Jacob Hreshchyshyn

module_param

Defined on line 4 in uinclude/linux/module.h

Signature:

#define module_param(name, type, perm)

Parameters:

The first parameter is the name of the variable to store the written information. The second parameter is the data type to indicate how to format the written information. The third parameter is the permissions level.

Description:

This macro is used to pass kernel module command line arguments into the c program. In the source code submission, it is used to store the desired value of a pid in an integer variable called pid. Therefore, the first parameter would be pid and the second parameter would be int. Because no permissions needed to be set for this assignment, the default permission value of 0 was used.

simple_init

Defined using module_init() macro, which is defined on line 86 in linux/module.h

Signature:

#define module_init(x)

Parameters

The macro takes a single parameter, which will act as the name of the function used to run when the module is loaded.

Description:

The simple_init function, which is defined by the macro module_init(), is an entry function that executes once a module is loaded into the kernel.

simple_exit

Defined using the module_exit() macro, which is defined on line 98 in linux/module.h

Signature:

#define module_exit(x)

Parameters:

The macro takes a single parameter, which will act as the name of the function used to run when the module is removed.

Description:

The simple_exit function, which is defined by the macro module_exit(), is an exit function that executes once a module is removed from the kernel.

printk

Defined in linux/printk.h on line 163.

Signature:

Int printk(const char *fmt, …)

Parameters:

The first parameter takes a string, followed by an unspecified number of other parameters.

Description:

Much like printf, which prints strings as standard console output, printk prints information to the kernel log. The string parameter can contain formatting control characters, also like printf, which can be used to print the values stored in variables. These values are described in the parameters following the string to be printed. Once it completes, the function returns an int, representing the number displayed.

for_each_process

Defined in linux/sched.h on line 3012.

Signature:

#define for_each_process(p)

Parameters:

It takes a single parameter p, which is a process likely contained in a doubly linked circular list.

Description:

This macro is used to specify a type of for-each loop that iterates through a list of processes in a doubly linked circular list until it reaches a process whose address matches the process stored at the head of the list.

list_for_each

Defined in linux/list.h on line 411.

Signature:

#define list_for_each(pos, head)

Parameters:

The macro has two parameters. The first is the address of the struct that is going to act as a “loop cursor” to keep track of the current position in the list that is being iterated over. The second parameter is the head of the list that is being iterated over.

Description:

This macro is used to specify a for loop that iterates over a list structure that contains various elements, usually represented as structs. This macro is used in the program to iterate over the collection of child processes associated with a specific process.

list_entry

Defined in linux/list.h on line 346

Signature:

#define list_entry(ptr, type, member)

Parameters:

This macro has three parameters. The first is the address of the list_head pointer, which might be the collection used to store a number of structs in a list. The second parameter is the type of the struct that the list is embedded in, which helps specify the struct to retrieve. The third parameter is the name of the list_head within the struct, or the name of the attribute that specifies the list containing the specified struct collection.

Description:

In short, this macro is used to access a specific struct contained in a list that is embedded as an attribute of another struct. More specifically, this macro is used to access specific child processes contained in a list, which itself is embedded in the struct task_struct.

MODULE_LICENSE

Defined in linux/module.h on line 199.

Signature:

#define MODULE_LICENSE(_license)

Parameters:

The macro has a single parameter called _license. A string can be passed in as a parameter to display the license of the module.

Description:

The macro simply defines a module license, which can be used in the context of bug reports and vetting.

MODULE_DESCRIPTION

Defined in linux/module.h on line 208

Signature:

#define MODULE_DESCRIPTION(_description)

Parameters:

The macro has a single parameter called _description. A string can be passed in as a parameter to display a description of the module.

Description:

The macro simply defines a description of the module, which can be used to inform users about what the module does.

MODULE_AUTHOR

Defined in linux/module.h on line 205.

Signature:

#define MODULE_AUTHOR(_author)

Parameters:

The macro has a single parameter called _author. A string can be passed in as a parameter to display the author of the module.

Description:

The macro simply defines an author of the module, which can be used to inform users about the author of the module. Multiple MODULE_AUTHOR() statements would be used if there exist multiple authors for a module.

KERN_INFO

Defined in linux/kern_levels.h on line 13.

Signature:

#define KERN_INFO

Parameters:

There are no parameters for this macro.

Description:

This macro is simply an informational macro that displays kernel information, usually in the context of a printk instruction.

struct task_struct

Fully defined in linux/sched.h starting at line 1486. It is used to store information about running processes.

Attributes Used:

volatile long state:

This variable is used to represent the state of the program, that is, whether the program is runnable (0), unrunnable (-1), or stopped (>0). It is some of the process information printed in the source code submission.

int prio:

This variable is used to represent the priority of the running task. It is some of the process information printed in the source code submission.

int static_prio:

This variable is used to represent the static priority of the running task. It is some of the process information printed in the source code submission.

int normal_prio:

This variable is used to represent the normal priority of the running task. It is some of the process information printed in the source code submission.

char comm[TASK_COMM_LEN]:

This variable is used to represent the name of the running task. It is some of the process information printed in the source code submission.

pid_t pid:

This variable is used to represent the process identification number. This pid is printed in the source code submission and is also used to regulate which processes have their information printed.

struct list_head children:

This variable is used to reference the collection of child processes that belong to a running process. It is used to access the head of the circular doubly linked list.

struct list_head sibling:

This variable is another way of referencing the collection of child processes that belong to a running process. It is used to access individual entries of the circular doubly linked list. These entries are individual child processes.

struct list_head

Defined at linux/types.h on line 184.

Attributes Used:

The program implicitly used the struct list_head pointer attributes next and previous in order to iterate through the circular doubly linked list of entries to access child process information.