Φροντιστήριο: 4/11/2016
Παράδοση: 14/11/2016

Implementation of system calls for “Least Remaining Time First” scheduling policy in the Linux Operating System

Ο “least-remaining-time-first” είναι ένας αλγόριθμος χρονοπρογραμματισμού διεργασιών. Κάθε διεργασία δηλώνει τον χρόνο που χρειάζεται για να ολοκληρώσει την εκτέλεσή της. Αν μια διεργασία δεν δηλώσει τον χρόνο που χρειάζεται, θεωρούμε πως είναι άπειρος. Σε κάθε scheduling interval ο kernel επιλέγει προς εκτέλεση την διεργασία με τον “μικρότερο εναπομείναντα χρόνο” (least-remaining-time), την εκτελεί για ένα κβάντο και ανανεώνει τις απαραίτητες πληροφορίες που διατηρεί για κάθε διεργασία. Αν ο “μικρότερος εναπομείναντας χρόνος” για κάποια διεργασία γίνει αρνητικός (η διεργασία εκτελέστηκε για περισσότερο χρόνο απ’ όσο είχε αρχικά δηλώσει) ο kernel την τιμωρεί ορίζοντάς τον σε άπειρο.

Σε αυτήν την άσκηση θα πρέπει να υλοποιήσετε δύο καινούρια system calls, τα οποία θα χρησιμοποιήσετε και στην επόμενη άσκηση του μαθήματος (**4η άσκηση**). Πιο συγκεκριμένα θα χρειαστεί να προσθέσετε στον πυρήνα (kernel) του λειτουργικού συστήματος Linux τα δύο νέα system calls: “**set_total_computation_time**” και “**get_remaining_time**”. Έπειτα θα τα δοκιμάσετε χρησιμοποιώντας τον προσομοιωτή (emulator) QEMU (δείτε αναλυτικές οδηγίες στην παράγραφο **Β**. σελ.3) κάνοντας compile τον Linux kernel με τις αλλαγές σας, θα τρέξετε το Linux με τον καινούργιο kernel στον προσομοιωτή QEMU και, τέλος, εκεί θα γράψετε και θα τρέξετε ένα user-level demo πρόγραμμα που θα χρησιμοποιεί τα καινούργια system calls.

Στόχος της άσκησης είναι να εξοικειωθείτε με τον source code του Linux kernel, με τον τρόπο που είναι δομημένο ένα system call στο Linux, με την μεταγλώττιση του kernel, και με τη χρήση ενός προσομοιωτή.

Α. Προσθήκη νέων system calls στον Linux kernel

Η αλλαγή που θα κάνετε στον Linux kernel 2.6.38.1 είναι να προσθέσετε δύο καινούργια system calls (δείτε αναλυτικές οδηγίες στην παράγραφο **Γ**. σελ.6) που θα λέγονται “**set_total_computation_time**” και “**get_remaining_time**”. Επίσης, θα πρέπει να προσθέσετε στη δομή task_struct, η οποία αναπαριστά μια διεργασία στον Linux kernel, τρία καινούργια πεδία:

```
unsigned int   total_computation_time;   /* Ο χρόνος που δήλωσε η διεργασία */
int            remaining_time;           /* Ο χρόνος που απομένει */
unsigned int   infinite;                 /* Ένα flag για την ποινή του απείρου */
```

Η τιμή στο πρώτο πεδίο θα αλλάζει από το system call “**set_total_computation_time**”. Το system call “**get_remaining_time**” θα επιστρέφει την τιμή της μεταβλητής remaining_time και infinite.

Συγκεκριμένα:

- set_total_computation_time(int pid, unsigned int total_time)

Δηλώνει στο σύστημα τον χρόνο που ζητά η διεργασία με το συγκεκριμένο pid για την ολοκλήρωσή της. Μόλις βρείτε τη σωστή διεργασία (τρέχουσα διεργασία αν το πρώτο όρισμα είναι -1 ή διεργασία με pid == pid), ο kernel θα πρέπει να θέτει στο πεδίο total_computation_time την αντίστοιχη τιμή από το δεύτερο όρισμα του system call. Μια διεργασία πρέπει να μπορεί να ορίσει **μόνο** τις δικές της παραμέτρους και των **θυγατρικών** διεργασιών της.

