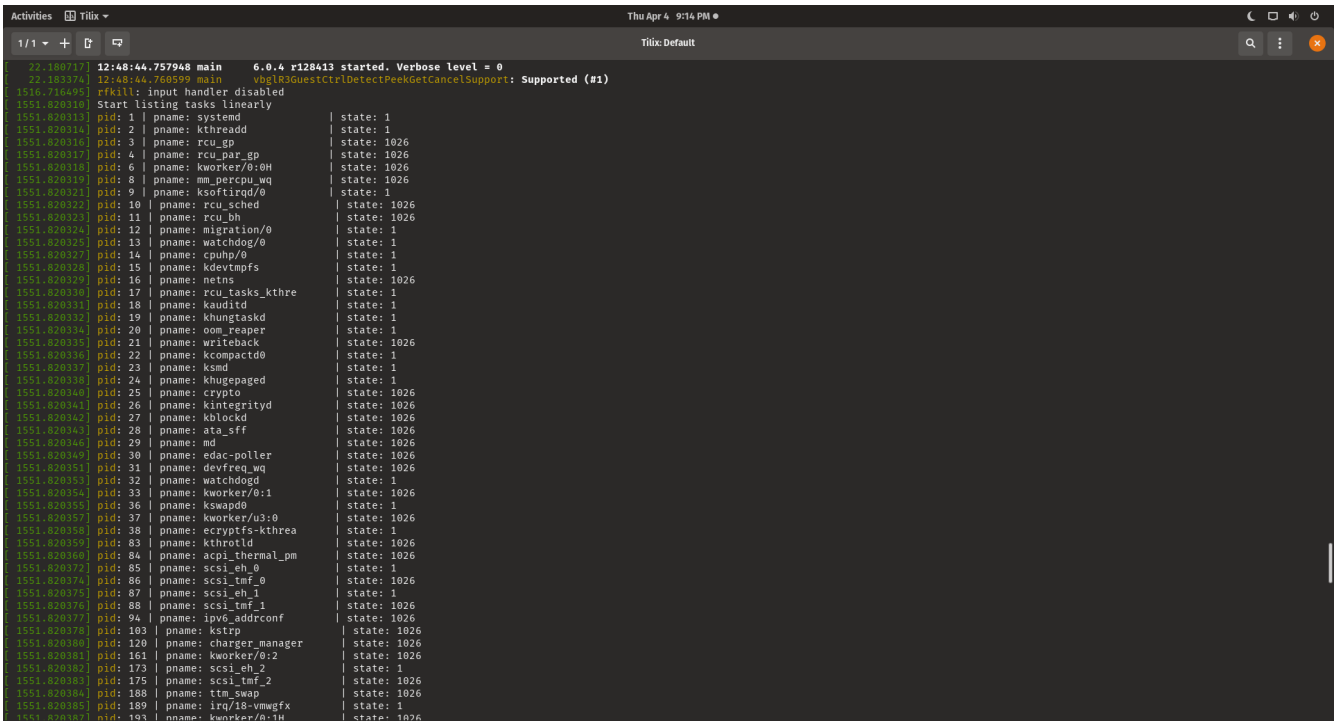
Homework 2

Name: Bijon Setyawan Raya 林湧

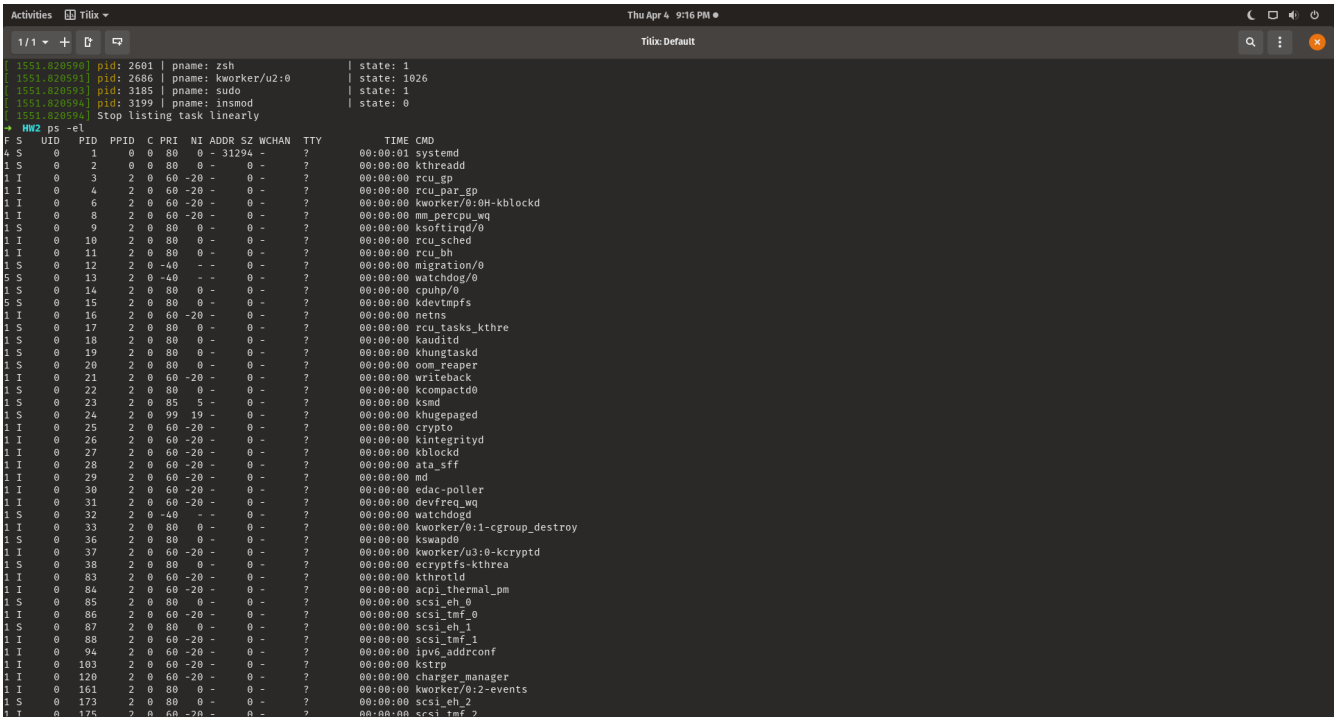
Student No. : 104062181

Listing Tasks Linearly

The result with `dmesg` command



The result with `ps -el` command



Each of the process listed in the picture above has the same PIDs and the same name, therefore my code works properly as intended.

