Echahid Hamma Lakhdar University - El Oued
Exact Sciences Faculty
Computer Sciences Department

Internet of Things & Cybersecurity– 3rd*Sem* — **Date:** 06/01/2025
Architectures of advanced OS — **Time limit:** $01h30min$

Exercice 01 (05 Pts)

Indicate True or False for the following statements. For each incorrect statement, provide the correction

1. **Memory protection ensures that one process cannot access the memory space of another process in a multitasking system.**
2. **In a system that uses paging, the page table maps virtual addresses directly to physical addresses.**
3. **A process in a running state can be moved to a suspended state without being terminated.**
4. **In a multi-core system, the operating system can assign a single task to multiple cores simultaneously without creating synchronization issues.**
5. **Interrupt handling in modern operating systems ensures that high-priority interrupts are serviced before lower-priority ones, even if they occur simultaneously.**

Exercice 02 (15 Pts)

The subsequent questions reference resources available in the Annex at the end of this document. Please refer to them as needed to answer these questions.

1. **What is the purpose of the** `module_init` **and** `module_exit` **macros in the above code?**
2. **What is the role of the** `MODULE_LICENSE` **macro, and why is it important in kernel programming?**

3. **How can you verify that the `printk` messages appear in the kernel log?**
4. **What does the `insmod` command do, and why does it require `sudo` privileges?**
5. **How can you check if a module is currently loaded into the kernel?**
6. **Explain the purpose of `dmesg` in kernel programming.**
7. **Why does the kernel module use `msleep()` during node creation? How does this simulate real-world behavior?**
8. **What is the significance of storing the `jiffies` value in each node? How can this information be useful in debugging or performance monitoring?**
9. **How does `list_for_each_entry_safe` prevent crashes when deleting nodes from the list?**
10. **Modify the module to calculate and log the total time elapsed (in seconds) from the creation of the first node to the deletion of the last node. How would you implement this?**
11. **If a memory leak occurs in the module, how can you identify and resolve it? Suggest debugging techniques.**

ANNEX
Advanced Linked List Kernel Module

1 Code Implementation

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/slab.h>
#include <linux/list.h>
#include <linux/jiffies.h>
#include <linux/delay.h>

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Student Name");
MODULE_DESCRIPTION("Advanced Linked List Example");
MODULE_VERSION("1.0");

struct my_node {
    int data;
    unsigned long timestamp;  // Store jiffies at node creation
    struct list_head list;
};

static LIST_HEAD(my_list);

static int __init list_init(void) {
    struct my_node *new_node;
    int i;

    for (i = 1; i <= 10; i++) {  // Create 10 nodes
        new_node = kmalloc(sizeof(*new_node), GFP_KERNEL);
        if (!new_node) {
            printk(KERN_ERR "Memory allocation failed\n");
            return -ENOMEM;
        }

        new_node->data = i * 10;
        new_node->timestamp = jiffies;  // Capture creation timestamp
        list_add_tail(&new_node->list, &my_list);
        printk(KERN_INFO "Node %d created with data: %d at time: %lu\n"
            ↪ ,
               i, new_node->data, new_node->timestamp);

        msleep(100);  // Simulate processing delay
    }

    return 0;
}

static void __exit list_exit(void) {
    struct my_node *node, *tmp;

    list_for_each_entry_safe(node, tmp, &my_list, list) {
        printk(KERN_INFO "Deleting node with data: %d created at time:
           ↪ %lu\n",
               node->data, node->timestamp);
        list_del(&node->list);
        kfree(node);
    }

    printk(KERN_INFO "All nodes deleted, list cleaned up\n");
}

module_init(list_init);
module_exit(list_exit);
```

Listing 1: Advanced Linked List Kernel Module

2 Makefile

```
obj-m += advanced_list.o

all:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
    make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean
```

Listing 2: Makefile for Advanced Linked List Module

3 Kernel Commands for Testing

