Readme

For this question we used modules like linux/init.h and linux/kernel.h to access various functions necessary for making a module. sched/signal.h was used to gain access to the struct task_struct.

After importing all necessary modules, we set necessary metadata like License and Author using the MODULE_LICENSE() and MODULE_AUTHOR() functions.

Then we created two functions: static int __init processCounterInit and static int __exit processCounterExit. Inside the processCounterInit we create a task_struct struct to store the PCB contents of a process. The for_each_process(task_struct) macro loops through every process in the PCB and copies its content into task_struct struct. If it's a running process (inferred by __state == 0), the counter is incremented. We then print the counter.

Exit functions simply print that the module is removed.

module_init(processCounterInit) - runs the processCounterInit function when the module is loaded

module_exit(processCounterExit) - runs the processCounterExit function when the module is removed.

A Makefile was created which generates the .ko file for the C program accordingly. By using the insmod command the ko file is then linked to the kernel and the module is run. Similarly rmmod unlinks the module.