# Λειτουργικά Συστήματα Δραστηριότητα 1

2024-2025

ΑΜ: 1096060

Ονοματεπώνυμο: Μαρία-Νίκη Ζωγράφου

**ΔΡΑΣΤΗΡΙΟΗΤΑ 1**

## Άσκηση 1:

Βρείτε το αρχείο στον πηγαίο κώδικα του xv6 όπου ορίζονται οι κλήσεις συστήματος οι οποίες θα είναι διαθέσιμες σε προγράμματα χρήση. Έπειτα, διαλέξτε ένα από τα προγράμματα χρήστη στον φάκελο user. Εξηγήστε τη λειτουργία του κάνοντας συγκεκριμένες αναφορές στα system calls που καλεί:

Στο αρχείο **user.h** ορίζονται όλα τα system calls τα οποία είναι διαθέσιμα στον χρήστη. Στα προγράμματα χρήστη γίνεται include user.h ώστε να έχει εκεί πρόσβαση ο χρήστης.

Ορισμός αυτών των συναρτήσεων γίνεται στο **usys.S**. Εκεί ο preprocessor της C αντικαθιστά το SYSCALL(name) με:

.globl name; \

name: \

movl $SYS\_ ## name, %eax; \

int $T\_SYSCALL; \int instruction causes the processor to generate a trap [1]

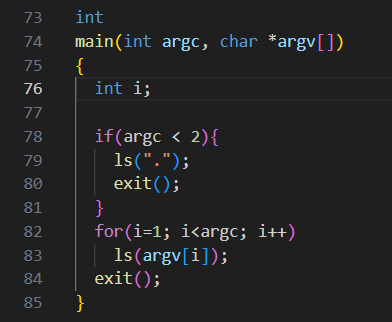
ret

Τοποθετεί δηλαδή το «όνομα» της συνάρτησης που καλεί ο χρήστης στον καταχωρητή eax, έτσι ώστε, όταν το int $T\_SYSCALL κάνει interrupt και ο έλεγχος μεταφερθεί στο Kernel, να κληθεί η κατάλληλη συνάρτηση.

Στην πραγματικότητα στον καταχωρητή eax τοποθετείται ένας αριθμός που αντιστοιχεί στο όνομα της συνάρτησης. Αυτή η αντιστοίχιση γίνεται στο αρχείο syscall.h με χρήση define.

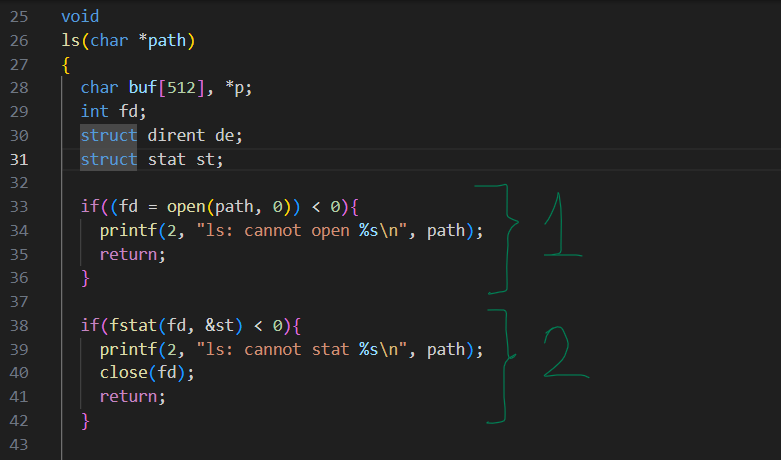
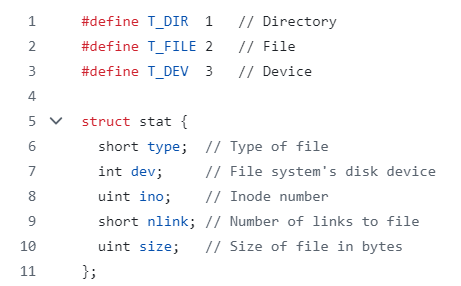
(Όσον αφορά το kernel, oι κλήσεις συστήματος γίνονται define στον φάκελο kernel στο αρχείο syscall.c όπου υπάρχει ένας πίνακας από function pointers για όλες τις συναρτήσεις που αποτελούν κλήσεις του συστήματος. Οι συναρτήσεις αυτές ορίζονται (γίνονται define) στα αρχεία sysfile.c,sysproc.c.)

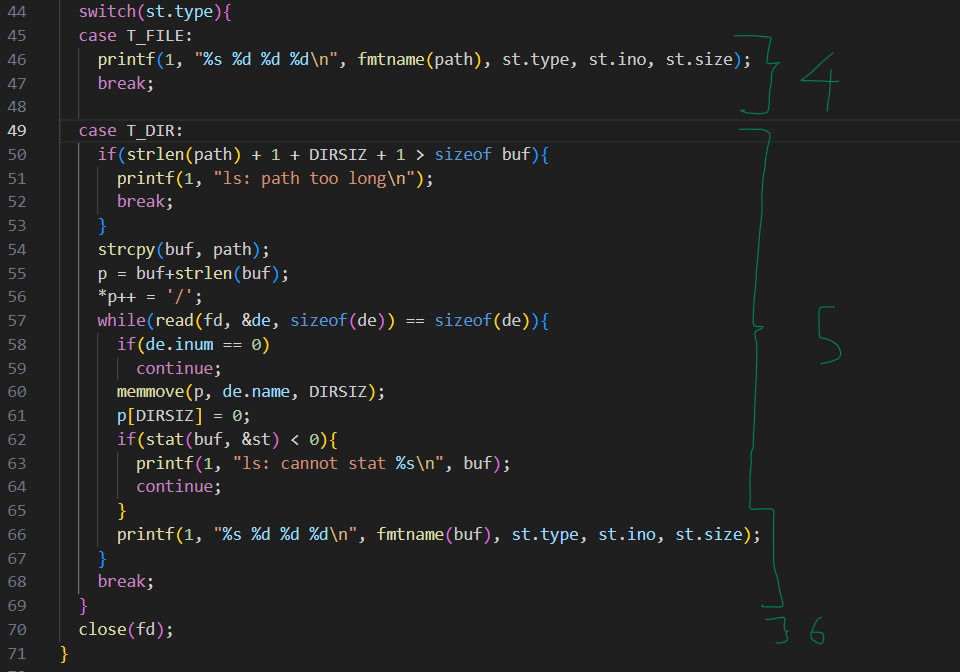
**Εξήγηση λειτουργίας του ls.c:**

Το ls.c τυπώνει τα αρχεία στο current path αν δεν δοθούν args αλλιώς τυπώνει τα αρχεία για κάθε Path που δόθηκε. (argc ο αριθμός των ορισμάτων και argv τα ορίσματα, αν είναι πολλά διαφορετικά Path το loop θα καλέσει την ls χωριστά για το καθένα).

Στο τέλος και των δύο περιπτώσεων καλείται η exit() system call. Αφού έχει γίνει #inlclude το user.h header (#include "user.h") o linker μπορεί να συνδέσει το αρχείο με το usys.S. H exit() τερματίζει το πρόγραμμα.

