

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Δραστηριότητα 3

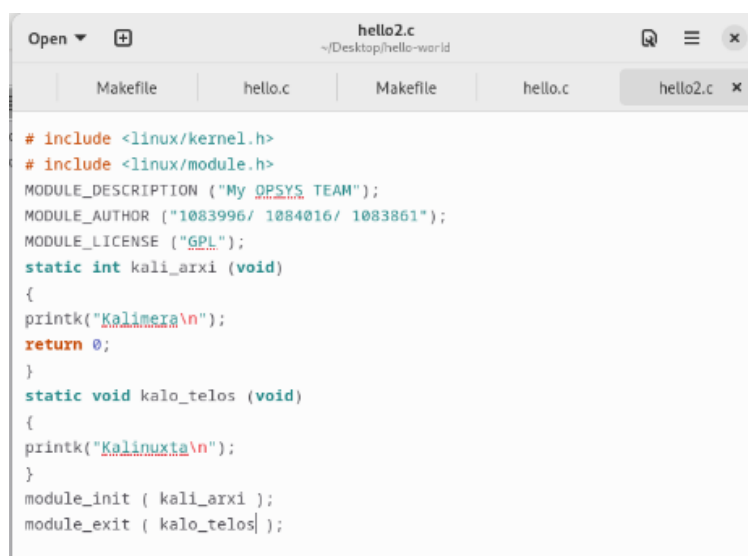
ΜΠΑΣΑΓΙΑΝΝΗ ΓΕΩΡΓΙΑ 1084016

ΠΑΝΟΥΡΓΙΑΣ ΑΝΤΩΝΙΟΣ 1083996

ΣΤΕΡΓΙΟΠΟΥΛΟΣ ΓΕΩΡΓΙΟΣ 1083861

Άσκηση 1

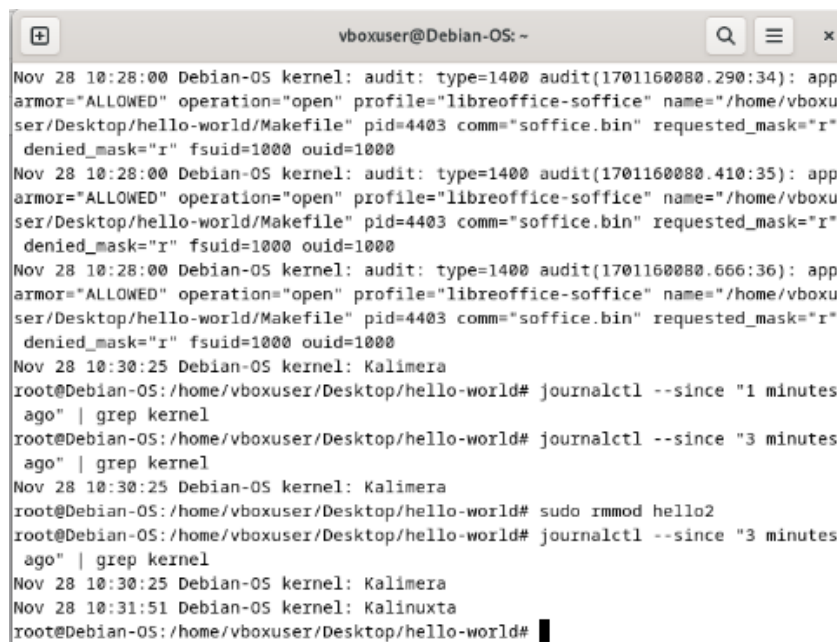
Στην άσκηση αυτή τροποποιήσαμε το αρχικό module με σκοπό να τυπώνει ένα διαφορετικό μήνυμα. Οι νέοι κώδικες έχουν την παρακάτω μορφή:



```
#include <linux/kernel.h>
#include <linux/module.h>
MODULE_DESCRIPTION("My OPSYS TEAM");
MODULE_AUTHOR("1083996/ 1084016/ 1083861");
MODULE_LICENSE("GPL");
static int kali_arxi (void)
{
    printk("Kalimera\n");
    return 0;
}
static void kalo_telos (void)
{
    printk("Kalinuxta\n");
}
module_init ( kali_arxi );
module_exit ( kalo_telos );
```

