

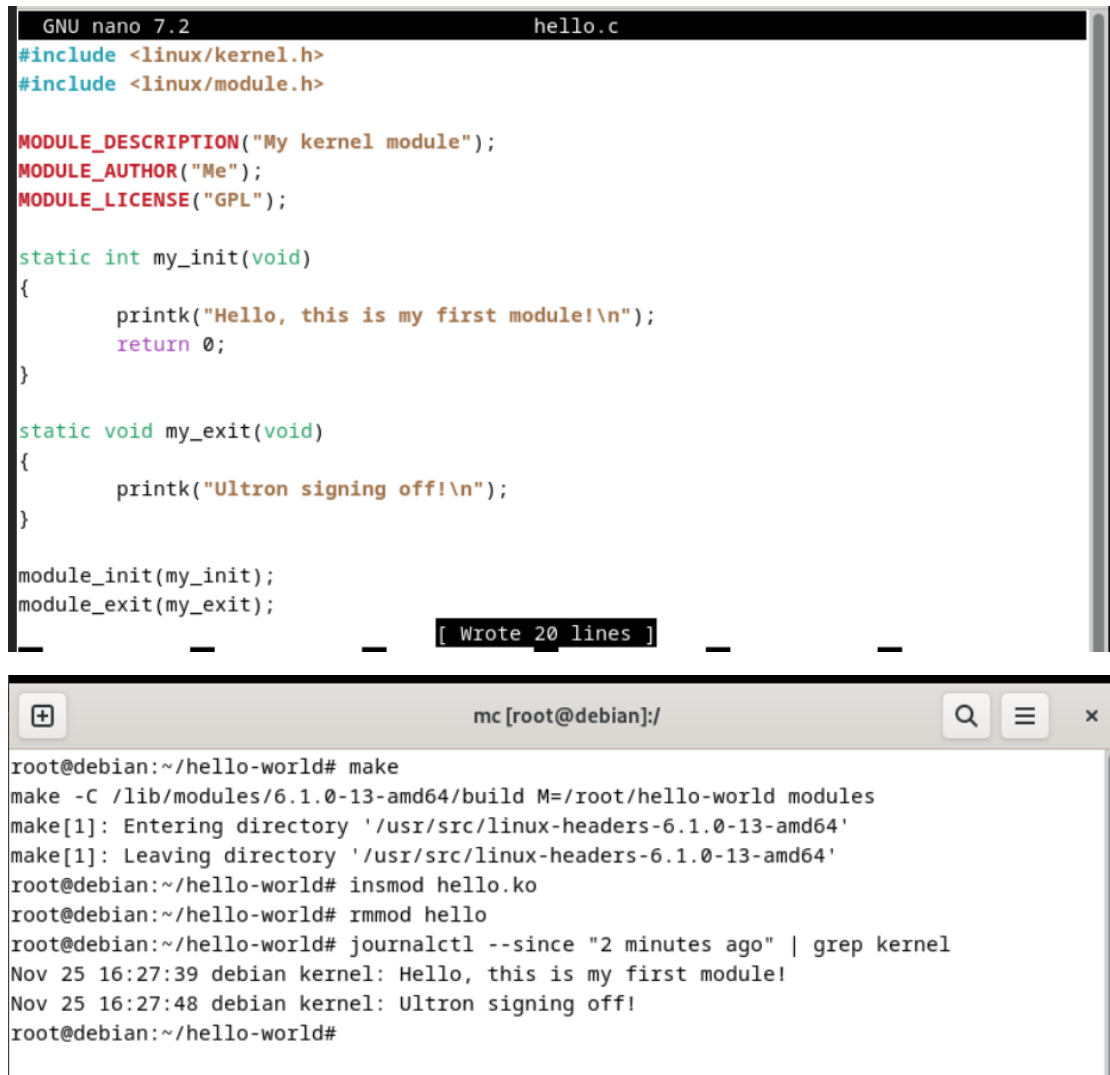
Λειτουργικά Συστήματα

3^η Εργασία

Τσάμπρας Κωνσταντίνος
υρ1083865

➔ 1)

Γράφουμε το hello.c και απλά αλλάζουμε το μήνυμα που τυπώνεται στα logs και ακολουθούμε την ίδια διαδικασία.



