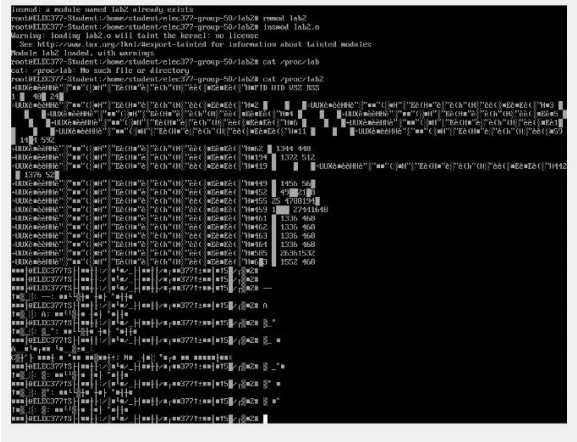
Student names: Paul Li, Mike Feng

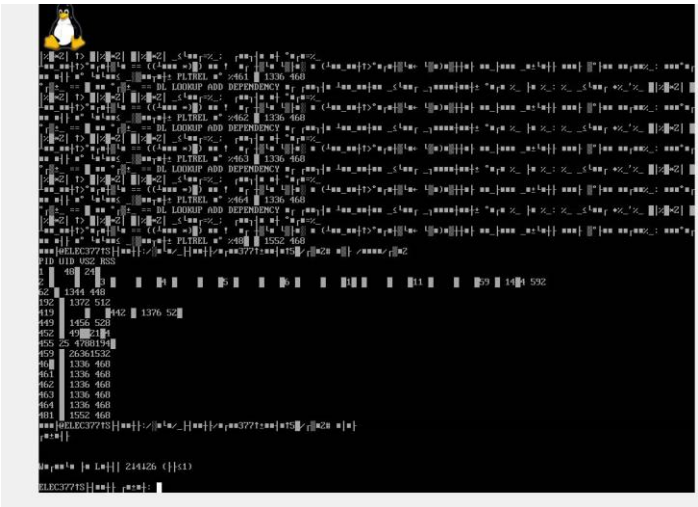
Description of the problem and how to solve them

1. After reading the discerption, we were unsure able how the init_task and next_tast works, we looked through the include <linux/sched.h> header file, and understood how those variables can form the circular linked list.

2. For the circular linked list, to check if the last element points back to the first element in the list, we use a big if loop to check if it's at first task, and if it not finish reading, instead of using a no while loop which iterates over all of the processes in my_read_proc, we used the auto calling back which is nature of kernel in Linux to prevent to write over the end of the buffer. Write over the end of the buffer could make the contents goes beyond 4k size, which could make data overflow and crash the system. So, every time we go through the task, it will build a new 4k size memory and auto call my_read_proc, which could control the overflow.



3. There is a potentially fatal bug in the first call to sprintf (for the heading PID). Unlike java, C will not auto initialize the variables, the uninitialized value will result in printing the random characters. numChars is a local variable and can have a completely random value. We depend on it being zero which is wrong. So, we adjusted that the first call to sprintf in the if and the first call to sprint in the else had changed to like the sprintf for Hello World.



4. We also made a mistake of how to use fpos in the if condition, with some further studying we found out that fpos means file position. It is the position that the kernel thinks the file is in. For example, if I want to return 40 bytes (bytes 0-39), then the kernel calls it with 40 (position 40 in the file).

Used special trick

For the big if condition, which is when the current position in the file is zero, means nothing to read yet. We would initialize needed pointers. The while loop inside the if condition move the variable theTask to point to the next valid task. For little if else condition inside the big if, the virtual memory information about the process is stored in the field mm, we firs check mm is NULL or not, if not null then make mm points to a data structure that contains the fields total_vm and rss. The last do while loop is to advance to next task. For the big else condition, we first check if theTask pointer whether reached to the end of the list, if not, print all the data for the current task. For init_module function, the if condition is to check If there was a problem it will return NULL.