- get_remaining_time(int pid, struct t_params *t_arguments)

Επιστρέφει μέσω του pointer t_arguments τις τιμές των παραμέτρων remaining_time και infinite για την διεργασία με το συγκεκριμένο pid που δίδεται ως όρισμα. Για αυτό το struct θα πρέπει να δεσμεύει μνήμη το user-level πρόγραμμα πριν καλέσει το system call. Στη συνέχεια, ο kernel θα πρέπει να το γεμίζει με όλες τις πληροφορίες που χρειάζονται για την διεργασία που ζητήθηκε και θα επιστρέφει αυτές τις πληροφορίες (by reference) στο user-level πρόγραμμα που κάλεσε το system call. Αν αυτό το δεύτερο όρισμα είναι NULL, τότε το system call θα πρέπει να επιστρέφει το error value EINVAL. Όπως επίσης αν συμβεί οποιοδήποτε άλλο λάθος (π.χ. αν ο kernel δεν μπορεί να αντιγράψει τις ζητούμενες πληροφορίες σε user space), θα επιστρέφεται το ίδιο error value. Αν το system call τρέξει επιτυχώς, αυτό θα πρέπει να επιστρέφει 0. Αντίστοιχα, το user-level πρόγραμμα που χρησιμοποιεί το system call θα πρέπει να ελέγχει την επιστρεφόμενη τιμή για ενδεχόμενο λάθους. Αν το return value του system call είναι 0 (success), τότε τα πραγματικά αποτελέσματα θα επιστραφούν μέσω του t_arguments. Μια διεργασία πρέπει να μπορεί να επιστρέψει **μόνο** τις δικές της παραμέτρους και των **θυγατρικών** διεργασιών της.

Σημειώσεις:

1. Κατά την κλήση των συναρτήσεων αυτών αν το pid είναι -1, τότε μας ενδιαφέρει η τρέχουσα διεργασία που καλεί το system call. Αν έχει άλλη τιμή, τότε θα πρέπει να βρείτε τη διεργασία που έχει αυτό το pid και έπειτα να αλλάξετε το παραπάνω πεδίο σε αυτήν τη διεργασία. Αν η τιμή του pid είναι αρνητική (αλλά όχι -1), ή αν δεν αντιστοιχεί σε κάποια τρέχουσα διεργασία, τότε το system call θα πρέπει επιστρέφει το error value EINVAL. Το EINVAL και τα υπόλοιπα error values είναι ορισμένα στο: linux-2.6.38.1/include/asm-generic/errno-base.h

2. Το pid_t ορίζετε στο: linux-2.6.38.1/include/linux/types.h

3. Για να βρείτε την τρέχουσα διεργασία κοιτάξτε στο αρχείο:
   linux-.6.38.1/arch/x86/include/asm/current.h
   Εκεί υπάρχει το inline function current που επιστρέφει το task_struct entry της τρέχουσας διεργασίας.

4. Κάθε φορά που καλούνται τα system calls **set_total_computation_time** και **get_remaining_time** στο επίπεδο του πυρήνα, θα πρέπει να τυπώνετε με την συνάρτηση printk μια γραμμή με το όνομα σας, τον Α.Μ. σας, και το όνομα του system call. Τα μηνύματα που τυπώνετε στον kernel με την συνάρτηση printk μπορείτε να τα βλέπετε όταν έχετε φορτώσει το Linux με το συγκεκριμένο kernel τρέχοντας το dmesg ή κάνοντας cat /var/log/messages. Έτσι, κάθε φορά που καλούνται τα συγκεκριμένα system calls από ένα user-level πρόγραμμα, θα πρέπει με αυτόν τον τρόπο να βλέπουμε το μήνυμα που εκτυπώσατε με το όνομά σας από τον kernel. Με τον ίδιο τρόπο (printk) μπορείτε να εκτυπώνετε ότι άλλα μηνύματα θέλετε από τον kernel (π.χ.

ένας απλός τρόπος να κάνετε debugging το system call που φτιάχνετε) και να τα βλέπετε όταν τρέχει ο kernel, αφού καλέσετε το system call, από το dmesg.