Επιπλέον στο Makefile βάλαμε αντι για hello.o, προφανώς, hello2.o

Αφού κάναμε make το hello2.c, φορτώσαμε το module στο kernel και με τις παρακάτω εντολές είδαμε τα μηνύματά μας, από την φόρτωση και εκφόρτωση του module αντίστοιχα (logs)



```
vboxuser@Debian-OS: ~
Nov 28 10:28:00 Debian-OS kernel: audit: type=1400 audit(1701160080.290:34): app
armor="ALLOWED" operation="open" profile="libreoffice-soffice" name="/home/vboxu
ser/Desktop/hello-world/Makefile" pid=4403 comm="soffice.bin" requested_mask="r"
denied_mask="r" fsuid=1000 ouid=1000
Nov 28 10:28:00 Debian-OS kernel: audit: type=1400 audit(1701160080.410:35): app
armor="ALLOWED" operation="open" profile="libreoffice-soffice" name="/home/vboxu
ser/Desktop/hello-world/Makefile" pid=4403 comm="soffice.bin" requested_mask="r"
denied_mask="r" fsuid=1000 ouid=1000
Nov 28 10:28:00 Debian-OS kernel: audit: type=1400 audit(1701160080.666:36): app
armor="ALLOWED" operation="open" profile="libreoffice-soffice" name="/home/vboxu
ser/Desktop/hello-world/Makefile" pid=4403 comm="soffice.bin" requested_mask="r"
denied_mask="r" fsuid=1000 ouid=1000
Nov 28 10:30:25 Debian-OS kernel: Kalimera
root@Debian-OS: /home/vboxuser/Desktop/hello-world# journalctl --since "1 minutes
ago" | grep kernel
root@Debian-OS: /home/vboxuser/Desktop/hello-world# journalctl --since "3 minutes
ago" | grep kernel
Nov 28 10:30:25 Debian-OS kernel: Kalimera
root@Debian-OS: /home/vboxuser/Desktop/hello-world# sudo rmmod hello2
root@Debian-OS: /home/vboxuser/Desktop/hello-world# journalctl --since "3 minutes
ago" | grep kernel
Nov 28 10:30:25 Debian-OS kernel: Kalimera
Nov 28 10:31:51 Debian-OS kernel: Kalinuxta
root@Debian-OS: /home/vboxuser/Desktop/hello-world#
```

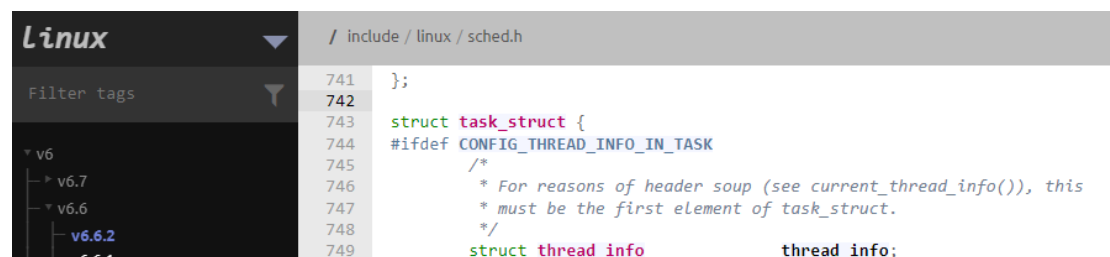
Άσκηση 2

Στην άσκηση αυτή μας ζητείται να περιγράψουμε την δομή `task_struct` που περιέχεται στο Linux Kernel Code. Βρίσκουμε λοιπόν την δομή αυτή, μέσω του elixir.bootlin.com, και συμπαιρνούμε τα εξής:

- Είναι μια απο τις κεντρικές δομές δεδομένων και περιέχει όλα τα χαρακτηριστικά, τις λεπτομέρειες ανγώρισης και τις εγγαφές κατανομής πόρων που διαθέτει μια διεργασία.