The image shows two terminal windows. The top window is a nano editor editing a file named 'hello.c'. The code defines a kernel module with a description, author, and license, and two functions: my_init which prints 'Hello, this is my first module!' and my_exit which prints 'Ultron signing off!'. The bottom window shows the execution of the module. It runs 'make' to compile the module into 'hello.ko', then 'insmod hello.ko' to load it, and 'rmmod hello' to unload it. Finally, it uses 'journalctl --since "2 minutes ago" | grep kernel' to view the kernel logs, which show the two messages printed by the module.

```
GNU nano 7.2 hello.c
#include <linux/kernel.h>
#include <linux/module.h>

MODULE_DESCRIPTION("My kernel module");
MODULE_AUTHOR("Me");
MODULE_LICENSE("GPL");

static int my_init(void)
{
    printk("Hello, this is my first module!\n");
    return 0;
}

static void my_exit(void)
{
    printk("Ultron signing off!\n");
}

module_init(my_init);
module_exit(my_exit);
[ Wrote 20 lines ]

mc [root@debian]:/
root@debian:~/hello-world# make
make -C /lib/modules/6.1.0-13-amd64/build M=/root/hello-world modules
make[1]: Entering directory '/usr/src/linux-headers-6.1.0-13-amd64'
make[1]: Leaving directory '/usr/src/linux-headers-6.1.0-13-amd64'
root@debian:~/hello-world# insmod hello.ko
root@debian:~/hello-world# rmmod hello
root@debian:~/hello-world# journalctl --since "2 minutes ago" | grep kernel
Nov 25 16:27:39 debian kernel: Hello, this is my first module!
Nov 25 16:27:48 debian kernel: Ultron signing off!
root@debian:~/hello-world#
```

➔ 2)

Η δομή δεδομένων του linux task_struct είναι ανάλογη με την δομή δεδομένων process του xv6, αφού και αυτή αντιπροσωπεύει μια διεργασία και αποθηκεύει όλες τις πληροφορίες που χρειάζεται μια διεργασία όπως η κατάσταση της διεργασίας, το αναγνωριστικό της διεργασίας, λεπτομέρειες διαχείρισης μνήμης, δείκτες αρχείων και πολλά άλλα. Όμως το xv6 είναι ένα εκπαιδευτικό kernel και έτσι η δομή process ήταν αρκετά απλή και δεν περιείχε πάρα πολλές πληροφορίες. Αντιθέτως η δομή task_struct χρησιμοποιείται στα λειτουργικά linux που εξυπηρετούν τεράστιο πλήθος διαφορετικών λειτουργιών, οπότε είναι και πολύ πιο σύνθετες δομές.

```
elixir.bootlin.com/linux/latest/source/include/linux/sched.h
/ include / linux / sched.h All symbols
740 #endif
741 };
742
743 struct task_struct {
744 #ifdef CONFIG_THREAD_INFO_IN_TASK
745     /*
746      * For reasons of header soup (see current_thread_info()), this
747      * must be the first element of task_struct.
748      */
749     struct thread_info thread_info;
750 #endif
751     unsigned int __state;
752
753 #ifdef CONFIG_PREEMPT_RT
754     /* saved state for "spinlock sleepers" */
755     unsigned int saved_state;
756 #endif
757
758     /*
759      * This begins the randomizable portion of task_struct. Only
760      * scheduling-critical items should be added above here.
761      */
762     randomized_struct_fields_start
763
764     void *stack;
765     refcount_t usage;
766     /* Per task flags (PF_*), defined further below: */
767     unsigned int flags;
768     unsigned int ptrace;
769
770 #ifdef CONFIG_SMP
771     int on_cpu;
772     struct __call_single_node wake_entry;
773     unsigned int wakee_flips;
774     unsigned long wakee_flip_decay_ts;
775     struct task_struct *last_wakee;
776 }
```

...

➔ 3)

Δημιουργούμε το module αρχικά συγγράφοντας το αρχείο list.c (παράλειψη το ότι δεν ονομάστηκε list-processes.c). Όπως και στο αρχείο που δίνεται, στην εναρκτήρια συνάρτηση τυπώνουμε το current process (το οποίο πρέπει να είναι το insmod που εισάγει το module στο kernel).

