Tushar Mohan
2019393

Description of your code and how you implemented the function :

- It majorly consists of 2 files namely “test.c” and “sh_task_info.c”.
- test.c
    - It is used to take input from the user and make a call to the systemcall defined by me in the sh_task_info.c file.
    - Inputs are the process id and the file name, where the output is to be saved.
    - While making a call to the systemcall defined by me, 3 arguments are passed to the very function, which includes the process id, file path and the length of the file path.
- sh_task_info.c
    - This includes a function named “SYSTEMCALL_DEFINE3” which has 4 arguments namely :
        - The name of the systemcall : sh_task_info
        - Process id (pid) of type long
        - File path (path) of type char *
        - Length of the file path (len) of type int
    - first the copy_from_user() function is called which copies the path of the file to path_buffer in the kernel space from user space, if the file exists, else it prints an error message and returns EFAULT error code, which is 14, for bad address.
    - if the file is found, the code moves on to get the process from the process_id passed to the systemcall via function named find_task_by_vpid, into the struct of type task_struct* into the variable named proces.
    - Then it checks if the task_struct proces is null or not.
    - If it is, then it returns the ESRCH error code, which is 3.
    - If the task exists, then its as well its parent’s data is printed via printk() on the Kernel SysLog.
    - Now the data printed needs to be saved in the file specified by the user input which is first saved to a char variable named buffer via sprintf.
    - The file pointer is initiated and file with the specified path is opened in write_only, create or append mode. The pointer returned by the file_open() us saved in the variable named destFile, file pointer.
    - if the file pointer is null or there is some error in opening the files, then an error message is printed in the Kernel SysLogs, and subsequently returns the ENOENT error code which corresponds to the file not found error.
    - Now kernel_write() is called to finally write the data from buffer to the destination file.
    - if kernel_write() returns a number less than 0, it means there is some error and hence -1 is returned else, file is closed by filp_close and 0 is returned.

The inputs the user should give :

- User should input a process id as integer and a file name.

Expected output (and how to interpret it) :

- A message which indicates either successful or unsuccessful termination of the systemcall.
- In the kernel SysLogs the data with respect to the process id is printed which includes:
    - Process name
    - Pid number
    - Process state
    - Priority
    - RunTime Priority
    - Static Priority
    - Normal Priority
    - Parent Death signal
    - Process Trace
    - Policy
    - Exit State
    - Exit Code
    - Exit signal
    - Personality
    - Virtual Runtime
    - Atomic flags
    - Parent process name
    - Parent process id

Error values and how to interpret them :

- EFAULT (14)
    - Which tells that the file path passed to the system call is not a valid path (or bad address)
- ESRCH (3)
    - Which tells that the process asked for by the user is not present.
- ENOENT (2)
    - Which tells that the file or directory doesn't exist
- -1
    - Which indicated that their was sum error in writing to the file specified by the user.
- 0
    - Which includes successful run of the file test.c and the systemcall defined by me on the inputs provided by the user.

References:

- https://www.halolinux.us/kernel-architecture/elements-in-the-task-structure.html