5. Ο Linux kernel αποθηκεύει τις αναλυτικές πληροφορίες για όλες τις τρέχουσες διεργασίες σε μία doubly linked list από task_struct structs. Μπορείτε να χρησιμοποιήσετε το macro for_each_process() για να προσπελάσετε όλες τις διεργασίες του συστήματος (task_struct) που είναι σε αυτήν τη λίστα, μια προς μια.
   Το struct task_struct ορίζεται στο αρχείο:
   linux-2.6.38.1/ include/linux/sched.h
   Εκεί μπορείτε να βρείτε όλες τις πληροφορίες για κάθε διεργασία: το pid, το όνομα της διεργασίας, το χρόνο που ξεκίνησε η εκτέλεση της, και τα user, system time κάθε διεργασίας.

6. Θα πρέπει να ορίσετε το struct t_params αρχείο:
   linux-2.6.38.1/include/linux/t_params.h που θα δημιουργήσετε. Όπως είπαμε, αυτό το struct θα περιέχει τις πληροφορίες που χρειαζόμαστε για τη διεργασία με το pid που δίνετε στο πρώτο όρισμα του system call **get_remaining_time**. Αυτές οι πληροφορίες θα είναι οι τιμές για τα πεδία remaining_time και infinite που ισχύουν για αυτή τη διεργασία.
   Αναλυτικά, το struct t_params θα πρέπει να έχει τα παρακάτω πεδία:

```
struct t_params {
        int             remaining_time;
        unsigned int    infinite
};
```

7. Αν το system call **set_total_computation_time** εκτελεστεί επιτυχώς, θα πρέπει να επιστρέφει 0. Αν όχι, τότε το system call θα πρέπει να επιστρέφει με το error value EINVAL. Το user-level πρόγραμμα που χρησιμοποιεί το system call θα πρέπει να ελέγχει την επιστρεφόμενη τιμή για ενδεχόμενο λάθους.

B. Running Linux on QEMU Emulator

Οι emulators είναι πολύ διαδεδομένοι για πολλούς λόγους. Για παράδειγμα, 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 (π.χ. /spare/[username]) που χρησιμοποιείτε, 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 [username] ώστε να έχετε μόνο εσείς read/write access:

```
$ 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. Περιέχει επίσης τον αρχικό Linux kernel 2.6.38.1 που χρησιμοποιεί για να ξεκινήσει το λειτουργικό σύστημα. 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 "csdhy345", όπως θα σας τυπώσει και το σύστημα. Επίσης μπορείτε να κάνετε login και με το account του "root" με password "hy345".

Μεταγλώττιση του αλλαγμένου 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 (~hy345/qemu-linux/linux-2.6.38.1.tar.bz2) στον κατάλογο που χρησιμοποιείτε στο /spare/[username] του μηχανήματος.

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 (~hy345/qemulinux/.config) ώστε να το

χρησιμοποιήσετε για τον kernel που θα φτιάξετε. Το μόνο option που πρέπει να αλλάξετε στο .config είναι το 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.

Τέλος, για να γίνει compile ο kernel με τις δικές σας αλλαγές, θα τρέξετε make ARCH=i386 bzImage. Το καινούργιο kernel image (bzImage) θα είναι το αρχείο: linux-2.6.38.1/arch/x86/boot/bzImage (Αφού τον καινούργιο kernel δεν θα τον χρησιμοποιήσετε στο host OS, δεν χρειάζεται να κάνετε make install). Παρακάτω περιγράφουμε συνοπτικά τα βήματα για να κάνετε compile τον kernel.

```
$ cp ~hy345/qemu-linux/linux-2.6.38.1.tar.bz2 /spare/[username]/
$ cd /spare/[username]
$ tar -jxvf linux-2.6.38.1.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
Edit .config, find CONFIG_LOCALVERSION="-hy345", and append to the kernel's
version name your username and a revision number
$ make ARCH=i386 bzImage
```

Εκτέλεση του αλλαγμένου Linux kernel με το QEMU

Το επόμενο βήμα είναι να χρησιμοποιήσετε το καινούργιο kernel image (linux-2.6.38.1/arch/x86/boot/ bzImage) με τις αλλαγές που κάνατε στον kernel για να ξεκινήσετε το Linux με 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 to 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.

X11 Forwarding - Για να δουλεύετε remotely με QEMU

Linux:

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. Αν χρησιμοποιείτε μηχάνημα Linux, όταν κάνετε enable από το gate1/gate2 θα σας επιστρέψει το command που χρειάζεται να εκτελέσετε για να συνδεθείτε στο , π.χ.