Στην συνέχεια χρησιμοποιούμε το macro for_each_process χάρις στο οποίο η μεταβλητή p θα διατρέξει τον πίνακα των διεργασιών. Για κάθε μία από αυτές τις διεργασίες τυπώνουμε τον αριθμό τους (μέσα στην δικιά μας καταμέτρηση), το pid τους και το όνομά τους.

```
GNU nano 7.2 list.c
MODULE_AUTHOR("Me");
MODULE_LICENSE("GPL");

static int my_proc_init(void)
{
    struct task_struct *p; /* Needed for later */
    int counter = 0;
    printk("Current process: pid = %d; name = %s\n",
        current->pid, current->comm);
    printk("\nProcess list:\n\n");

    for_each_process(p)
    {
        printk("Process(%d): pid = %d; name = %s\n", counter++, p->pid, p->comm);
    }
    printk("End of list!\n");

    return 0;
}
```

Στα αποτελέσματα (τα οποία βλέπουμε τυπώνοντας τα logs) βλέπουμε ότι πρώτη διεργασία με pid=1 είναι πάντα το system το οποίο είναι «πρόγονος» των υπολοίπων διεργασιών. Η λίστα είναι μεγάλη (176 διεργασίες) με τελευταία να εμφανίζεται η insmod, κάτι που είναι λογικό, αφού αυτή η διεργασία δημιούργησε αυτήν την λίστα.

```
root@debian:~/list-processes# nano list.c
root@debian:~/list-processes# make
make -C /lib/modules/6.1.0-13-amd64/build M=/root/list-processes modules
make[1]: Entering directory '/usr/src/linux-headers-6.1.0-13-amd64'
  CC [M] /root/list-processes/list.o
  MODPOST /root/list-processes/Module.symvers
  LD [M] /root/list-processes/list.ko
  BTF [M] /root/list-processes/list.ko
Skipping BTF generation for /root/list-processes/list.ko due to unavailability of vmlinux
make[1]: Leaving directory '/usr/src/linux-headers-6.1.0-13-amd64'
root@debian:~/list-processes# sudo insmod list.ko
root@debian:~/list-processes# sudo rmmod list
root@debian:~/list-processes# journalctl --since "10 minutes ago" | grep kernel
Nov 25 20:26:36 debian kernel: Current process: pid = 8158; name = insmod
Nov 25 20:26:36 debian kernel:
Nov 25 20:26:36 debian kernel: Process(0): pid = 1; name = systemd
Nov 25 20:26:36 debian kernel: Process(1): pid = 2; name = kthreadd
Nov 25 20:26:36 debian kernel: Process(2): pid = 3; name = rcu_gp
Nov 25 20:26:36 debian kernel: Process(3): pid = 4; name = rcu_par_gp
Nov 25 20:26:36 debian kernel: Process(4): pid = 5; name = slub_flushwq
Nov 25 20:26:36 debian kernel: Process(5): pid = 6; name = netns
Nov 25 20:26:36 debian kernel: Process(6): pid = 10; name = mm_percpu_wq
Nov 25 20:26:36 debian kernel: Process(7): pid = 11; name = rcu_tasks_kthre
Nov 25 20:26:36 debian kernel: Process(8): pid = 12; name = rcu_tasks_rude_
Nov 25 20:26:36 debian kernel: Process(9): pid = 13; name = rcu_tasks_trace
Nov 25 20:26:36 debian kernel: Process(10): pid = 14; name = ksoftirqd/0
Nov 25 20:26:36 debian kernel: Process(11): pid = 15; name = rcu_preempt
Nov 25 20:26:36 debian kernel: Process(12): pid = 16; name = migration/0
```

