In the file arch/x86/entry/syscalls/syscall_64.tbl, this file stores all the implemented system calls.
I added a line at the end

```
548     64      sh_task_info     sys_sh_task_info
```

Here 548 is the identification number of my system call. We can invoke our system call using this number.
64 is telling this system call works for 64-bit architecture.

In the file include/linux/syscalls.h
I added a line before the last endif statement

```
asmlinkage long sys_sh_task_info(int pid, char* path);
```

This is declaration of my system call.

In my system call code,
I created a function itoa which converts integer to string so that I can append it to a character pointer using strcpy(as it takes 2 char ptrs as arguments).
Then I iterate all the current running processes and appended their fields and values in a array called str which I finally write to the file. In case any error occurs in struct file, while writing to the file or process doesn't exist I return the respective flags so that errno is set.
Important: kernel cannot access memory address in user space, we need to use a function copy_from_user to do so. Directly accessing memory from userspace will cause the os to crash/hang till eternity.

I use this errno flag in test.c file to display the specific error.
Common errors handles are -
1) Given file is a directory.
2) Process id entered does not exist.
3) Errors in opening the entered file name.
4) Errors in writing to the file.
5) Error copying data from userspace into kernel space.

In case the file doesn't exist, one is created with the same name entered.

When we run the test file.
It prompts for process id and path.
If simply the name is entered (abc.txt). Then file is created in the current working directory or if already present, details of the syscall corresponding to the pid entered is written to the file.
If we want to create the file in some other directory then we can give full path ie (/home/username/Desktop/filename.txt).