Πιο συγκεκριμένα:

- Είναι process descriptor και περιέχει όλες τις πληροφορίες που χρειάζεται ο πυρήνας για αυτήν την διεργασία
- Στον πυρήνα τα tasks αναφέρονται άμεσα με δείκτη στο task struct τους.
- Ο πυρήνας αποθηκεύει την λίστα διεργασιών σε μια κυκλική διπλά διασυνδεδεμένη λίστα με όνομα task list. Κάθε στοιχείο του task list είναι ένα process descriptor τύπου `task_struct`.
- Τα περιεχόμενα του `task_struct` είναι open files, processes addresses space, pending signals, process state, process context, process family tree κ.α



Άσκηση 3

Στην άσκηση αυτή δημιουργήσαμε ένα νέο αρχείο `list-processes.c` με σκοπό να δημιουργήσουμε ένα module πυρήνα το οποίο τυπώνει, όταν φορτώνεται, πληροφορίες για όλες τις υπάρχουσες διεργασίες.

Για να το επιτύχουμε αυτό κάνουμε χρήση της macro εντολής `for_each_process(p)` που ορίζεται στο `linux/include/linux/sched/signal.h` και υλοποιεί ουσιαστικά την διαπέραση όλων των processes.

```
#define for_each_process(p) \
    for (p = &init_task ; (p = next_task(p)) != &init_task ; )

extern bool current_is_single_threaded(void);

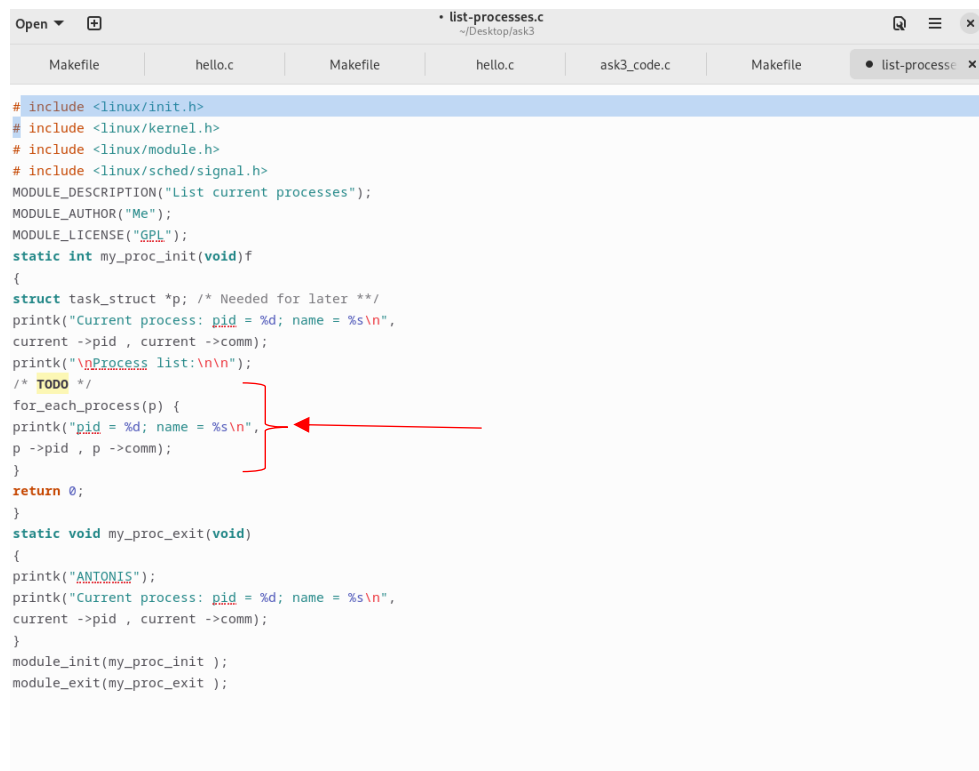
/*
 * Without tasklist/siglock it is only rcu-safe if g can't exit/exec,
 * otherwise next_thread(t) will never reach g after list_del_rcu(g).
 */
#define while_each_thread(g, t) \
    while ((t = next_thread(t)) != g)

#define __for_each_thread(signal, t) \
    list_for_each_entry_rcu(t, &(signal)->thread_head, thread_node)

#define for_each_thread(p, t) \
    __for_each_thread((p)->signal, t)

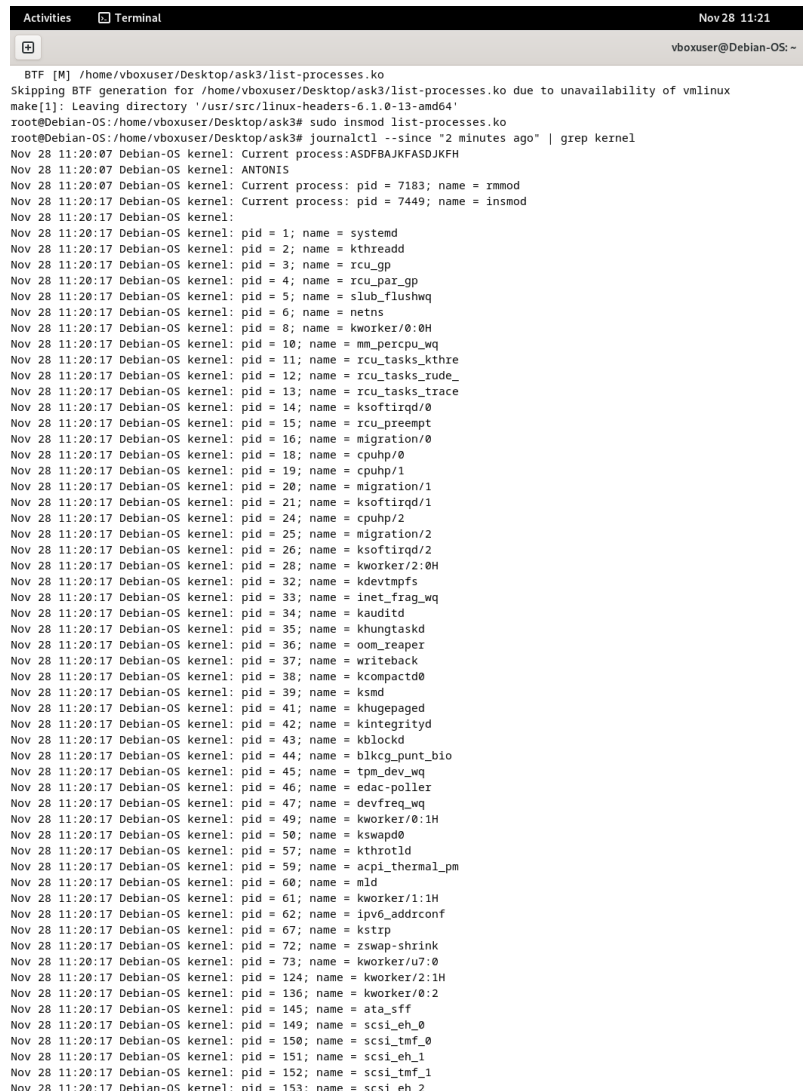
/* Careful: this is a double loop, 'break' won't work as expected. */
#define for_each_process_thread(p, t) \
    for_each_process(p) for_each_thread(p, t)
```

