calls. Όλοι οι ορισμοί αυτοί μπορούν να τοποθετηθούν στο αρχείο /usr/include/unistd.h που υπάρχει στο Guest OS και το οποίο θα κάνετε include σε όλα σας τα δοκιμαστικά προγράμματα.

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 Mekefile 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 αναφέρουμε στην εκφώνηση της άσκησης σε οποιοδήποτε βήμα μπορείτε επίσης να το αναφέρετε στο README.

5. Μπορείτε να φτιάξετε έναν φάκελο με τα τροποποιημένα source code αρχεία του kernel (αν θέλετε θα είναι καλό να κρατήσετε την δομή καταλόγων που υπάρχει στον linux kernel), έναν φάκελο με τα test προγράμματα και header files από το guest OS, και τέλος να τους μεταφέρετε σε ένα φάκελο 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 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.

Παρατηρήσεις

1. Η άσκηση είναι ατομική. Τυχόν αντιγραφές μπορούν να ανιχνευθούν εύκολα από κατάλληλο λογισμικό 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 των system calls.

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 απομένουν μόνο τα αρχεία που χρειάζονται για τη μεταγλώττιση.

4. Τοποθετήστε σε ένα φάκελο όλα τα αρχεία προς παράδοση για την άσκηση 3. Παραδώστε τα παραπάνω αρχεία χρησιμοποιώντας το πρόγραμμα turnin και συγκεκριμένα εκτελώντας `turnin assignment_3@hy345 directory_name` από τον κατάλογο που περιέχει τον κατάλογο directory_name.

5. Σε πολλές περιπτώσεις τα ονόματα των αρχείων είναι ενδεικτικά. Μπορείτε να χρησιμοποιήσετε όποια ονόματα σας βολεύουν.