Κλήση της ls:

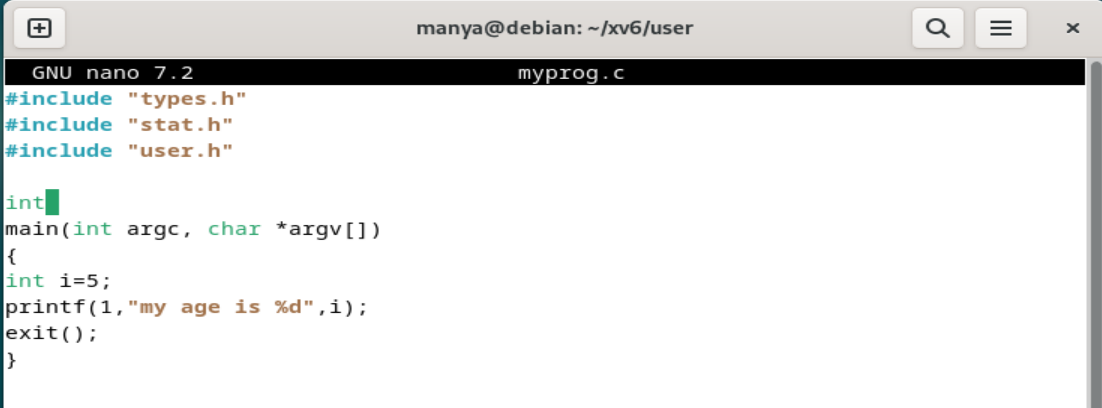
1. Όταν καλείται η συνάρτηση ls() γίνεται, έλεγχος αν το path που δόθηκε μπορεί να «ανοιχτεί», αν όχι τότε γίνεται return στην main.
2. Ύστερα γίνεται κλήση της fstat system call (με την προϋπόθεση ότι ο πρώτος έλεγχος ήταν επιτυχής και υπάρχει fd). Από το βιβλίο ([1] Russ Cox, Frans Kaashoek και Robert Morris. The Design of the xv6 x86 Operating System. 2019), γνωρίζουμε ότι η fstat επιστρέφει πληροφορίες για το αντικείμενο στο οποίο αναφέρεται το file descriptor. Άμα επιστραφεί τιμή μικρότερη του μηδενός τότε η ls τερματίζεται (δεν επιστρέφει τίποτα και σταματάει). Πριν γίνει το return όμως κάνουμε close(fd), system call που απελευθερώνει το file descriptor fd ώστε να μην ξαναχρησιμοποιηθεί.
3. Με ένα switch statement ελέγχουμε διαφορετικές περιπτώσεις. Στην ειδική δομή δεδομένων stat (έχει οριστεί στο αρχείο stat.h) έχει δοθεί από την fstat() (system call) τιμή που δείχνει αν πρόκειται για αρχείο, directory ή άλλη συσκευή. Η fstat καλείται με όρισμα το &st, δηλαδή με έναν pointer που δείχνει στην stat st.
4. Στην περίπτωση που πρόκειται για αρχείο, το πρόγραμμα τυπώνει το όνομα το αρχείου και κάποια χαρακτηριστικά του, που έχουν αποθηκευτεί από την fstat() στην stat st(st.type, st.ino, st.size). Για να τυπωθεί το όνομα του αρχείου χρησιμοποιείται η fmtname() η οποία απλά το επιστρέφει σαν string με συγκεκριμένο format.



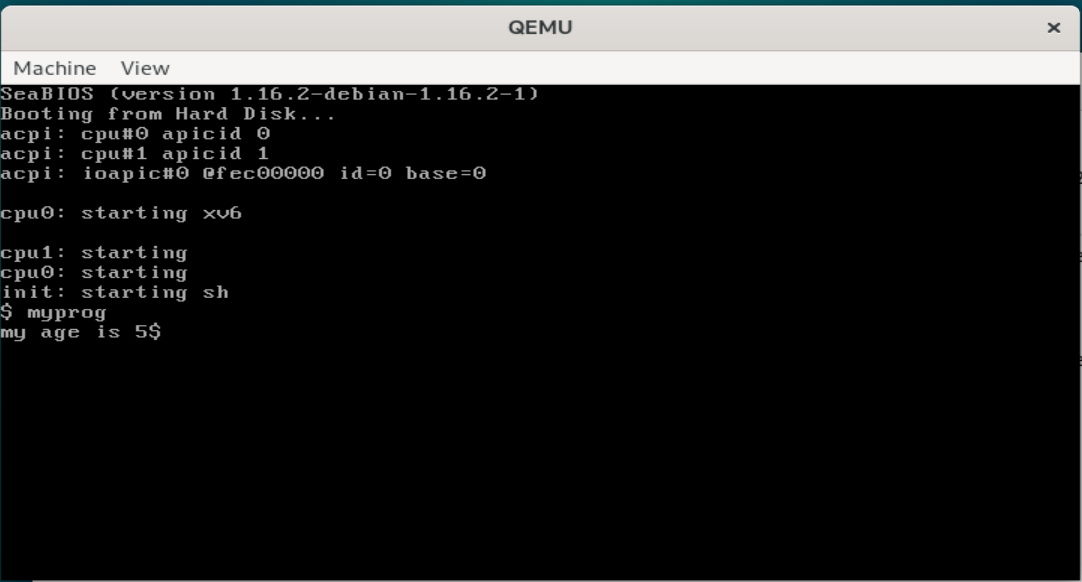
1. Στην περίπτωση που πρόκειται για directory εκτελούνται περισσότερες ενέργειες. Αρχικά αν το Path είναι πολύ μεγάλο η συνάρτηση σταματάει (Break). Αν όχι, το directory το μεταφέρουμε στο buffer buf και βάζουμε ‘/’ στο τέλος. Μετά με ένα loop διατρέχουμε τα περιεχόμενα. Καλούμε την read() system call. Όταν το directory δεν έχει άλλα περιεχόμενα για ανάγνωση, η read() θα επιστρέψει τιμή 0 και το loop θα σταματήσει. Αν το inum == 0 δεν έχει διαβαστεί σωστό αρχείο ή φάκελος οπότε πάμε παρακάτω. Αλλιώς προσθέτουμε το όνομα του αρχείου ή του φακέλου στο buffer που έχει το Path. Καλούμε την συνάρτηση stat() (από ulib.c) για την συγκεκριμένη εγγραφή(file or directory) στη συνέχεια. Αν επιστρέψει έγκυρη τιμή τυπώνουμε τα στατιστικά του αρχείου. Στην επομενη επανάληψη στο Buffer θα αντικατσταθει το παλιο αρχείο με το όνομα του νέου κ.οκ.
2. Tέλος με την sytem call close() απελευθερώνουμε το file descriptor για αποφυγή data leaks. (δεν ελέγχουμε όμως αν έγινε επιτυχώς αυτό).

## Άσκηση 2:

Δημιουργήστε ένα πρόγραμμα χρήστη για το xv6.