Ο κώδικας έχει την παρακάτω, πλέον, μορφή:



```
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/sched/signal.h>
MODULE_DESCRIPTION("List current processes");
MODULE_AUTHOR("Me");
MODULE_LICENSE("GPL");
static int my_proc_init(void) {
    struct task_struct *p; /* Needed for later */
    printk("Current process: pid = %d; name = %s\n",
           current ->pid, current ->comm);
    printk("\nProcess list:\n\n");
    /* TODO */
    for_each_process(p) {
        printk("pid = %d; name = %s\n",
               p ->pid, p ->comm);
    }
    return 0;
}
static void my_proc_exit(void) {
    printk("ANTONIS");
    printk("Current process: pid = %d; name = %s\n",
           current ->pid, current ->comm);
}
module_init(my_proc_init);
module_exit(my_proc_exit);
```

Και φορτώνοντας το module, και βλέποντας τα πιο πρόσφατα logs, βλέπουμε ότι επιτύχαμε τον σκοπό μας. Ενδεικτικό μέρος, των logs:



```
BTf [M] /home/vboxuser/Desktop/ask3/list-processes.ko
Skipping BTF generation for /home/vboxuser/Desktop/ask3/list-processes.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.1.0-13-amd64'
root@Debian-OS: /home/vboxuser/Desktop/ask3# sudo insmod list-processes.ko
root@Debian-OS: /home/vboxuser/Desktop/ask3# journalctl --since "2 minutes ago" | grep kernel
Nov 28 11:20:07 Debian-OS kernel: Current process:ASDFBAJKFASDJKFH
Nov 28 11:20:07 Debian-OS kernel: ANTONIS
Nov 28 11:20:07 Debian-OS kernel: Current process: pid = 7183; name = rmmmod
Nov 28 11:20:17 Debian-OS kernel: Current process: pid = 7449; name = insmod
Nov 28 11:20:17 Debian-OS kernel:
Nov 28 11:20:17 Debian-OS kernel: pid = 1; name = systemd
Nov 28 11:20:17 Debian-OS kernel: pid = 2; name = kthreadd
Nov 28 11:20:17 Debian-OS kernel: pid = 3; name = rcu_gp
Nov 28 11:20:17 Debian-OS kernel: pid = 4; name = rcu_par_gp
Nov 28 11:20:17 Debian-OS kernel: pid = 5; name = slub_flushwq
Nov 28 11:20:17 Debian-OS kernel: pid = 6; name = netns
Nov 28 11:20:17 Debian-OS kernel: pid = 8; name = kworker/0:0H
Nov 28 11:20:17 Debian-OS kernel: pid = 10; name = mm_percpu_wq
Nov 28 11:20:17 Debian-OS kernel: pid = 11; name = rcu_tasks_kthre
Nov 28 11:20:17 Debian-OS kernel: pid = 12; name = rcu_tasks_rude_
Nov 28 11:20:17 Debian-OS kernel: pid = 13; name = rcu_tasks_trace
Nov 28 11:20:17 Debian-OS kernel: pid = 14; name = ksoftirqd/0
Nov 28 11:20:17 Debian-OS kernel: pid = 15; name = rcu_preempt
Nov 28 11:20:17 Debian-OS kernel: pid = 16; name = migration/0
Nov 28 11:20:17 Debian-OS kernel: pid = 18; name = cpuhp/0
