ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
21/12/2024

Λειτουργικά Συστήματα

Δραστηριότητα 4: Linux και Δέσμευση Μνήμης

ΑΣΚΗΣΗ 1

1. Στο *linux / tools / lib / slab.c* υλοποιείται η kmalloc που χρησιμοποιείται για δυναμική δέσμευση συνεχόμενης μνήμης σε kernel space.
   Το πρώτο όρισμα είναι το μέγεθος της μνήμης που δεσμεύται σε bytes και το δεύτερο, ο τρόπος με τον οποίο θα γίνει η δέσμευση (GFP_KERNEL, GFP_ATOMIC, __GFP_DMA).
   Επιστρέφει έναν δείκτη με τη διεύθυνση του πρώτου στοιχείου της μνήμης που δεσμεύτηκε ή NULL σε περίπτωση αποτυχίας.

```
void *kmalloc(size_t size, gfp_t gfp)
{
        void *ret;

        if (!(gfp & __GFP_DIRECT_RECLAIM))
                return NULL;

        ret = malloc(size);
        uatomic_inc(&kmalloc_nr_allocated);
        if (kmalloc_verbose)
                printf("Allocating %p from malloc\n", ret);
        if (gfp & __GFP_ZERO)
                memset(ret, 0, size);
        return ret;
}
```

2. Στο *linux / tools / lib / slab.c* υλοποιείται η kfree που χρησιμοποιείται για απελευθέρωση της μνήμης που δεσμεύτηκε με την kmalloc .Το void *p είναι ο δείκτης με την διεύθυνση από την οποία ξεκινά η αποδέσμευση μνήμης.

```
void kfree(void *p)
{
        if (!p)
                return;
        uatomic_dec(&kmalloc_nr_allocated);
        if (kmalloc_verbose)
                printf("Freeing %p to malloc\n", p);
        free(p);
}
```

3. Στο *linux / mm / page_alloc.c* υλοποιείται η get_free_pages_noprof που χρησιμοποιείται για δέσμευση συνεχόμενων σελίδων πλήθους 2^{order}
   Επιστρέφει έναν δείκτη με τη διεύθυνση του πρώτου byte της μνήμης που δεσμεύτηκε.

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
21/12/2024

```
unsigned long get_free_pages_noprof(gfp_t gfp_mask, unsigned int order)
{
        struct page *page;

        page = alloc_pages_noprof(gfp_mask & ~__GFP_HIGHMEM, order);
        if (!page)
                return 0;
        return (unsigned long) page_address(page);
}
EXPORT_SYMBOL(get_free_pages_noprof);
```

4. Στο *linux /linux /types.h* υπάρχει η δομή atomic_t που κρατάει μία ακέραια τιμή και χρησιμοποιείται για τον συγχρονισμό πρόσβασης σε μία μεταβλήτη. Οι λειτουργίες RMW σε αυτόν τον τύπο είναι ατομικές, δεν διακόπτονται από άλλα νήματα.

```
typedef struct {
        int counter;
} atomic_t;
```

5. *Στο linux / include / linux / atomic / atomic-instrumented.h* υλοποιείται η atomic_read, η οποία διαβάζει ατομικά μία μεταβλητή τύπου atomic_t, δηλαδή δεν επηρεάζεται από λειτουργίες άλλων νημάτων που γράφουν ή τροποποιούν την ίδια μεταβλητή.

```
static __always_inline int
atomic_read(const atomic_t *v)
{
        instrument_atomic_read(v, sizeof(*v));
        return raw_atomic_read(v);
}
```

ΑΣΚΗΣΗ 2

Με δικαιώματα root δημιουργώ ένα directory memory-module και μέσα σε αυτό τα αρχεία memory-module.c και το Makefile για την μεταγλώττιση.

Ο τύπος int είναι 4 bytes, οπότε δεσμεύω με την kmalloc 1024 στοιχεία τύπου int , δηλαδή 4096 bytes. Μόλις φορτωθεί το memory-module.ko, δεσμεύονται τα 4096 bytes και τυπώνονται στα logs του συστήματος. Παρατηρούμε ότι έχουν αρχικοποιηθεί με 0. Όταν εκφορτωθεί το module, απελευθερώνεται η μνήμη με την kfree.