Εκτέλεση προγράμματος από το terminal του xv6:

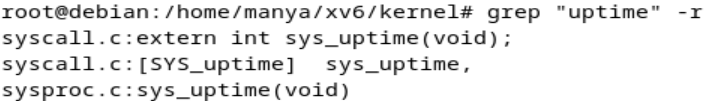


## Άσκηση 3:

Αναζητώντας μέσα στον πηγαίο κώδικα του xv6, βρείτε την κλήση συστήματος uptime.

1. Βρείτε όλα τα σημεία του κώδικα του πυρήνα στα οποία γίνεται αναφορά σε αυτή.

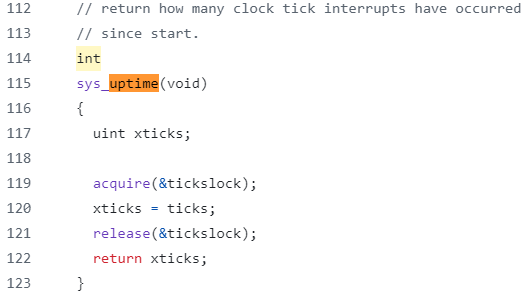
2. Εξηγήστε την υλοποίησή της:

Η συνάρτηση uptime ορίζεται στο αρχείο sysproc.c. Αναζητούμε που αλλού αναφέρεται μέσα στο kernel:

Η uptime υλοποιείται ως εξής:

Ορίζουμε unsigned int xticks.

Καλούμε την acquire() η οποία μας δίνει πρόσβαση στην μεταβλητή ticks και κλειδώνει το ticksclock για να μην μπορούν να την χρησιμοποιήσουν άλλες διεργασίες.

Ύστερα η xticks παίνρει την τιμή ticks. H ticks είναι μια global variable, της οποίας η τιμή αυξάνεται κάθε φορά που το ρολόι προκαλεί Interrupt. Έτσι δείχνει πόσα τικ έγιναν από όταν ξεκίνησε το σύστημα.

Με το release απελευθερώνουμε ξανά το ρολόι ticksclock και έχουν πρόσβαση σε αυτό οι άλλες διεργασίες. Η συνάρτηση acquire ορίζεται στο spinlock.c

## Άσκηση 4:

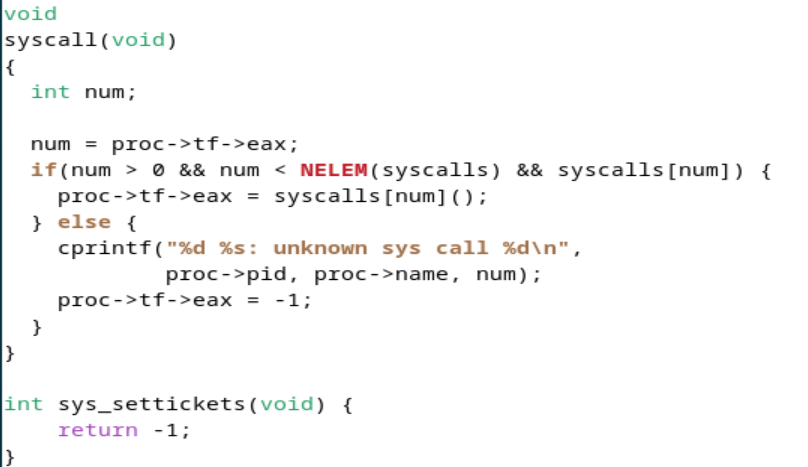
Βρείτε τη συνάρτηση μέσα στον kernel κώδικα όπου τελικά εκτελούνται οι system calls. Καταγράψτε τη σειρά των ενεργειών που γίνονται όταν καλείται ένα system call:

Περιγραφή με κείμενο:

Το πρόγραμμα καλεί κάποια system call. Αυτό είναι εφικτό επειδή τα user προγράμματα έχουν #inlcude “user.h”. Αφού το πρόγραμμα χρήστη γίνει compile, ο linker θα βρει τα implementations των συναρτήσεων στο usys.S. όπως εξηγήθηκε παραπάνω, στο αρχείο usys.S τοποθετείται στον καταχωρητή eax ο αριθμός της κλήσης συστήματος που ζητάμε(SYS\_name, όπου name η αντίστοιχη συνάρτηση). Αυτός ο αριθμός ορίζεται στο syscall.h.

Η εντολή int $T\_SYSCALL, προκαλεί διακοπή και ο έλεγχος περνάει από τον User στο kernel. Aν ο αριθμός του interrupt είναι έγκυρος καλείται η syscall() της οποίας ο κωδικός έχει τοποθετηθεί στον eax.

Η συνάρτηση syscall() ορίζεται στο syscall.c. Εκεί υπάρχουν τα declarations (ορισμοί) όλων των system calls στην μορφή extern int sys\_name(void). Το extern δηλώνει στον compiler ότι η συνάρτηση αυτή μπορεί να διαβαστεί και από άλλα αρχεία.

Στο ίδιο αρχείο υπάρχει ένας πίνακας από function pointers που αντιστοιχίζουν τους αριθμούς που τοποθετούνται στον eax με τα system calls. Με την εντολή syscall() ελέγχουμε αν είναι έγκυρη η τιμή του eax =num. Αν ναι, η εντολή syscalls[num]() καλεί την κατάλληλη συνάρτηση ορισμένη στα αρχεία sysporc.c και sysfile.c. Μετά την κλήση της system call, τo αποτέλεσμα της αποθηκεύεται στον eax.

Πίνακας αντιστοίχισης στο syscall.c: Κάθε SYS\_name που έχει τοποθετηθεί στον eax αντιστοιχίζεται με τον κατάλληλο function pointer. (lines 149-173)