Nov 28 11:20:17 Debian-OS kernel: pid = 19; name = cpuhp/1
Nov 28 11:20:17 Debian-OS kernel: pid = 20; name = migration/1
Nov 28 11:20:17 Debian-OS kernel: pid = 21; name = ksoftirqd/1
Nov 28 11:20:17 Debian-OS kernel: pid = 24; name = cpuhp/2
Nov 28 11:20:17 Debian-OS kernel: pid = 25; name = migration/2
Nov 28 11:20:17 Debian-OS kernel: pid = 26; name = ksoftirqd/2
Nov 28 11:20:17 Debian-OS kernel: pid = 28; name = kworker/2:0H
Nov 28 11:20:17 Debian-OS kernel: pid = 32; name = kdevtmpfs
Nov 28 11:20:17 Debian-OS kernel: pid = 33; name = inet_frag_wq
Nov 28 11:20:17 Debian-OS kernel: pid = 34; name = kauditd
Nov 28 11:20:17 Debian-OS kernel: pid = 35; name = khungtaskd
Nov 28 11:20:17 Debian-OS kernel: pid = 36; name = oom_reaper
Nov 28 11:20:17 Debian-OS kernel: pid = 37; name = writeback
Nov 28 11:20:17 Debian-OS kernel: pid = 38; name = kcompactd0
Nov 28 11:20:17 Debian-OS kernel: pid = 39; name = ksm
Nov 28 11:20:17 Debian-OS kernel: pid = 41; name = khugepaged
Nov 28 11:20:17 Debian-OS kernel: pid = 42; name = kintegrityd
Nov 28 11:20:17 Debian-OS kernel: pid = 43; name = kblockd
Nov 28 11:20:17 Debian-OS kernel: pid = 44; name = blkcg_punt_bio
Nov 28 11:20:17 Debian-OS kernel: pid = 45; name = tpm_dev_wq
Nov 28 11:20:17 Debian-OS kernel: pid = 46; name = edac-poller
Nov 28 11:20:17 Debian-OS kernel: pid = 47; name = devfreq_wq
Nov 28 11:20:17 Debian-OS kernel: pid = 49; name = kworker/0:1H
Nov 28 11:20:17 Debian-OS kernel: pid = 50; name = kswapd0
Nov 28 11:20:17 Debian-OS kernel: pid = 57; name = kthrotld
Nov 28 11:20:17 Debian-OS kernel: pid = 59; name = acpi_thermal_pm
Nov 28 11:20:17 Debian-OS kernel: pid = 60; name = mld
Nov 28 11:20:17 Debian-OS kernel: pid = 61; name = kworker/1:1H
Nov 28 11:20:17 Debian-OS kernel: pid = 62; name = ipv6_addrconf
Nov 28 11:20:17 Debian-OS kernel: pid = 67; name = kstrp
Nov 28 11:20:17 Debian-OS kernel: pid = 72; name = zswap-shrink
Nov 28 11:20:17 Debian-OS kernel: pid = 73; name = kworker/u7:0
Nov 28 11:20:17 Debian-OS kernel: pid = 124; name = kworker/2:1H
Nov 28 11:20:17 Debian-OS kernel: pid = 136; name = kworker/0:2
Nov 28 11:20:17 Debian-OS kernel: pid = 145; name = ata_sff
Nov 28 11:20:17 Debian-OS kernel: pid = 149; name = scsi_eh_0
Nov 28 11:20:17 Debian-OS kernel: pid = 150; name = scsi_tm_f_0
Nov 28 11:20:17 Debian-OS kernel: pid = 151; name = scsi_eh_1
Nov 28 11:20:17 Debian-OS kernel: pid = 152; name = scsi_tm_f_1
Nov 28 11:20:17 Debian-OS kernel: pid = 153; name = scsi_eh_2
```