```
$ ssh username@gate1.csd.uoc.gr -p 17724 –Y
```

Δοκιμάστε να τρέξετε xterm, θα πρέπει να σας ανοίξει το 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
```

Είτε να αντιγράψετε το kernel image (αφού έχετε αλλάξει και έχετε κάνει compile τον Linux kernel σε κάποιο μ ηχάνημα του τμήματος) και το disk image, και έπειτα να τρέξετε το QEMU (αφού το εγκαταστήσετε) τοπικά στον υπολογιστή σας.

Windows:

Κατεβάστε το Xming, εγκαταστήστε το και τρέξτε το:
http://sourceforge.net/projects/xming/files/Xming/6.9.0.31/Xming-6-9-0-31-setup.exe/download

Στο putty πρέπει να κάνετε τις παρακάτω ρυθμίσεις:
Session -> Hostname: gate1.csd.uoc.gr
Session -> Port: το port που θα σας επιστρέψει το "enable", π.χ. 17724
Connection -> ssh -> X11: tick στο enable X11 forwarding.

Και μετά open στο putty για να συνδεθείτε. Αν έχετε firewall πρέπει να κάνετε unblock το Xming. Επίσης αν έχει lag δοκιμάστε να συνδεθείτε με VPN.

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 απλά να κάνετε:

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

μέσα από το QEMU (guest OS). 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 το «.» μετά το «test.c» **δεν** είναι τυπογραφικό λάθος !

Γ. Γενικές οδηγίες για την προσθήκη νέου system call στον Linux kernel

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 στον Linux kernel 2.6 μπορείτε να βρείτε εδώ . Ο παραπάνω οδηγός περιγράφει συνοπτικά πως είναι δομημένο ένα system call στον Linux kernel και πως μπορείτε να προσθέσετε ένα

καινούργιο. Υπάρχουν τρία βασικά βήματα για να υλοποιήσετε ένα καινούργιο system call στον Linux kernel:

1. Να προσθέσετε ένα καινούργιο system call number στον kernel για το δικό σας system call.

2. Να προσθέσετε ένα entry στον system call table του kernel με το system call number του δικού σας καινούργιου system call. Αυτό το entry θα καθορίσει ποια συνάρτηση του kernel 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, που ξεχωρίζει από το system call number).

3. Πρέπει να προσθέσετε κώδικα στον kernel που να υλοποιεί την λειτουργικότητα που θα προσφέρει το system call. Για αυτό 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 και user space με τις αντίστοιχες συναρτήσεις που υπάρχουν στον kernel.

Ένα απλό παράδειγμα:
Έστω ότι θέλουμε να προσθέσουμε ένα system call με όνομα dummy_sys το οποίο παίρνει ένα όρισμα από το user-level πρόγραμμα που τον κάλεσε: έναν ακέραιο αριθμό. Το dummy_sys system call θα τυπώνει απλά αυτόν τον αριθμό που δόθηκε σαν όρισμα και θα επιστρέφει τον διπλάσιο του στο userlevel πρόγραμμα.

1. Ανοίγουμε το 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, π.χ. 341 στον δικό μας kernel:

   ```
   #define __NR_dummy_sys 341
   ```

   Επίσης αυξάνουμε το NR_syscalls κατά ένα (π.χ. από 341 σε 342 στον δικό μας kernel). Έτσι ορίσαμε το system call number 341 για το system call dummy_sys. Αυτός ο αριθμός θα χρησιμοποιηθεί μετά από ένα trap ώστε να βρεί ο kernel στον system call table την κατάλληλη συνάρτηση (system call function pointer) που υλοποιεί το system call.

2. Στο δεύτερο βήμα ανοίγουμε το αρχείο:
   linux-2.6.38.1/arch/x86/kernel/syscall_table_32.S και προσθέτουμε στην τελευταία γραμμή το όνομα της συνάρτησης που υλοποιεί το καινούργιο system call (system call function pointer):

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

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

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

   Αν έχετε να προσθέσετε type definitions πρέπει να φτιάξετε και ένα header file στο linux- 2.6.38.1/include/linux/ το οποίο θα κάνετε include όπου χρειάζεται (δεν