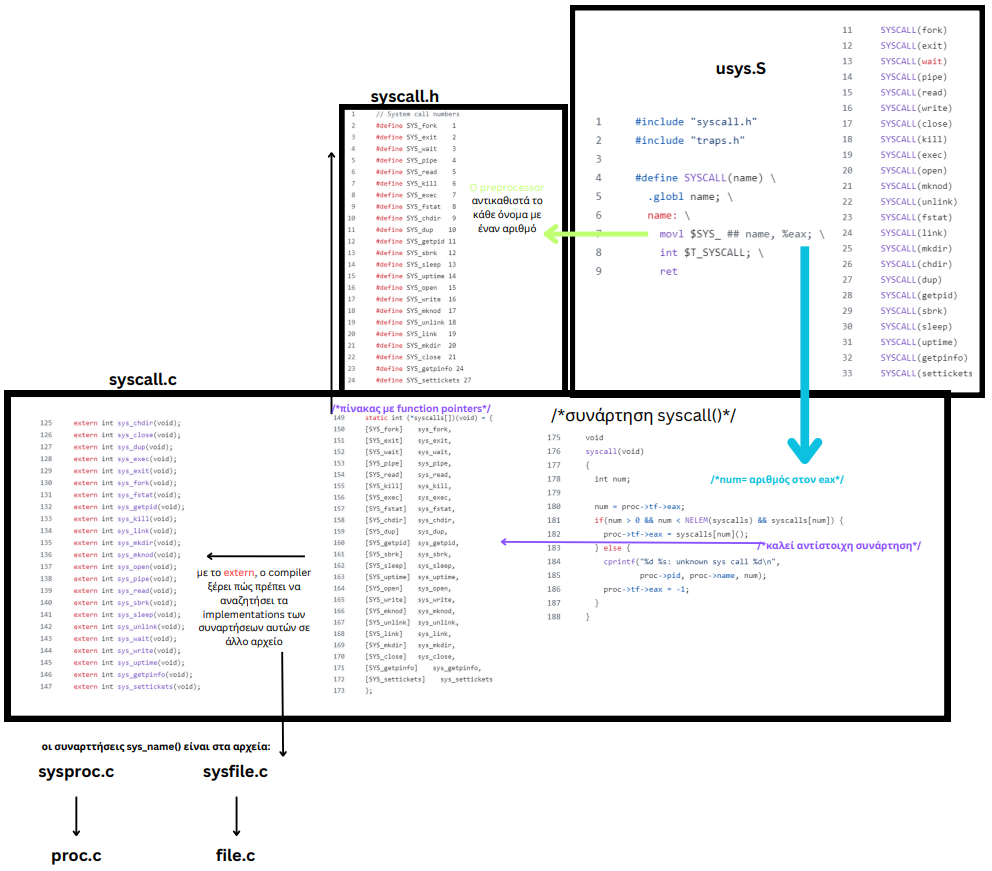
ΠΡΟΣΟΧΗ: Κάποιες συναρτήσεις στο sysproc.c καλούν συναρτήσεις από το proc.c.

Σχεδιάγραμμα σε μορφή εικόνας:

user.h

System call

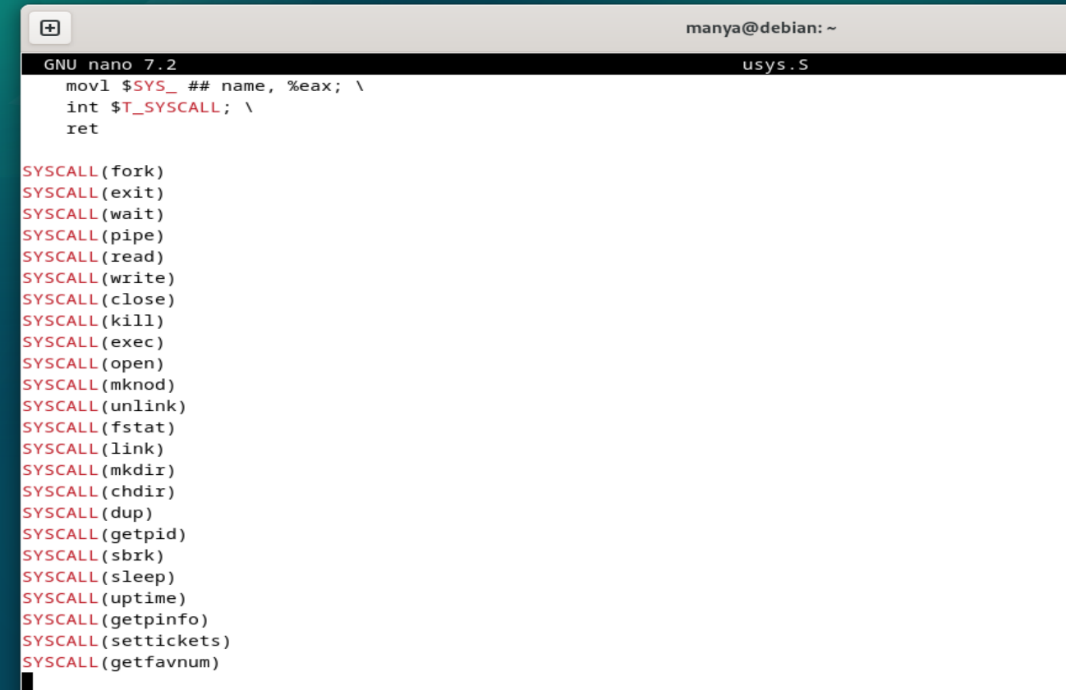
random\_user\_program.c

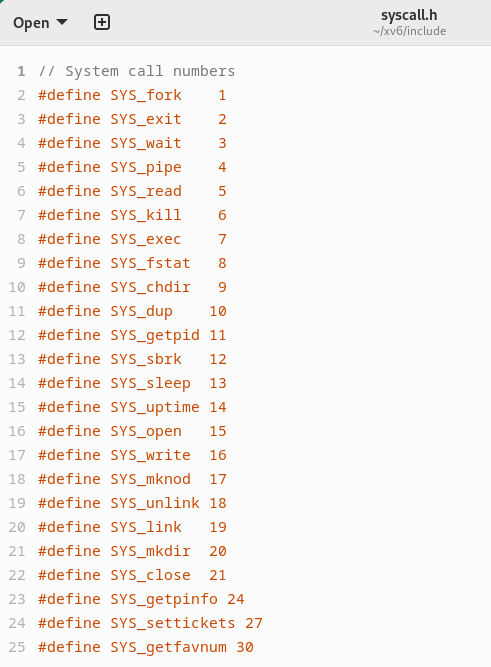


## Άσκηση 5:

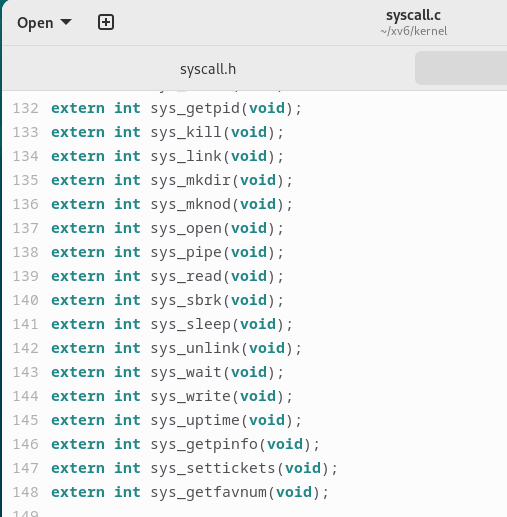
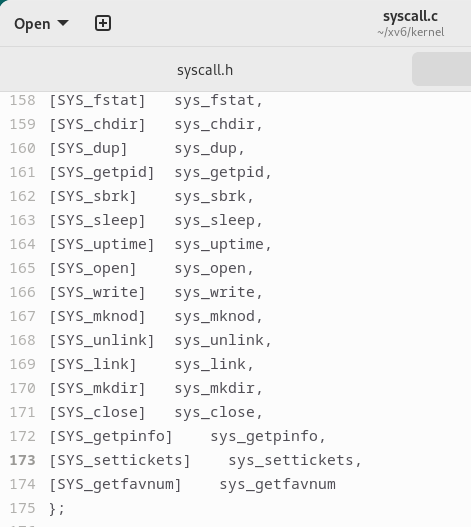
Προσθέστε μία κλήση συστήματος η οποία θα επιστρέφει τον αγαπημένο σας αριθμό. Η κλήση που θα φτιάξετε πρέπει να έχει το εξής πρωτότυπο: int getfavnum(void). Μία τέτοια κλήση δεν έχει κανέναν πρακτικό σκοπό αφού δεν εκμεταλλεύεται κάποια πληροφορία ή πόρο που κατέχει το λειτουργικό σύστημα. Στόχος εδώ είναι να κατανοήσουμε τον τρόπο με τον οποίο προστίθενται κλήσεις συστήματος στο xv6.