Άσκηση 4

Το ζητούμενο της άσκησης αυτής, είναι πάλι να εκτυπώσουμε τις πληροφορίες για μια διεργασία αλλά και για όλες τις θυγατρικές της.

Οι θυγατρικές δημιουργούνται μέσω του δοθέντος αρχείου forking.c, το οποίο ανα SLEEP TIME δημιουργεί ένα child process (μέσω της fork) στην διεργασία με το PID που φορτώθηκε αρχικά.

Θέλουμε μέσω του νέου module που θα δημιουργήσουμε, να «κοιτάμε» το task struct που αντιστοιχεί στην παραπάνω διεργασία και μέσω αυτού να εκτυπώσουμε τα παιδιά της.

1. Στο linux/tools/include/linux/sched.h βρίσκουμε τα πεδία children και sibling, τα οποία είναι τα head μιας λίστας. Οι λίστες αυτές με χρήση των κατάλληλων macros θα γεμίσουν με τα στοιχεία των children process και sibling αντίστοιχα.

```
/*
 * Children/sibling form the list of natural children:
 */
struct list_head children;
struct list_head sibling;
```

2. Βλέπουμε επίσης στο linux/tools/include/linux/list.h την macro list_for_each_entry, η οποία θα μας βοηθήσει να διαπεράσουμε την λίστα με τα child processes.

```
763 /**
764  * list_for_each_entry - iterate over list of given type
765  * @pos: the type * to use as a loop cursor.
766  * @head: the head for your list.
767  * @member: the name of the list_head within the struct.
768  */
769 #define list_for_each_entry(pos, head, member)
770     for (pos = list_first_entry(head, typeof(*pos), member);
771         !list_entry_is_head(pos, head, member);
772         pos = list_next_entry(pos, member))
773
774 /**
```

Κάνουμε λοιπόν τις απαραίτητες προσθήκες:

Στο αρχείο list-children.c προσθέτουμε αυτόν τον κώδικα:

```
if (task)
{
    printk("pid: %d, name: %s\n", task->pid, task->comm);

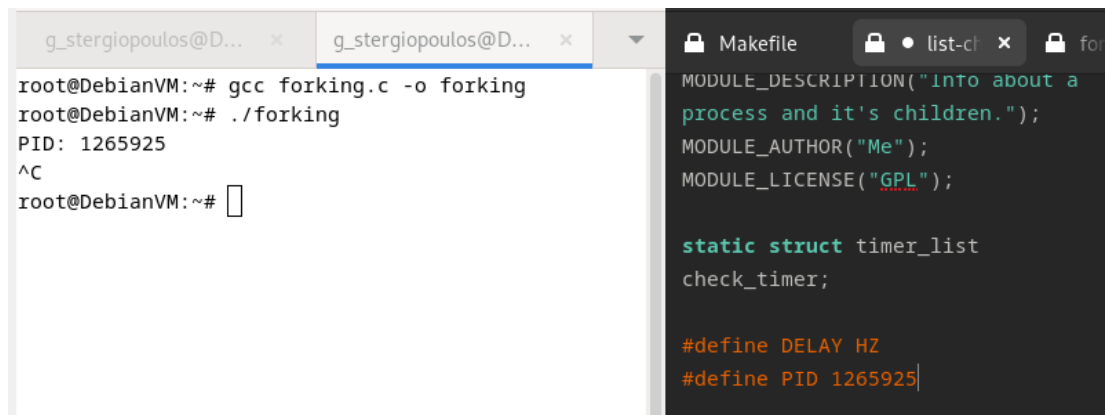
    /* TODO: iterate over the process' children and print their PID */
    list_for_each_entry(child, &task->children, sibling) {
        printk("CHILD %d\n", child->pid);
    }
}
```

Μέσω αυτού πηγαίνουμε σε κάθε task, βρίσκουμε το παιδί του και κοιτάμε τα αδέρφια του. Με αυτόν τον τρόπο παίρνουμε όλες τις διεργασίες-παιδιά που δημιούργησε η fork μας στο αρχείο forking.c

Πως ελέγξαμε την λειτουργία?

Αρχικά κάναμε compile (gcc forking.c -o forking) το αρχείο αυτό που μας δημιουργεί τα παιδιά, και το τρέξαμε (./forking).

Η εκτέλεση αυτού μας επέστρεψε το PID της γονικής διεργασίας το οποίο εμείς ορίσαμε μέσω του define στο αρχείο list-children.c.



```
g_stergiopoulos@D... x g_stergiopoulos@D... x
root@DebianVM:~# gcc forking.c -o forking
root@DebianVM:~# ./forking
PID: 1265925
^C
root@DebianVM:~#

Makefile
MODULE_DESCRIPTION("Info about a
process and it's children.");
MODULE_AUTHOR("Me");
MODULE_LICENSE("GPL");

static struct timer_list
check_timer;

#define DELAY HZ
#define PID 1265925
```

Όσο στο δεύτερο terminal τρέχει η forking, και αφού βάλουμε κατάλληλη τιμή στο #define PID, όπως αναφέραμε παραπάνω, θα κάνουμε make.