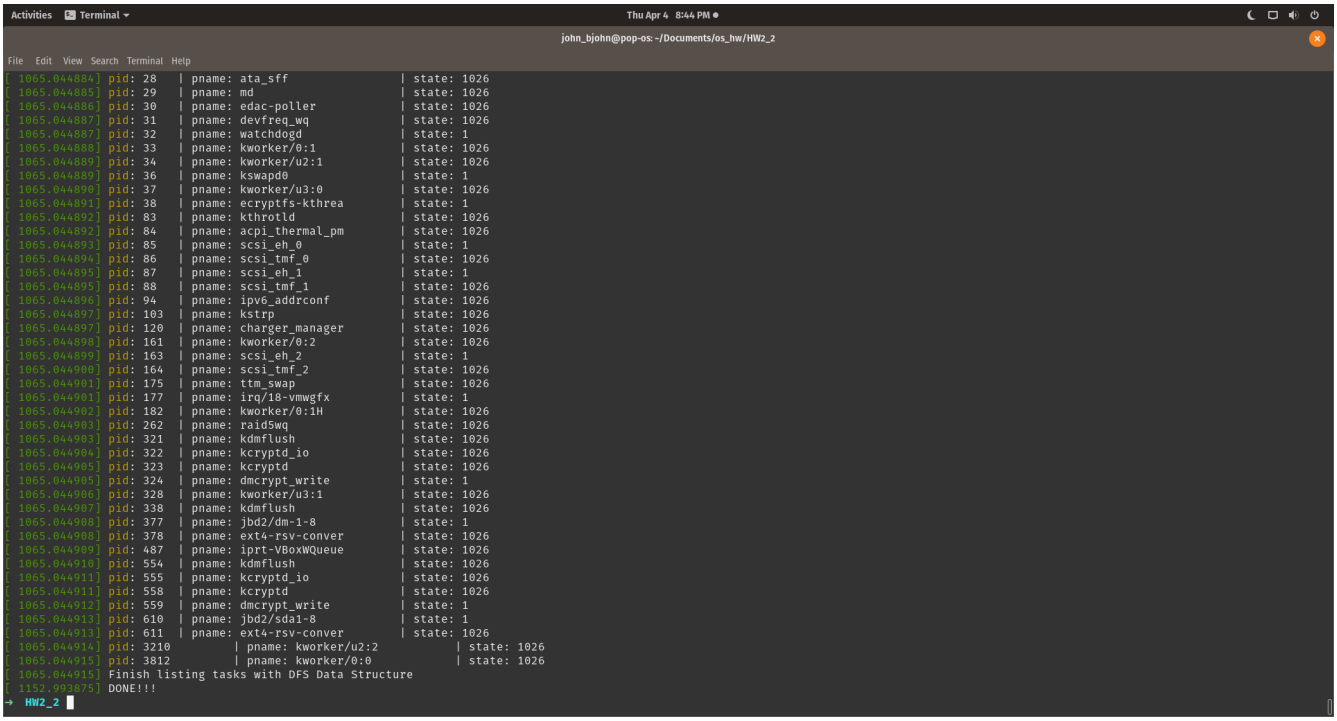
```
int start (void) {
    struct task_struct *task;

    printk(KERN_INFO "Start listing tasks linearly\n");
    for_each_process (task) {
        printk(KERN_INFO "pid: %d | pname: %-20s | state: %ld", task->pid, task->comm,
task->state);
    }
    printk(KERN_INFO "Stop listing task linearly");
    return 0;
}
```