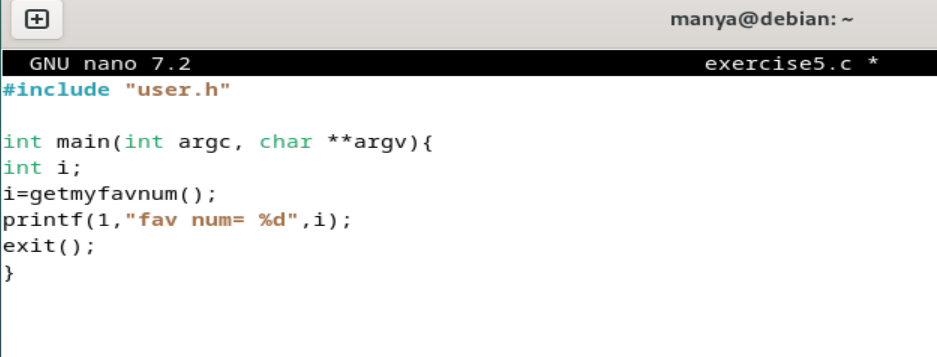
Βήμα 1: Προσθήκη στο user.h του ορισμού int getfavnum(void);

Βήμα 2: Προσθήκη SYSCALL(getfavnum) στο usys.S

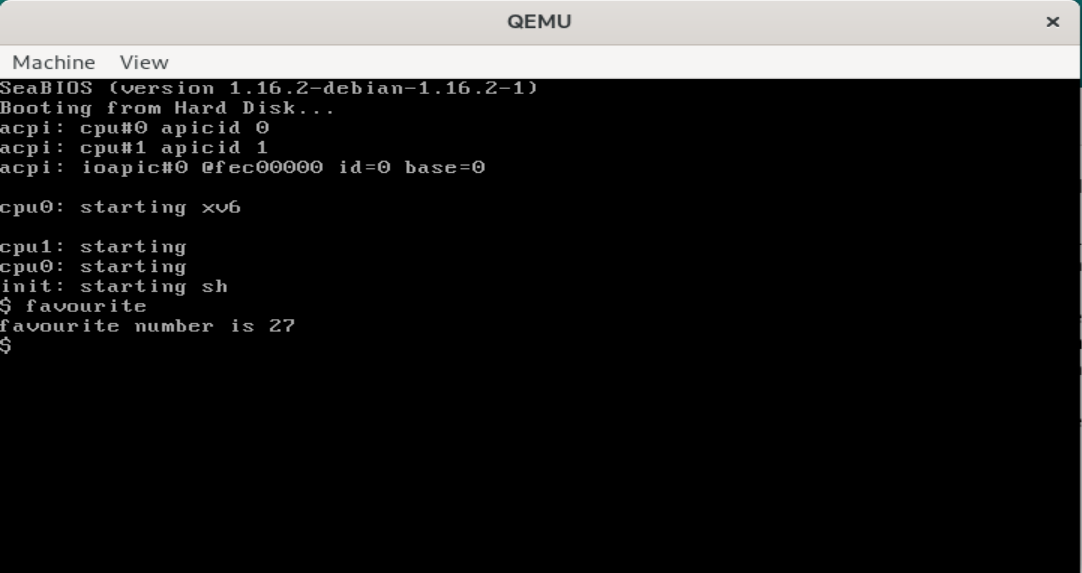
Βήμα 3: Αντιστοίχιση της νέας system call με αριθμό στο syscall.h header (π.χ. 30)

Βήμα 4: Προσθέτουμε στο syscall.c του kernel, το declaration extern int sys\_getfavnum(void); και ύστερα προσθέτουμε στον πίνακα από function pointers το ζεύγος [SYS\_getfavnum] sys\_getfavnum.



Βήμα 5: Υλοποίηση της getmyfavnum(). Είτε στο αρχείο syscall.c ή στο sysproc.c υλοποιούμε την συνάρτηση. Ένας απλός τρόπος ο εξής:

Βήμα 6: Φτιάχνουμε μια user εντολή που να ελέγχει την ορθή λειτουργία του system call. Το προσθέτουμε μετά στα προγράμματα που περιλαμβάνει το makefile και ύστερα καλούμε την εντολή αυτή από το terminal του xv6.



## Άσκηση 6:

Προσθέστε μία κλήση συστήματος η οποία θα εκτελεί λειτουργία «shutdown». Η κλήση που θα φτιάξετε πρέπει να έχει το εξής πρωτότυπο: void halt(void). Έπειτα, φτιάξτε ένα πρόγραμμα χρήστη με όνομα shutdown το οποίο θα καλεί την κλήση συστήματος που φτιάξατε, ώστε χρήστες του xv6 να μπορούν πλέον να κάνουν shutdown το σύστημα χωρίς να χρειάζεται να το κάνουν από το host μηχάνημα. Σε αυτή τη κλήση συστήματος θα πρέπει να επικοινωνήσετε άμεσα με το hardware της εικονικής μηχανής (στην περίπτωσή μας το QEMU). Για την απενεργοποίηση του συστήματος το hardware περιμένει σε κάποια συγκεκριμένη διεύθυνση (port) να υπάρξει μία συγκεκριμένη τιμή. Οι αριθμοί για αυτά τα δύο ορίζονται από τον κατασκευαστή. Οδηγίες για την τιμή που πρέπει να στείλετε σε ποια πόρτα έχουν καταγραφεί για διάφορους emulators εδώ. Εξηγήστε τη λειτουργία της συνάρτησης outw την οποία θα χρησιμοποιήσετε.

*Σημείωση: Η συνάρτηση outw είναι ήδη υλοποιημένη στο xv6 και βρίσκεται στο αρχείο include/x86.h.*

Πρέπει να κάνουμε κάποιες προσθήκες για την υλοποίηση της halt:

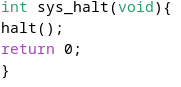
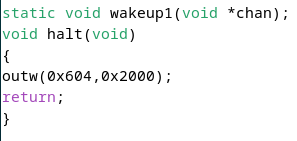
Βήμα 1: void halt(void) πρότυπο στο user.h

Βήμα 2: #define SYS\_halt 31 στο systemcall.h

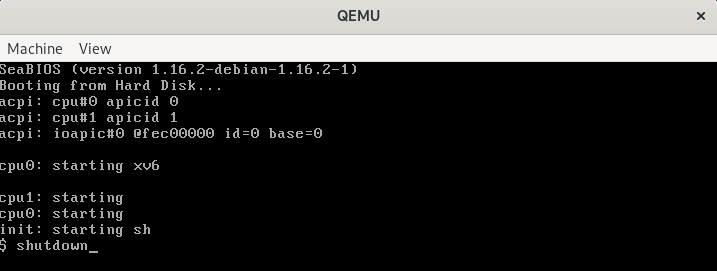
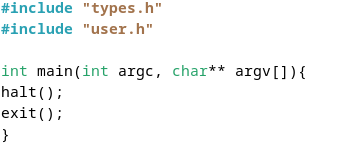
Βήμα 3: SYSCALL(halt) στο usys.S

Βήμα 4: Προσθέτουμε στο syscall.c του kernel, το declaration extern int sys\_halt(void); και ύστερα προσθέτουμε στον πίνακα από function pointers το ζεύγος [SYS\_halt] sys\_halt.