...

```
mc [root@debian]:/
Nov 25 20:26:36 debian kernel: Process(153): pid = 2518; name = Privileged Cont
Nov 25 20:26:36 debian kernel: Process(154): pid = 2567; name = WebExtensions
Nov 25 20:26:36 debian kernel: Process(155): pid = 2602; name = Isolated Web Co
Nov 25 20:26:36 debian kernel: Process(156): pid = 3016; name = RDD Process
Nov 25 20:26:36 debian kernel: Process(157): pid = 3021; name = Utility Process
Nov 25 20:26:36 debian kernel: Process(158): pid = 3090; name = su
Nov 25 20:26:36 debian kernel: Process(159): pid = 3091; name = bash
Nov 25 20:26:36 debian kernel: Process(160): pid = 3809; name = Web Content
Nov 25 20:26:36 debian kernel: Process(161): pid = 3812; name = Web Content
Nov 25 20:26:36 debian kernel: Process(162): pid = 3847; name = Web Content
Nov 25 20:26:36 debian kernel: Process(163): pid = 4001; name = gvfsd-network
Nov 25 20:26:36 debian kernel: Process(164): pid = 4017; name = gvfsd-dnssd
Nov 25 20:26:36 debian kernel: Process(165): pid = 4500; name = seahorse
Nov 25 20:26:36 debian kernel: Process(166): pid = 6917; name = fwupd
Nov 25 20:26:36 debian kernel: Process(167): pid = 7557; name = kworker/u2:0
Nov 25 20:26:36 debian kernel: Process(168): pid = 8100; name = kworker/u2:2
Nov 25 20:26:36 debian kernel: Process(169): pid = 8118; name = kworker/0:1
Nov 25 20:26:36 debian kernel: Process(170): pid = 8119; name = kworker/0:0
Nov 25 20:26:36 debian kernel: Process(171): pid = 8128; name = kworker/u2:1
Nov 25 20:26:36 debian kernel: Process(172): pid = 8131; name = kworker/u2:3
Nov 25 20:26:36 debian kernel: Process(173): pid = 8132; name = kworker/0:2
Nov 25 20:26:36 debian kernel: Process(174): pid = 8156; name = sudo
Nov 25 20:26:36 debian kernel: Process(175): pid = 8157; name = sudo
Nov 25 20:26:36 debian kernel: Process(176): pid = 8158; name = insmod
Nov 25 20:26:36 debian kernel: End of list!
Nov 25 20:26:45 debian kernel: Current process: pid = 8162; name = rmmod
```

➔ 4)

- Forking_test.c

Αρχικά επεξεργάστηκα το forking.c (μετονομάστηκε σε forking_test.c) προσθέτοντας επιπλέον χρόνο αρχικά για να γίνουν οι απαραίτητες αλλαγές στο list-processes.c (προσθήκη του νέου PID). Επίσης τροποποίησα το αρχείο ώστε οι forked processes (τα παιδιά) να μην κάνουν fork ώστε να μην γεμίζει το log με μηνύματα forking που δεν χρειάζονται, αφού μας ενδιαφέρουν μόνο τα παιδιά της main process και όχι οι απόγονοι των παιδιών:

```
GNU nano 7.2 forking_test.c
#define SLEEP_TIME 2

int main(int argc , char *argv [])
{
    int n = 0;
    long pid= (long)getpid ();
    printf("PID: %ld\n", (long)getpid ());
    /* Sleep for SLEEP_TIME seconds. */
    sleep(60);
    while (1)
    {
        sleep(SLEEP_TIME );
        /* break after a few iterations (too many processes) */
        if (n++ > 15) exit (0);
        int forked = fork();
        if(!forked) {sleep(60); exit(0)};
        printf("Forked! PID: %d\n",forked);
    }
    return 0;
}
```

Έτσι έχουμε αρχική αναμονή 60 δευτερολέπτων και έπειτα, κάθε 2 δευτερόλεπτα η main process δημιουργεί ένα νέο παιδί και τυπώνει το PID του παιδιού.

- list-children.c

Αρχικά έχουμε την σωστή τιμή για το PID

```
GNU nano 7.2 list-children.c
#define DELAY HZ
#define PID 11395

static void print_process_info(struct timer_list *unused)
{
    struct task_struct* task;
    struct task_struct* child;
    struct list_head* list;
    /* Synchronization mechanism needed before searching for the process */
    rcu_read_lock();

    /* Search through the global namespace for the process with the given PID */
    task = pid_task(find_pid_ns(PID, &init_pid_ns), PIDTYPE_PID);
```

Στην συνέχεια έχουμε τον κώδικα που προσθέσαμε, στον οποίο απλά διατρέχουμε την διπλά διασυνδεδεμένη λίστα που είναι αποθηκευμένη στο children μέλος του task με το κατάλληλο macro. Για κάθε παιδί που βρίσκουμε τυπώνουμε το pid και το όνομα του process. Η συνάρτηση αυτή καλείται ανά τακτά χρονικά διαστήματα ανάλογα με τον timer που έχουμε ορίσει (εδώ πρακτικά κάθε δευτερόλεπτο).

```

/* Search through the global namespace for the process with the given PID */
task = pid_task(find_pid_ns(PID, &init_pid_ns), PIDTYPE_PID);

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

    list_for_each(list, &task->children)
    {
        child = list_entry(list, struct task_struct, sibling);
        printk("  child pid: %d, name: %s\n", child->pid, child->comm);
    }
}

rcu_read_unlock(); /* Task pointer is now invalid! */

/* Restart the timer. */
check_timer.expires = jiffies + DELAY;
add_timer(&check_timer);