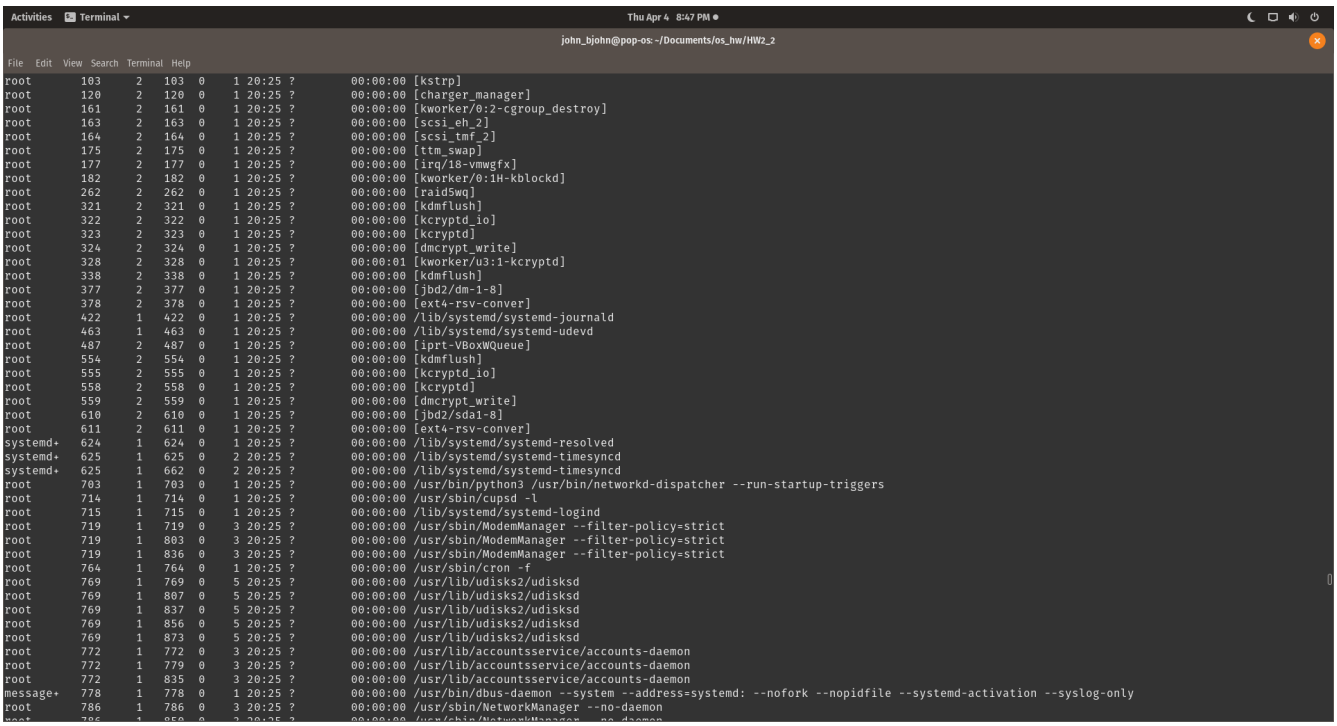
This is the main part of the code for listing task linearly. It basically creates a pointer named `task` that has all properties of a task in an operating system. Later on, I use `for_each_process()` to iterate all the tasks linearly from the first task to the n-th and print out all the id, name and the state of each task linearly.

Listing Tasks Using Depth-First-Search Data Structure

The result with `dmesg` command



The result with `ps -eFl` command



Each of the process listed in the picture above has the same PIDs and the same name, therefore my code works properly as intended.

```
void depth_first_search (struct task_struct *task) {
    struct list_head *children_list;
    struct task_struct *new_child;

    printk(KERN_INFO "pid: %d\t | pname: %-20s\t | state: %ld\n", task->pid, task->comm,
task->state);
```

```
    // loop over children_list pointer
    list_for_each (children_list, &task->children) {
        // initialize a new child that points to the next child
        new_child = list_entry(children_list, struct task_struct, sibling);

        // iterate from new_child
        depth_first_search(new_child);
    }
}
```

This is the main part of the code for listing task using Depth First Search Data Structure. It basically passing a pointer called `task()` to the function that I made called `depth_first_search()`, and print each task's properties from the root node.

After printing the root node's properties, it needs to list out both of its children's properties. Before listing them out, we need to pass `children_list` that acts a list_head pointer and `&task->children` that acts as the struct of each child in `children_list`.

After passing those parameters, I immediately initialize a new child that points to the next child and store this new child into `children_list`.

Finally, recall `depth_first_function()` again and pass `new_child` as a parameter. For the next recursive, this `new_child` acts as parent node.