Βήμα 5: Υλοποίηση της sys\_halt. Στο αρχείο sysproc.c υλοποιούμε την int sys\_halt(void) η οποία καλεί την void halt(void) Που την υλοποιούμε στο proc.c

Για να γίνει shutdown o emulator χρειάζεται η outw(0x604, 0x2000);

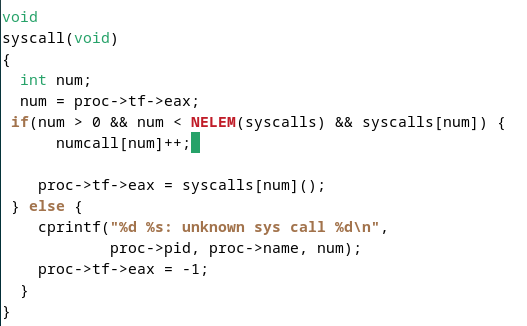
Βήμα 6: Φτιάχνουμε πρόγραμμα shutdown.c στον φάκελο User και το προςθέτουμε στο Makefile. Τώρα με την εντολή shutdown, o qemu emulator κλείνει.

******shutdown.c:** **terminal:**

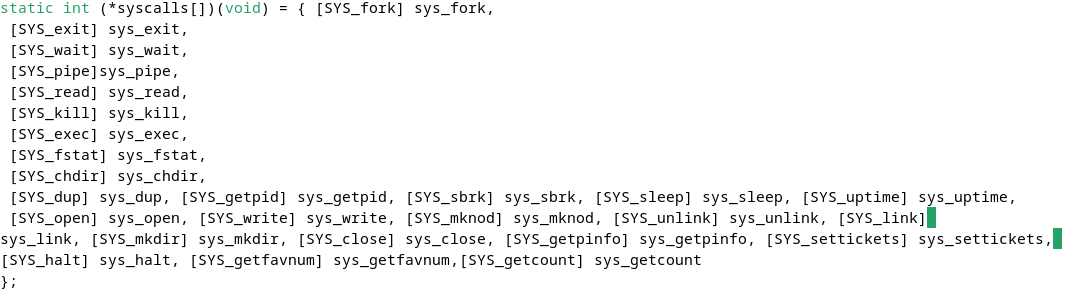
## Άσκηση 7:

Προσθέστε μία κλήση συστήματος η οποία θα επιστρέφει το πόσες φορές έχει εκτελεστεί μία συγκεκριμένη κλήση συστήματος η οποία θα περνιέται σαν όρισμα. Η κλήση που θα φτιάξετε πρέπει να έχει το εξής πρωτότυπο: int getcount(int syscall).:

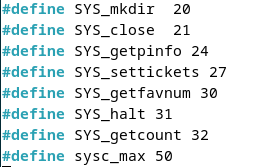
Για να μετράμε τις φορές που θα κληθεί η κάθε συνάρτηση θα δημιουργήσουμε έναν κενό πίνακα στο syscall.c, τον int numcall[sysc\_max]. Η τιμή sysc\_max έχει γίνει #define sysc\_max 50 στο syscall.h έτσι ώστε ο πίνκας να χωράει όλες τις τιμές των systemcalls και να έχουμε περιθώριο να φτιάξουμε μερικές παραπάνω.

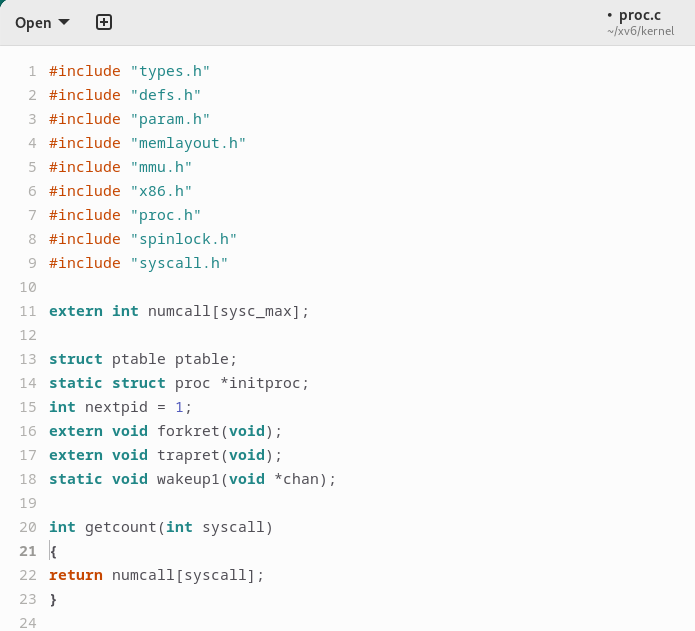
Θέλουμε στην συνάρτηση syscall() του syscall.c, κάθε φορά που καλείται να αυξάνεται ο αριθμός στην αντίστοιχη θέση του πίνακα.

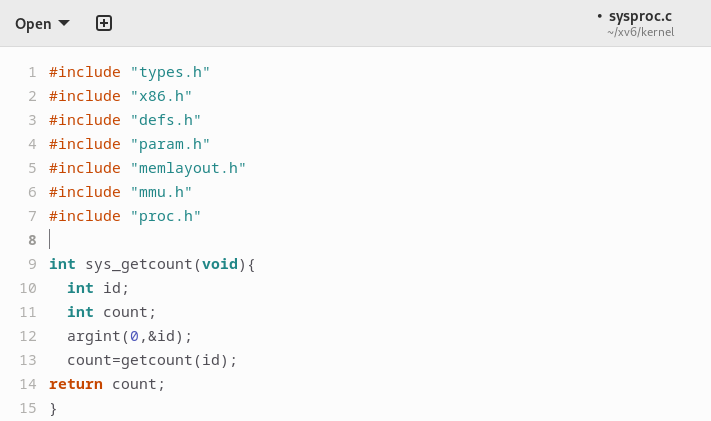
Περνάμε στην υλοποίηση της getcount τώρα. Στο αρχείο syscall.c την καταγράφουμε με extern int sys\_getcount(void); ώστε να μπορεί να κληθεί από την syscall (o compiler θα δει ότι είναι σε άλλο αρχείο). Επίσης την προσθέτουμε στον πίνακα των συναρτήσεων την sys\_getcount, και τον αριθμό κλήσης της στο syscall.h.