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
21/12/2024

```
  GNU nano 7.2                                          memory-module.c
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/slab.h>

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

int *my_memory;
static int my_init(void){
        printk("Allocate 4096 bytes\n");
        my_memory = kmalloc(1024*sizeof(int),GFP_KERNEL);
        for(int i=0;i<1024;i++){
          printk("element %d is %d", i, my_memory[i]);
          }
        return 0;
}

static void my_exit(void){
        kfree(my_memory);
        printk("Free previously allocated 4096 bytes\n");
}

module_init(my_init);
module_exit(my_exit);
```

Παρακάτω φαίνονται τα αποτελέσματα φόρτωσης-εκφόρτωσης του module:

```
root@nataliarouska:~/memory-module# insmod memory-module.ko
root@nataliarouska:~/memory-module# rmmod memory-module.ko
root@nataliarouska:~/memory-module# journalctl --since "1 minutes ago" | grep kernel
Dec 21 13:30:28 nataliarouska kernel: Allocate 4096 bytes
Dec 21 13:30:28 nataliarouska kernel: element 0 is 0
Dec 21 13:30:28 nataliarouska kernel: element 1 is 0
Dec 21 13:30:28 nataliarouska kernel: element 2 is 0
Dec 21 13:30:28 nataliarouska kernel: element 3 is 0
Dec 21 13:30:28 nataliarouska kernel: element 4 is 0
Dec 21 13:30:28 nataliarouska kernel: element 5 is 0
Dec 21 13:30:28 nataliarouska kernel: element 6 is 0
Dec 21 13:30:28 nataliarouska kernel: element 7 is 0
Dec 21 13:30:28 nataliarouska kernel: element 8 is 0
Dec 21 13:30:28 nataliarouska kernel: element 9 is 0
Dec 21 13:30:28 nataliarouska kernel: element 10 is 0
Dec 21 13:30:28 nataliarouska kernel: element 11 is 0
Dec 21 13:30:28 nataliarouska kernel: element 12 is 0
Dec 21 13:30:28 nataliarouska kernel: element 13 is 0
Dec 21 13:30:28 nataliarouska kernel: element 14 is 0
Dec 21 13:30:28 nataliarouska kernel: element 15 is 0
Dec 21 13:30:28 nataliarouska kernel: element 16 is 0
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
21/12/2024

•••

```
Dec 21 13:30:28 nataliarouska kernel: element 1017 is 0
Dec 21 13:30:28 nataliarouska kernel: element 1018 is 0
Dec 21 13:30:28 nataliarouska kernel: element 1019 is 0
Dec 21 13:30:28 nataliarouska kernel: element 1020 is 0
Dec 21 13:30:28 nataliarouska kernel: element 1021 is 0
Dec 21 13:30:28 nataliarouska kernel: element 1022 is 0
Dec 21 13:30:32 nataliarouska kernel: element 1023 is 0
Dec 21 13:30:32 nataliarouska kernel: Free previously allocated 4096 bytes
```

ΑΣΚΗΣΗ 3

Η διεργασία threads.c δημιουργεί 4 threads και επιστρέφει το pid του κύριου νήματος, το οποίο περνάμε σαν παράμετρο όταν φορτώνουμε το module process-mm-module.ko

Στην συνάρτηση print_process_info , εισάγουμε την παρακάτω γραμμή κώδικα που τυπώνει τον αριθμό των threads που χρησιμοποιούν κοινή μνήμη.

```
printk("Users of the shared memory: %d\n",atomic_read(&task->mm->mm_users));
```

```
 GNU nano 7.2                                                process-mm-module.c *
#include "asm/processor.h"
#include "linux/list.h"
#include "linux/nodemask.h"
#include "linux/printk.h"
#include "linux/sched.h"
#include <linux/kernel.h>
#include <linux/module.h>

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/5 // 200ms approximately
static int PID;

/* module parameters for setting the PID */
module_param(PID, int, S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP); //PID parameter type of (int) with permissions for read write access
MODULE_PARM_DESC(PID, "The process PID"); //parameter description

static void print_process_info(struct timer_list *unused)
{
    struct task_struct *task;
    struct task_struct *child;

