2. Demonstrate how to avoid the "zombie" state by properly waiting for child processes to exit in a parent process.

#include <stdio.h>

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

main()

{

    int pid,dip,cpid;

    pid=fork();

    if(pid==0)

    {

        printf("1st childs process id is %d\n",getpid());

        printf("first-child terminating from memory\n");

    }

    else

    {

        dip=fork();

        if(dip==0)

        {

            printf("2nd childs process id is %d\n",getpid());

            printf("second-child terminating\n");

        }

        else

        {

            cpid=wait(0);

            printf("Child with pid %d died\n",cpid);

            cpid=wait(0);

            printf("Child with pid %d died\n",cpid);

            printf("I am parent \n");

        }

    }

}

9. Create an LKM that prints information about a specific process, including its PID, resident set size (RSS), virtual memory size (VSZ), and command name.

#include <linux/init.h>

#include <linux/module.h>

#include <linux/sched.h>

#include<linux/signal.h>

#include<linux/pid\_namespace.h>

#include<linux/cdev.h>

#include<linux/proc\_fs.h>

#include<linux/slab.h>

static int \_\_init memory\_usage\_init(void) {

    struct task\_struct \*task;

    printk(KERN\_INFO "Process Memory Usage:\n");

    printk(KERN\_INFO "PID\tRSS (bytes)\tVSZ (bytes)\tCMD\n");

    for\_each\_process(task) {

        struct mm\_struct \*mm = get\_task\_mm(task);

        if (mm != NULL) {

            unsigned long rss = get\_mm\_rss(mm);

            unsigned long vsz = mm->total\_vm << (PAGE\_SHIFT - 10); // Convert pages to bytes

            printk(KERN\_INFO "%d\t%lu\t%lu\t%s\n", task->pid, rss, vsz, task->comm);

            mmput(mm);

        }

    }

    return 0;

}

static void \_\_exit memory\_usage\_exit(void) {

    printk(KERN\_INFO "Memory usage module unloaded\n");

}

module\_init(memory\_usage\_init);

module\_exit(memory\_usage\_exit);

MODULE\_LICENSE("GPL");

MODULE\_DESCRIPTION("Memory Usage Kernel Module");

MODULE\_AUTHOR("Your Name");

10. Create an LKM using kmalloc for allocating memory for the struct process\_info objects that initializes, tracks, and prints information about running processes.

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/proc\_fs.h>

#include <linux/sched.h>

#include <linux/uaccess.h>

#define MAX\_PROCESS\_COUNT 100 // Maximum number of tracked processes

#define MAX\_PROCESS\_NAME\_LENGTH 256 // Maximum process name length

struct process\_info {

    pid\_t pid;

    char name[MAX\_PROCESS\_NAME\_LENGTH];

};

static struct process\_info \*process\_info\_data[MAX\_PROCESS\_COUNT];

static int process\_info\_count = 0;

static int \_\_init task\_details\_lkm\_init(void) {

  struct task\_struct \*task;

      printk(KERN\_INFO "Initializing LKM for kernel status...\n");

        for\_each\_process(task) {

     if (process\_info\_count < MAX\_PROCESS\_COUNT) {

            struct process\_info \*info = kmalloc(sizeof(struct process\_info), GFP\_KERNEL);

            if (info) {

                info->pid = task->pid;

                snprintf(info->name, MAX\_PROCESS\_NAME\_LENGTH, "%s", task->comm);

                process\_info\_data[process\_info\_count++] = info;

            }

        }

    }

    // Print process information

    printk(KERN\_INFO "LKM initialized: Tracking running processes and their names\n");

    printk(KERN\_INFO "Number of running processes: %d\n", process\_info\_count);

    for (int i = 0; i < process\_info\_count; i++) {

        printk(KERN\_INFO "Process %d: %s (PID %d)\n", i + 1, process\_info\_data[i]->name, process\_info\_data[i]->pid);

    }

    return 0; // Module loaded successfully

}

static void \_\_exit task\_details\_lkm\_exit(void) {

      printk(KERN\_INFO "Exiting LKM...\n");

    // Clean up allocated process info data

    for (int i = 0; i < process\_info\_count; i++) {

        kfree(process\_info\_data[i]);

    }

    printk(KERN\_INFO "LKM exited\n");

}

module\_init(task\_details\_lkm\_init);

module\_exit(task\_details\_lkm\_exit);

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("Your Name");

MODULE\_DESCRIPTION("LKM for printing task details in a proc file");

11. Create an LKM that allocates memory using both kmalloc and vmalloc, and then compare their characteristics.

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/proc\_fs.h>

#include <linux/sched.h>

#include <linux/uaccess.h>

#include <linux/vmalloc.h>

     char \*kmalloc\_ptr;

    char \*vmalloc\_ptr;

static int \_\_init task\_details\_lkm\_init(void) {

    printk(KERN\_INFO "Initializing memory allocation LKM...\n");

    // Allocate memory using kmalloc

    kmalloc\_ptr = kmalloc(1024, GFP\_KERNEL); // Allocate 1KB of memory

    if (!kmalloc\_ptr) {

        printk(KERN\_ERR "Failed to allocate memory using kmalloc\n");

        return -ENOMEM;

    }

    // Allocate memory using vmalloc

    vmalloc\_ptr = vmalloc(4096); // Allocate 4KB of memory using vmalloc

    if (!vmalloc\_ptr) {

        printk(KERN\_ERR "Failed to allocate memory using vmalloc\n");

        kfree(kmalloc\_ptr); // Release previously allocated kmalloc memory

        return -ENOMEM;

    }

    printk(KERN\_INFO "Memory allocation using kmalloc and vmalloc successful\n");

    return 0; // Module loaded successfully

}

static void \_\_exit task\_details\_lkm\_exit(void) {

    // Release the allocated memory

    kfree(kmalloc\_ptr);

    vfree(vmalloc\_ptr);

    printk(KERN\_INFO "Exiting memory allocation LKM...\n");

}

module\_init(task\_details\_lkm\_init);

module\_exit(task\_details\_lkm\_exit);

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("Your Name");

MODULE\_DESCRIPTION("LKM for printing task details in a proc file");

12. Load the LKM into the Linux kernel, observe two threads running concurrently and using a mutex for synchronization.

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/kthread.h> // Required for kernel threads

#include <linux/delay.h>  // Required for msleep

#include <linux/mutex.h>  // Required for mutex

MODULE\_LICENSE("GPL");

MODULE\_AUTHOR("Your Name");

MODULE\_DESCRIPTION("LKM with Mutex Synchronization");

static DEFINE\_MUTEX(counter\_mutex); // Define a mutex

static int shared\_counter = 0;

// Thread function that increments the shared counter

static int increment\_thread(void \*data) {

    int i;

    for (i = 0; i < 5; i++) {

        mutex\_lock(&counter\_mutex); // Acquire the mutex

        shared\_counter++; // Increment the shared counter

        printk(KERN\_INFO "Incremented: %d\n", shared\_counter);

        mutex\_unlock(&counter\_mutex); // Release the mutex

        msleep(1000); // Sleep for 1 second

    }

    return 0;

}

static int \_\_init mutex\_example\_init(void) {

    printk(KERN\_INFO "Initializing Mutex Example LKM...\n");

    // Create and start two kernel threads

    kthread\_run(increment\_thread, NULL, "thread1");

    kthread\_run(increment\_thread, NULL, "thread2");

    return 0;

}

static void \_\_exit mutex\_example\_exit(void) {

    printk(KERN\_INFO "Exiting Mutex Example LKM...\n");

}

module\_init(mutex\_example\_init);

module\_exit(mutex\_example\_exit);