**syscall.c:**



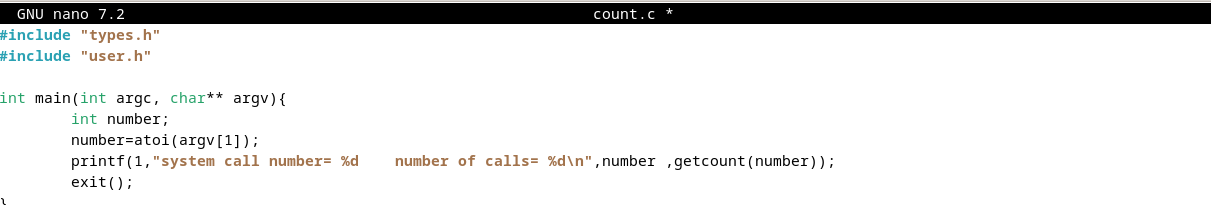
**syscall.h**

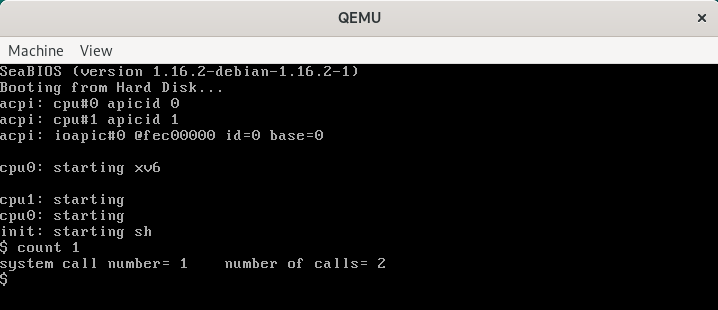
Υλοποιούμε τώρα στα αρχεία sysproc.c και proc.c την διαδικασία αυτή:

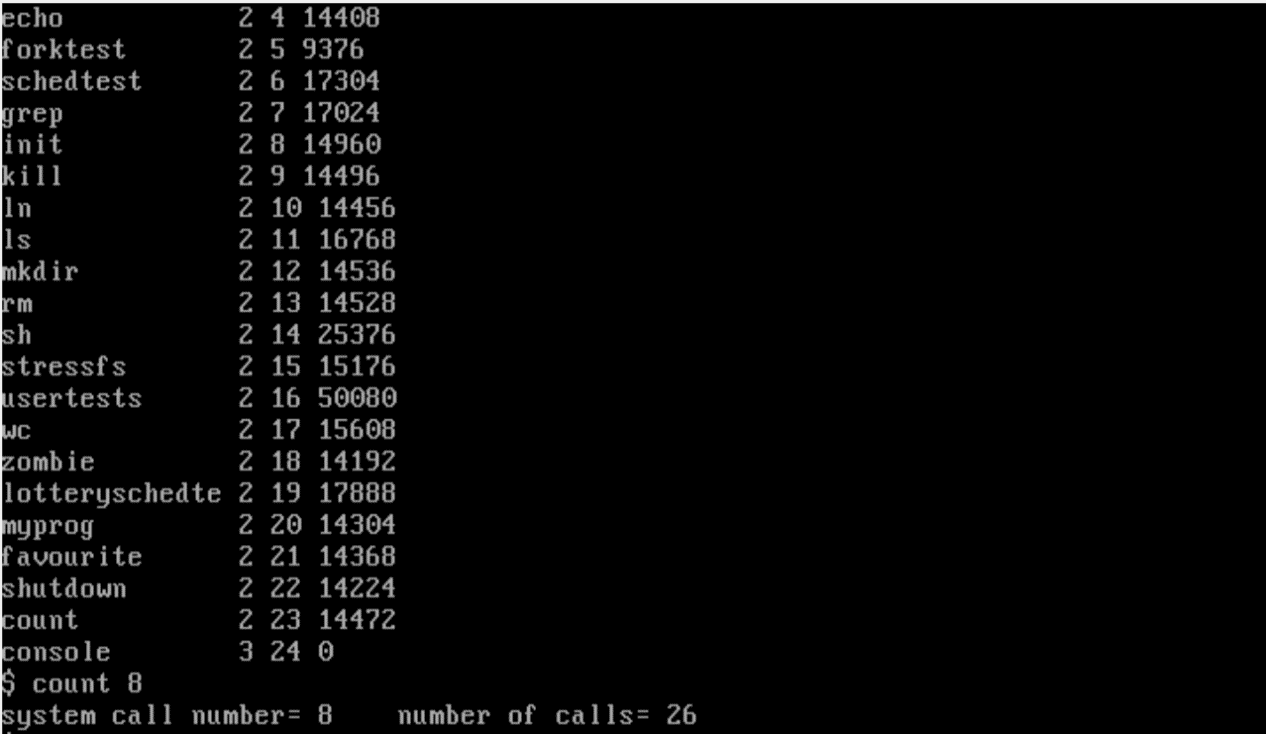


Το argint(0,&id) περνάει το πρώτο όρισμα που θα δώσει ο χρήστης ως τιμή στο int, δηλαδή το νούμερο της system call που θέλουμε να δούμε πόσες φορές κλήθηκε.

Τώρα φτιάχνουμε ένα User πρόγραμμα που να τυπώνει τα αποτελέσματα αυτής της εντολής και το προσθέτουμε στο Makefile.





Καλούμε την ls και μετά ξανά την count για το 8(της fstat η οποία χρησιμοποιείται από την ls). Διαπιστώνουμε ότι όντως έχει κληθεί 26 φορές.

## Άσκηση 8:

Προσθέστε μία κλήση συστήματος η οποία θα τερματίζει μια τυχαία διεργασία με πρότυπο int killrandom(void). Θα χρειαστεί να προσθέσετε μία γεννήτρια τυχαίων αριθμών στο kernel. Πώς μπορείτε να βρείτε τα PID όλων των τρεχόντων διεργασιών στο σύστημα;

*Σημείωση: Για να σκοτώσετε μία διεργασία δοθέντος ενός συγκεκριμένου PID μπορείτε να χρησιμοποιήσετε την κλήση συστήματος kill.*

Από το βιβλίο [1] γνωρίζουμε ότι:

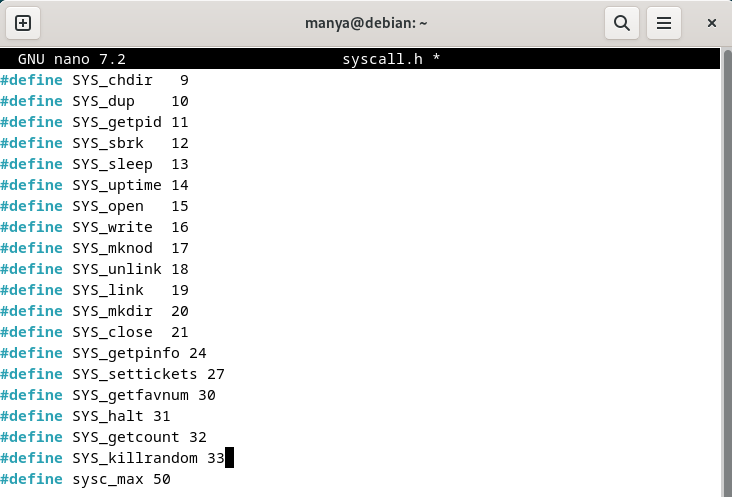
“… kill (2975) lets one process request that another be terminated. It would be too complex for kill to directly destroy the victim process, since the victim might be executing on another CPU or sleeping while midway through updating kernel data structures. To address these challenges, kill does very little: it just sets the victim’s p->killed and, if it is sleeping, wakes it up. Eventually the victim will enter or leave the kernel, at which point code in trap will call exit if p->killed is set. If the victim is running in user space, it will soon enter the kernel by making a system call or because the timer (or some other device) interrupts.”