Το make θα μας δημιουργήσει το module, το οποίο φορτώνουμε με την εντολή:

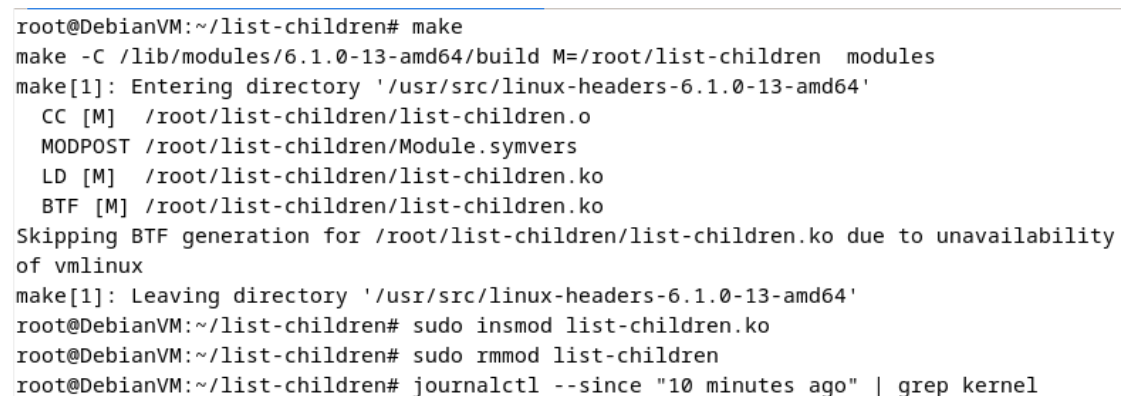
```
sudo insmod list-children.ko
```

Και αφαιρούμε με την:

```
sudo rmmod list-children
```

Τέλος εκτελούμε την:

```
journalctl --since "10 minutes ago" | grep kernel
```



```
root@DebianVM:~/list-children# make
make -C /lib/modules/6.1.0-13-amd64/build M=/root/list-children modules
make[1]: Entering directory '/usr/src/linux-headers-6.1.0-13-amd64'
  CC [M]  /root/list-children/list-children.o
  MODPOST /root/list-children/Module.symvers
  LD [M]  /root/list-children/list-children.ko
  BTF [M] /root/list-children/list-children.ko
Skipping BTF generation for /root/list-children/list-children.ko due to unavailability
of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.1.0-13-amd64'
root@DebianVM:~/list-children# sudo insmod list-children.ko
root@DebianVM:~/list-children# sudo rmmod list-children
root@DebianVM:~/list-children# journalctl --since "10 minutes ago" | grep kernel
```

Για να δούμε τα τελευταία logs στο σύστημα μας (μπορούμε να μειώσουμε τα 10 λεπτά για να μην μπλέκονται τα logs από προηγούμενες φορτώσεις module).

πχ παράδειγμα εκτέλεσης για την μητέρα διεργασία με PID=1288079:

```
Nov 28 20:40:53 DebianVM kernel: pid: 1288079, name: forking
Nov 28 20:40:53 DebianVM kernel: CHILD 1288081
Nov 28 20:40:53 DebianVM kernel: CHILD 1288086
Nov 28 20:40:53 DebianVM kernel: CHILD 1288101
Nov 28 20:40:53 DebianVM kernel: CHILD 1288136
Nov 28 20:40:53 DebianVM kernel: CHILD 1288160
Nov 28 20:40:53 DebianVM kernel: CHILD 1288208
Nov 28 20:40:53 DebianVM kernel: CHILD 1288436
Nov 28 20:40:53 DebianVM kernel: CHILD 1288753
Nov 28 20:40:53 DebianVM kernel: CHILD 1289035
Nov 28 20:40:53 DebianVM kernel: CHILD 1289686
Nov 28 20:40:53 DebianVM kernel: CHILD 1291028
```