χρειαζόμαστε για το dummy_sys, αλλά εσείς θα χρειαστείτε για τα δικά σας system calls). Έπειτα φτιάχνουμε ένα καινούργιο αρχείο dummy_sys.c στο linux-2.6.38.1/kernel, π.χ. το linux-source-2.6.38.1/kernel/dummy_sys.c το οποίο περιέχει τον κώδικα του system call:

```
#include <linux/kernel.h>
#include <linux/syscalls.h>
#include <asm/uaccess.h>

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

Αν έχετε arguments που περνάνε by reference από user space σε kernel space θα πρέπει να τα αντιγράψετε: αφού καλέσετε το access_ok(), θα τα αντιγράψετε καλώντας την συνάρτηση copy_from_user(). Αντίστοιχη διαδικασία θα κάνετε για να αντιγράψετε τα δεδομένα πίσω στο user space (access_ok και copy_to_user). Τέλος, θα πρέπει να αλλάξετε το linux-2.6.38.1/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
```

Σημείωση:

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

Hints:

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

2. Το Linux Cross Reference θα σας βοηθήσει να περιηγηθείτε στον source code του Linux Kernel. Το “Identifier Search” πιθανώς να σας φανεί χρήσιμο για να εντοπίσετε συναρτήσεις και δομές στα διάφορα αρχεία του source code.
   http://lxr.free-electrons.com/

3. Για να τροποποιήσετε αρχεία μέσα από το VM (guest OS) θα μπορείτε να χρησιμοποιήσετε τον editor Vi ο οποίος λειτουργεί παρόμοια με τον editor Vim. Εδώ θα βρείτε συγκεντρωμένες όλες τις απαραίτητες εντολές:
   http://www.lagmonster.org/docs/vi.html

Δ. Δοκιμή των νέων 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 test προγράμματα που να χρησιμοποιούν τα **set_total_computation_time** και **get_remaining_time** στο guest Linux OS. Συνήθως ένα system call καλείται μέσω κάποιας συνάρτησης που τρέχει σε user level και υπάρχει σε κάποια βιβλιοθήκη (π.χ. libc). Στην συνέχεια, αυτή η user- level συνάρτηση καλεί το macro syscall με το system call number του συγκεκριμένου system call για να μεταβιβάσει τον έλεγχο στον kernel (trap) και εκεί να τρέξει αυτό το system call. Αν δεν έχει υλοποιηθεί αυτή η συνάρτηση σε κάποια user-level library (θα πρέπει αυτό να γίνει μέσα στο guest Linux OS) ένα test προγράμμα μπορεί να τρέχει κατευθείαν το syscall macro με το system call number που έχει οριστεί για το system call στον αλλαγμένο kernel. Για παράδειγμα, το παρακάτω test πρόγραμμα καλεί το dummy_test system call με το system call number 341, δίνοντας σαν όρισμα τον αριθμό 42. Μπορείτε να κάνετε compile το test πρόγραμμα κανονικά με τον gcc.

```
#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);
    /* you should check return value for errors */
    printf("Back to user level\n");
}
```

Εσείς θα πρέπει, είτε να κάνετε define δύο macros για τα **set_total_computation_time** και **get_remaining_time**, ώστε τα syscalls να μοιάζουν με function calls, είτε να φτιάξετε δύο wrapper functions τα **set_total_computation_time** και **get_remaining_time** (με τα ορίσματα που δέχονται αυτά τα system calls) που να καλούν εσωτερικά το αντίστοιχο syscall. Έτσι θα καλείτε το **set_total_computation_time** η **get_remaining_time** macro ή wrapper function αντί για το syscall μέσα στο πρόγραμμά σας, σύμφωνα με το function prototype που ορίσαμε. Για παράδειγμα, για το dummy_sys:

```
/* either macro */
#define dummy_sys(arg1)     syscall(341, arg1)

/* or wrapper function */
long
dummy_sys(int arg1)
{
        syscall(341, arg1);
}
```

Στο demo πρόγραμμα καλούμε:

```
dummy_sys(42);
```

Demo program:

Θα φτιάξετε ένα πρόγραμμα το οποίο θα καλεί αρχικά το **set_total_computation_time** για την τρέχουσα διεργασία με κάποιες τιμές για το πεδίο total_computation_time και έπειτα θα καλεί το **get_remaining_time** και θα τυπώνει όλα τα πεδία για την τρέχουσα διεργασία. Έπειτα θα κάνετε την αντίστοιχη διαδικασία για μία θυγατρική διεργασία καθώς και λανθασμένες κλίσεις των system calls π.χ. **set_total_computation_time** με όρισμα pid διεργασίας η οποία δεν υπάρχει.

Τι πρέπει να παραδώσετε:

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 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 τα system calls (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 system call.

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 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 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_3@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.