OS: Project Report

Kernel module that lists all current tasks in a Linux system beginning from the init task

Nandan N
PES1UG21CS361
Sec 'F'
Roll no 41

Code:

Makefile

```
kuchangi@kuchangi:~/OS$ cat Makefile
obj-m += project.o

KERNELDIR ?= /lib/modules/$(shell uname -r)/build
PWD := $(shell pwd)

all:
	$(MAKE) -C $(KERNELDIR) M=$(PWD)
clean:
	$(MAKE) -C $(KERNELDIR) M=$(PWD) clean
```

Project: Kernel module

```
kuchangi@kuchangi:~/OS$ cat project.c
#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/sched/task.h>

void dfs(struct task_struct *task,int indent)
{
 struct task_struct *task_next;
 struct list_head *list;
 indent+=5;
 list_for_each(list, &task->children)
 {
  task_next = list_entry(list, struct task_struct, sibling);
  printk("pid: %*d | pname: %s | state: %d\n",indent,task_next->pid, task_next->comm,task_next->__state);
  dfs(task_next,indent);
 }
}

int tasks_lister_dfs_init(void)
{
 printk("Loading module...\n");
 dfs(&init_task,0);
 printk("Module loaded.\n");
 return 0;
}

void tasks_lister_dfs_exit(void)
{
 printk("Module removed.\n");
}

module_init(tasks_lister_dfs_init);
module_exit(tasks_lister_dfs_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Simple Module");
MODULE_AUTHOR("Nihaal");
```

New: To create a process

```
kuchangi@kuchangi:~/OS$ cat new.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

int main() {
    pid_t pid1, pid2;
    int status1, status2;
    printf("Parent PID = %d\n",getpid());
    // fork the first child process
    pid1 = fork();

    // check for errors in forking
    if (pid1 < 0) {
        printf("Error: Failed to fork first child process\n");
        exit(1);
    }
    else if (pid1 == 0) {
        // we are in the child process
        printf("Child PID = %d created.\n", getpid());

        // fork the child process
        pid2 = fork();

        // check for errors in forking
        if (pid2 < 0) {
            printf("Error: Failed to fork second child process\n");
            exit(1);
        }
        else if (pid2 == 0) {
            // we are in the grandchild process
            int a = 0;
            printf("Grandchild PID = %d created.\n", getpid());
            scanf("%d",&a);
        }
        else {
            // we are still in the child process
            // wait for the grandchild process to complete
            wait(&status2);
            printf("Second child process (PID %d) completed with status %d.\n", pid2, status2);
            exit(0);
        }
    }
    else {
        // we are in the parent process
        printf("Parent process (PID %d) created.\n", getpid());

        // wait for the child process to complete
        wait(&status1);
        printf("First child  process (PID %d) completed with status %d.\n", pid1, status1);
    }

    return 0;
}
```

Output:

```
kuchangi@kuchangi:~/OS$ gcc new.c
kuchangi@kuchangi:~/OS$ ./a.out
Parent PID = 3224
Parent process (PID 3224) created.
Child PID = 3225 created.
Grandchild PID = 3226 created.
```

```
[ 1861.873841] pid:         1977 | pname: xdg-desktop-por | state: 1
[ 1861.873843] pid:         2005 | pname: gjs | state: 1
[ 1861.873844] pid:         2074 | pname: gvfsd-metadata | state: 1
[ 1861.873846] pid:         2102 | pname: gnome-terminal- | state: 1
[ 1861.873848] pid:             3193 | pname: bash | state: 1
[ 1861.873849] pid:                 3202 | pname: sudo | state: 1
[ 1861.873851] pid:                     3203 | pname: sudo | state: 1
[ 1861.873853] pid:                         3204 | pname: su | state: 1
[ 1861.873855] pid:                             3205 | pname: bash | state: 1
[ 1861.873856] pid:                                 3522 | pname: insmod | state: 0
[ 1861.873858] pid:             3212 | pname: bash | state: 1
[ 1861.873860] pid:                 3224 | pname: a.out | state: 1
[ 1861.873862] pid:                     3225 | pname: a.out | state: 1
[ 1861.873863] pid:                         3226 | pname: a.out | state: 1
[ 1861.873865] pid:         2160 | pname: gsd-xsettings | state: 1
[ 1861.873867] pid:         2184 | pname: ibus-x11 | state: 1
[ 1861.873869] pid:      1459 | pname: gnome-keyring-d | state: 1
[ 1861.873871] pid:    2 | pname: kthreadd | state: 1
[ 1861.873873] pid:        3 | pname: rcu_gp | state: 1026
[ 1861.873875] pid:        4 | pname: rcu_par_gp | state: 1026
[ 1861.873876] pid:        5 | pname: slub_flushwq | state: 1026
[ 1861.873878] pid:        6 | pname: netns | state: 1026
[ 1861.873880] pid:        8 | pname: kworker/0:0H | state: 1026
[ 1861.873882] pid:       10 | pname: mm_percpu_wq | state: 1026
```

Regarding the brief explanation of your understanding of the project…..

So, a kernel project is like this really advanced computer thing where you design, build and make a kernel. And if you're like "what's a kernel?" then don't worry, it's just the super important part of a computer that makes sure everything runs smoothly. It's in charge of stuff like the CPU, memory and all those input/output devices that we use. Writing a kernel project is pretty hard and you need to know a lot about computer stuff like how they work and how to code in languages like C or assembly language.

So, when you're doing a kernel project, you start with the designing part where you figure out what the kernel needs to do and what kind of computer system it's gonna be working with. Then you move on to actually writing the code which is really technical and difficult. You gotta make sure it's efficient, reliable and super secure. You don't want any bugs or errors in there because that could make everything crash and burn.

Once you've got your code, it's time for the debugging part. This is where you figure out if there are any problems with your code and fix them before they become big issues. Debugging is important because you don't want your whole system to crash, right?

Lastly, you gotta test out your kernel to make sure it works on all kinds of different computer systems and operating systems. You need to run a bunch of tests and check all the logs and information to see if there are any problems.

So yeah, writing a kernel project is definitely not easy but if you can do it, you've basically created the backbone of an operating system. Pretty cool project overall, thank you likitha ma’am for this and also, extending the deadline.