Answer:

The answer provided below has been developed in a clear step by step manner.

Step 1 of 1

To implement a kernel system call as a module, we need to first define the system call and its associated function. The system call should have a unique number and a function that takes the necessary input arguments and returns the desired output. For example, we can define the system call `print_process_info` as follows:

```
#include <linux/syscalls.h>
#include <linux/types.h>
#include <linux/sched.h>

/* Define the system call number */
#define __NR_print_process_info 287

/* Define the system call function */
SYSCALL_DEFINE1(print_process_info, pid_t, pid)
{
  struct task_struct *task;

  /* Get the task struct of the given process */
  task = pid_task(find_vpid(pid), PIDTYPE_PID);

  /* Print the pid, user_id, pgid, and command path of the process */
  printk("pid: %d\n", task->pid);
  printk("user_id: %d\n", task->cred->uid.val);
  printk("pgid: %d\n", task->group_leader->pid);
  printk("command path: %s\n", task->comm);

  return 0;
}

```

Next, we need to create a Makefile to compile the module. The Makefile should specify the target module, the source files, and the compiler and linker flags. For example, the Makefile for our `print_process_info` module can be written as follows:

```
obj-m += print_process_info.o

all:
  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:
  make -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

```

To compile the module, we can run `make` in the terminal. This will generate the `print_process_info.ko` module file, which can be loaded into the kernel using the `insmod` command.
To use the system call, we need to call it from a user-space program. The system call can be called using the `syscall` function, which takes the system call number and the arguments of the system call as input. For example, to call the `print_process_info` system call for the `init` process (pid 1), we can write the following program:

```
#include <stdio.h>
#include <sys/syscall.h>

int main()
{
  /* Call the print_process_info system call for the init process */
  syscall(__NR_print_process_info, 1);

  return 0;
}

```

When the program is run, the system call will print the pid, user_id, pgid, and command path of the `init` process.
To unload the module, we can use the `rmmod` command.
The complete code for the `print_process_info` module and the user-space program can be found in the `print_process_info.c` and `print_process_info.c` files, respectively. The Makefile for the module is provided in the `Makefile` file.
Please leave a remark if you have any questions, and I will answe them without a doubt.
Thanks for keeping the questions clean and separate.
We have a very short window of opportunity to address your question.
Please comprehend and share your queries.
Please provide your honest opinion by giving the app a thumbs up.
Our efforts are evident in your thumbs up.
Please LIKE; your support means a lot to me.
I appreciate it.