```
    make
    sudo insmod advanced_list.ko
    sudo rmmod advanced_list
    dmesg | tail
```

Echahid Hamma Lakhdar University - El Oued
Exact Sciences Faculty
Computer Sciences Department
–Exam Correction –

Internet of Things & Cybersecurity– 3^{rd}Sem **Date:** 06/01/2025
Architectures of advanced OS **Time limit:** $01h30min$

Exercice 01 (05 Pts)

Indicate True or False for the following statements. For each incorrect statement, provide the correction.

1. **Memory protection ensures that one process cannot access the memory space of another process in a multitasking system.**

   –Answer: True. Memory protection ensures isolation between processes by preventing unauthorized access to their memory spaces.

2. **In a system that uses paging, the page table maps virtual addresses directly to physical addresses.**

   –Answer: False. In a paging system, the page table maps virtual addresses to page frames, which are then mapped to physical addresses through the MMU (Memory Management Unit).

3. **A process in a running state can be moved to a suspended state without being terminated.**

   –Answer: True. A running process can be suspended, typically to allow other processes to execute, without being terminated.

4. **In a multi-core system, the operating system can assign a single task to multiple cores simultaneously without creating synchronization issues.**

–Answer: False. When assigning a single task to multiple cores, synchronization issues can arise, particularly when multiple threads or processes are accessing shared resources. Proper synchronization mechanisms, such as mutexes, are needed to avoid conflicts.

5. **Interrupt handling in modern operating systems ensures that high-priority interrupts are serviced before lower-priority ones, even if they occur simultaneously.**

   –Answer: True. Interrupts are typically prioritized, with higher-priority interrupts being serviced before lower-priority ones, ensuring timely handling of critical tasks.

Exercice 02 (15 Pts)

1. **What is the purpose of the `module_init` and `module_exit` macros in the above code?**

   –Answer: The `module_init` macro specifies the function to be executed when the module is loaded into the kernel, initializing resources or data structures. The `module_exit` macro specifies the cleanup function to be executed when the module is unloaded, releasing allocated resources and preventing memory leaks.

2. **What is the role of the `MODULE_LICENSE` macro, and why is it important in kernel programming?**

   –Answer: The `MODULE_LICENSE` macro declares the licensing terms of the module. It ensures compatibility with the kernel's GPL license. If not defined or incompatible, the kernel may restrict certain functionalities and issue warnings.

3. **How can you verify that the `printk` messages appear in the kernel log?**

   –Answer: Use the `dmesg` command to view the kernel log messages. For example: [language=bash] dmesg — tail

4. **What does the `insmod` command do, and why does it require `sudo` privileges?**

   –Answer: The `insmod` command inserts a module into the kernel, requiring elevated privileges because it modifies the kernel's runtime environment, which is critical for system stability.

5. **How can you check if a module is currently loaded into the kernel?**

   –Answer: Use the `lsmod` command to list all currently loaded modules: [language=bash] lsmod — grep advanced$_list$

6. **Explain the purpose of `dmesg` in kernel programming.**

–Answer: `dmesg` retrieves messages from the kernel ring buffer, useful for debugging and monitoring kernel module activities.

7. **Why does the kernel module use `msleep()` during node creation? How does this simulate real-world behavior?**

   –Answer: `msleep()` introduces a delay, simulating real-world scenarios where processing tasks might take time (e.g., interacting with hardware). This highlights the behavior of the kernel in a realistic multitasking environment.

8. **What is the significance of storing the `jiffies` value in each node? How can this information be useful in debugging or performance monitoring?**

   –Answer: The `jiffies` value records the system's tick count at node creation, providing timestamps for events. It helps in debugging timing-related issues and monitoring module performance.

9. **How does `list_for_each_entry_safe` prevent crashes when deleting nodes from the list?**

   –Answer: `list_for_each_entry_safe` uses a temporary pointer to store the next node before deleting the current node. This avoids dereferencing invalid memory after a node's deletion.

10. **Modify the module to calculate and log the total time elapsed (in seconds) from the creation of the first node to the deletion of the last node. How would you implement this?**

    –Answer: Store the `jiffies` value at the first node's creation and again at the deletion of the last node. Calculate the elapsed time using the formula:

$$\text{elapsed\_time (seconds)} = \frac{\text{end\_jiffies} - \text{start\_jiffies}}{\text{HZ}} \tag{1}$$

    Add code in `list_init` to store `start_jiffies` and in `list_exit` to compute and log the elapsed time.

11. **If a memory leak occurs in the module, how can you identify and resolve it? Suggest debugging techniques.**

    –Answer: Identify memory leaks using tools like `kmemleak` or by monitoring kernel logs for allocation warnings. Ensure all allocated memory is freed in `module_exit`. Add extensive `printk` logs to trace allocations and deallocations.