    /* 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);

    if (task)
    {
        /* TODO: print the number of processes accesing the process' memory. */
        printk("Found pid: %d, name: %s\n", task->pid, task->comm);
        printk("Users of the shared memory: %d\n",atomic_read(&task->mm->mm_users));
        list_for_each_entry(child, &task->thread_group, thread_group){
          if(child->mm == task->mm){
          printk("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);
}
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
21/12/2024

```c
static int my_init(void)
{

    /*
       Timers are a kernel mechanism to execute a function at a given time.
       They take as arguments a pointer to a timer_list struct, a callback function (function to be called
       when the timer expires) and some flags.
       We want to have the print_process_info function run every DELAY ticks.
       The timer is destroyed after it expires.
       That's why we need to restart the timer inside the function.
       Remember to delete the timer on the module exit function.
    */
    timer_setup(&check_timer, print_process_info, 0);
    /*
       jiffies in the variable that holds the number of ticks (timer interrupts) since the machine booted.
       We want our callback function to execute after DELAY ticks.
    */
    check_timer.expires = jiffies + DELAY;
    /* Insert the timer to the global list of active timers. */
    add_timer(&check_timer);
    return 0;
}

static void my_exit(void)
{
    /* Finally, remove the timer. */
    del_timer(&check_timer);
}

module_init(my_init);
module_exit(my_exit);
```

Στην αρχή παρατηρούμε ότι η ατομική τιμή mm_users είναι 1 (το κυρίο νήμα) και όταν δημιουργηθούν και τα υπόλοιπα 4 νήματα γίνεται 5. Επίσης η συνάρτηση print_process_info καλείται ανά DELAY, οπότε για αυτό τυπώνονται τα ίδια στοιχεία ανα τακτά χρονικά διαστήματα μέχρι να εκφορτωθεί το module.

```
root@nataliarouska:/media/sf_shared_folder_vm# gcc threads.c -o thread
root@nataliarouska:/media/sf_shared_folder_vm# ./thread
PID: 7266
```

ΝΑΤΑΛΙΑ ΡΟΥΣΚΑ
1092581
21/12/2024

```
root@nataliarouska:~/process-mm-module# insmod process-mm-module.ko PID=7266
root@nataliarouska:~/process-mm-module# rmmod process-mm-module.ko
root@nataliarouska:~/process-mm-module# journalctl --since "1 minutes ago" |grep kernel
Dec 22 13:56:05 nataliarouska kernel: Found pid: 7266, name: thread
Dec 22 13:56:05 nataliarouska kernel: Users of the shared memory: 1
Dec 22 13:56:05 nataliarouska kernel: Found pid: 7266, name: thread
Dec 22 13:56:05 nataliarouska kernel: Users of the shared memory: 1
Dec 22 13:56:06 nataliarouska kernel: Found pid: 7266, name: thread
Dec 22 13:56:06 nataliarouska kernel: Users of the shared memory: 1
Dec 22 13:56:06 nataliarouska kernel: Found pid: 7266, name: thread
Dec 22 13:56:06 nataliarouska kernel: Users of the shared memory: 5
Dec 22 13:56:06 nataliarouska kernel: pid: 7269, name: thread
Dec 22 13:56:06 nataliarouska kernel: pid: 7270, name: thread
Dec 22 13:56:06 nataliarouska kernel: pid: 7271, name: thread
Dec 22 13:56:06 nataliarouska kernel: pid: 7272, name: thread
Dec 22 13:56:06 nataliarouska kernel: Found pid: 7266, name: thread
Dec 22 13:56:06 nataliarouska kernel: Users of the shared memory: 5
Dec 22 13:56:06 nataliarouska kernel: pid: 7269, name: thread
Dec 22 13:56:06 nataliarouska kernel: pid: 7270, name: thread
Dec 22 13:56:06 nataliarouska kernel: pid: 7271, name: thread
Dec 22 13:56:06 nataliarouska kernel: pid: 7272, name: thread
Dec 22 13:56:06 nataliarouska kernel: Found pid: 7266, name: thread
Dec 22 13:56:06 nataliarouska kernel: Users of the shared memory: 5
Dec 22 13:56:06 nataliarouska kernel: pid: 7269, name: thread
Dec 22 13:56:06 nataliarouska kernel: pid: 7270, name: thread
Dec 22 13:56:06 nataliarouska kernel: pid: 7271, name: thread
Dec 22 13:56:06 nataliarouska kernel: pid: 7272, name: thread
Dec 22 13:56:06 nataliarouska kernel: Found pid: 7266, name: thread
Dec 22 13:56:06 nataliarouska kernel: Users of the shared memory: 5
Dec 22 13:56:06 nataliarouska kernel: pid: 7269, name: thread
Dec 22 13:56:06 nataliarouska kernel: pid: 7270, name: thread
Dec 22 13:56:06 nataliarouska kernel: pid: 7271, name: thread
Dec 22 13:56:06 nataliarouska kernel: pid: 7272, name: thread
root@nataliarouska:~/process-mm-module#
```