```

- Αποτέλεσμα:

```

root@debian:~/forking# nano list-children.c
root@debian:~/forking# make
make -C /lib/modules/6.1.0-13-amd64/build M=/root/forking modules
make[1]: Entering directory '/usr/src/linux-headers-6.1.0-13-amd64'
CC [M] /root/forking/list-children.o
MODPOST /root/forking/Module.symvers
LD [M] /root/forking/list-children.ko
BTF [M] /root/forking/list-children.ko
Skipping BTF generation for /root/forking/list-children.ko due to unavailability of vml
inux
make[1]: Leaving directory '/usr/src/linux-headers-6.1.0-13-amd64'
root@debian:~/forking# sudo insmod list-children.ko
root@debian:~/forking# journalctl --since "3 minutes ago" | grep kernel
Nov 26 16:09:35 debian kernel: pid: 11395, name: a.out
Nov 26 16:09:36 debian kernel: pid: 11395, name: a.out
Nov 26 16:09:37 debian kernel: pid: 11395, name: a.out
Nov 26 16:09:38 debian kernel: pid: 11395, name: a.out
Nov 26 16:09:39 debian kernel: pid: 11395, name: a.out
Nov 26 16:09:39 debian kernel:  child pid: 11653, name: a.out
Nov 26 16:09:40 debian kernel: pid: 11395, name: a.out
Nov 26 16:09:40 debian kernel:  child pid: 11653, name: a.out
Nov 26 16:09:41 debian kernel: pid: 11395, name: a.out
Nov 26 16:09:41 debian kernel:  child pid: 11653, name: a.out
Nov 26 16:09:41 debian kernel:  child pid: 11654, name: a.out
Nov 26 16:09:42 debian kernel: pid: 11395, name: a.out
Nov 26 16:09:41 debian kernel:  child pid: 11653, name: a.out
Nov 26 16:09:41 debian kernel:  child pid: 11654, name: a.out
Nov 26 16:09:42 debian kernel: pid: 11395, name: a.out
Nov 26 16:09:42 debian kernel:  child pid: 11653, name: a.out
Nov 26 16:09:42 debian kernel:  child pid: 11654, name: a.out
Nov 26 16:09:43 debian kernel: pid: 11395, name: a.out
Nov 26 16:09:43 debian kernel:  child pid: 11653, name: a.out
Nov 26 16:09:43 debian kernel:  child pid: 11654, name: a.out
Nov 26 16:09:43 debian kernel:  child pid: 11655, name: a.out
Nov 26 16:09:44 debian kernel: pid: 11395, name: a.out
Nov 26 16:09:44 debian kernel:  child pid: 11653, name: a.out
Nov 26 16:09:44 debian kernel:  child pid: 11654, name: a.out
Nov 26 16:09:44 debian kernel:  child pid: 11655, name: a.out
Nov 26 16:09:45 debian kernel: pid: 11395, name: a.out
Nov 26 16:09:45 debian kernel:  child pid: 11653, name: a.out
Nov 26 16:09:45 debian kernel:  child pid: 11654, name: a.out
Nov 26 16:09:45 debian kernel:  child pid: 11655, name: a.out
Nov 26 16:09:45 debian kernel:  child pid: 11656, name: a.out

```

...

```
Nov 26 16:10:09 debian kernel: pid: 11395, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11653, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11654, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11655, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11656, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11657, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11658, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11659, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11660, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11661, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11662, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11663, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11664, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11665, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11666, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11667, name: a.out
Nov 26 16:10:09 debian kernel:   child pid: 11668, name: a.out
root@debian:~/forking#
```

Βλέπουμε δηλαδή ότι μετά την εισαγωγή του module στο Kernel, αρχίζει το module να τυπώνει την «αναφορά» ανά τακτά χρονικά διαστήματα.

Αρχικά η main process βρίσκεται ακόμα στην αναμονή οπότε το Module δεν «βρίσκει» κανένα παιδί.

Στην συνέχεια αρχίζουν να δημιουργούνται παιδιά κάθε 2 δευτερόλεπτα (περίπου κάθε δεύτερη αναφορά) τα οποία τυπώνονται στην λίστα.

Η εκτέλεση συνεχίζεται μέχρι να δημιουργηθούν και τα 15 παιδιά και μετά εμφανίζουμε τα logs του kernel με την εντολή

```
root@debian:~/forking# journalctl --since "3 minutes ago" | grep kernel
```