Module 5 Programming Writeup

Sai Chandu Nallagopula

Kernel Version- 5.13.0-27- generic

Main Source File:

Header files include

#include <linux/init.h>

#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/list.h>

#include <linux/types.h>

#include <linux/moduleparm.h>

#include <linux/sched/signal.h>

Simple_init:

When you compile and insert a Linux kernel module into the kernel the first function to be executed is __init.This function is basically used to perform initialization before you perform the main operations like registering a device driver etc

A simple init-style program to be used as the init program in a PID namespace. The program reaps the status of its children and provides a simple shell facility for executing commands.

Simple_exit:

the exit function calls all functions registered with atexit and terminates the program. File buffers are flushed, streams are closed, and temporary files are deleted.

Module_init() :

Many parts of the kernel are well served as a module (dynamically-loadable parts of the kernel). Using the module_init() and module_exit() macros it is easy to write code without #ifdefs which can operate both as a module or built into the kernel.

The module_init() macro defines which function is to be called at module insertion time (if the file is compiled as a module), or at boot time: if the file is not compiled as a module the module_init() macro

becomes equivalent to __initcall(), which through linker magic ensures that the function is called on boot.

The function can return a negative error number to cause module loading to fail (unfortunately, this has no effect if the module is compiled into the kernel). This function is called in user context with interrupts enabled, so it can sleep.

module_exit:

This macro defines the function to be called at module removal time (or never, in the case of the file compiled into the kernel). It will only be called if the module usage count has reached zero. This function can also sleep, but cannot fail: everything must be cleaned up by the time it returns.

Note that this macro is optional: if it is not present, your module will not be removable (except for 'rmmod -f').

Task struct:

The task_struct (type, a type of structure, like a class definition) comprises one of the interfaces to the scheduler, so it is defined in sched.h Header (.h) files typically begin with #include's of sometimes many other header files.

sched.h contains many occurrances of "struct task_struct", only one of which is where that structure is defined. To find the one place where a struct type is defined, a handy tip is to use your browser's search feature (hot-key Ctrl-f) to search for say "struct task_struct {" Only in the definition is struct task_struct immediatly followed by SPACE}

List_head:

It represents head of the circular linked listimplementation in the list.h file

Functions& macros:

*for each_process

*List_for_each

*List_entry