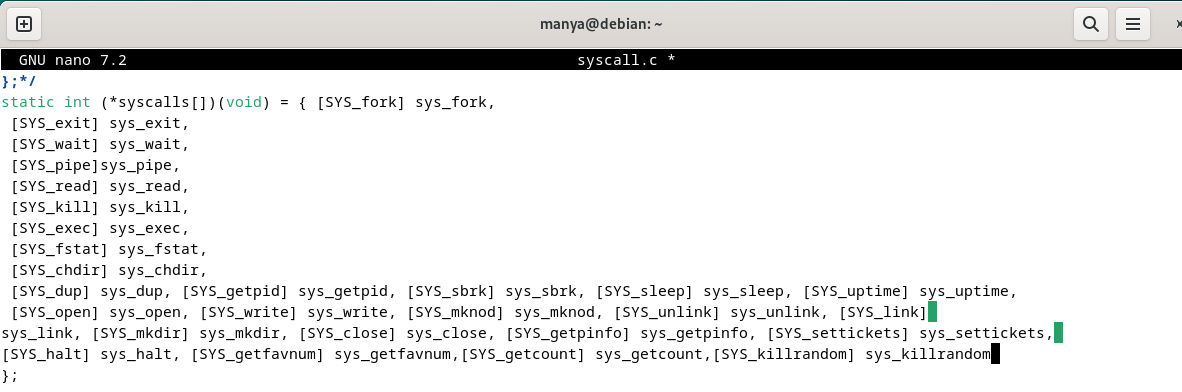
Η kill λοιπόν χρειάζεται απλά το pid μιας διεργασίας για να θέσει την κατάλληλη τιμή στο ptable της, και η trap αργότερα να την τερματίσει με την exit().

Επομένως, πρέπει να φτιάξουμε:

1. Μια γεννήτρια τυχαίων αριθμών
2. Μια system call που σκοτώνει ότι επιστρέφει η γεννήτρια int killrandom(void).
3. Ένα User program που καλεί την system call int killrandom(void).

Ορισμός αριθμού system call:

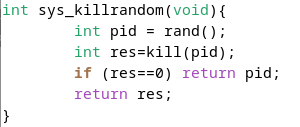


Προσθήκη συνάρτησης στο syscall.c

extern int sys\_killrandom(void);

Προσθήκη στο usys.S:

Υλοποίηση της συνάρτησης στο sysproc.c:



Αν η kill είναι επιτυχής επιστρέφει 0 και η sys\_killrandom θα επιστρέψει το pid. Αλλιώς επιστρέφει -1 (θα χρησιμοποιηθεί για να τυπωθεί error message).

Γεννήτρια τυχαίων αριθμών:

Σημείωση: Θέλουμε η μέγιστη τιμή που να επιστρέφει να είναι όσες και ο μέγιστος αριθμός από διεργασίες που μπορούν να υπάρξουν. Αυτός ορίζεται στο header param.h που είναι included στο sysproc.c και ονομάζεται NPROC. (#define NPROC 64).

*// The following functions define a portable implementation of rand and srand.*

static uint64 next = 1; *// αρχικοποίηση του next*

**static** unsigned long int next = 1; *// NB: "unsigned long int" is assumed to be 32 bits wide*

int rand(void) *// RAND\_MAX assumed to be 32767*

{

next = next \* 1103515245 + 12345;

**return** (unsigned int) (next / 65536) % **NPROC**;

}

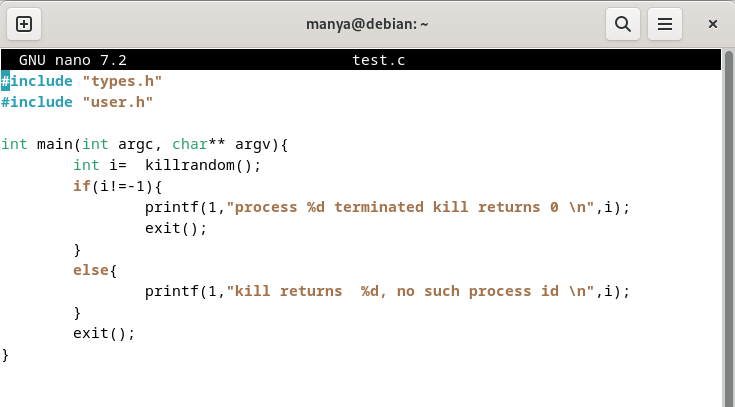
void srand(unsigned int seed)

{

next = seed;

}

User program για έλεγχο σωστής λειτουργίας:

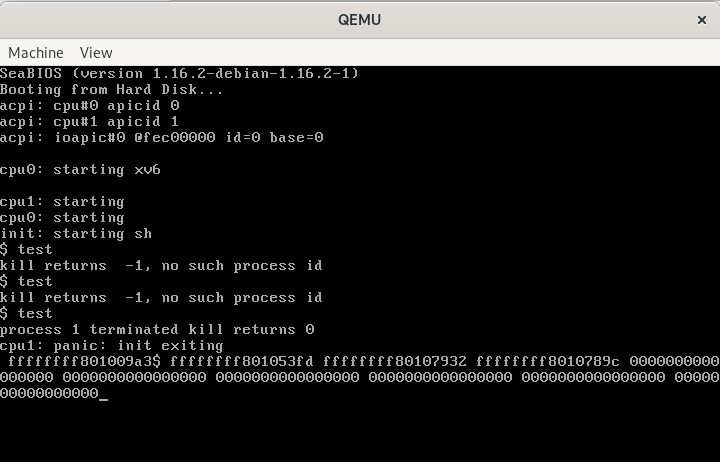


Αν δεν εκτελεστούν όλα με επιτυχία θα τυπωθεί πως δεν υπήρχε τέτοιο process id.

Αλλιώς θα τυπωθεί το process id της διεργασίας.

Προσθήκη στο Makefile με test\

Δοκιμή:



## Βιβλιογραφία

[1] ] Russ Cox, Frans Kaashoek και Robert Morris. The Design of the xv6 x86 Operating System. 2019. URL: <https://pdos.csail.mit.edu/6.828/2018/xv6/book-rev10.pdf>.

<https://wiki.osdev.org/Random_Number_Generator#Pseudorandom_number_generators>

<https://linux.die.net/man/2/fstat>