CSE 231 Operating Systems
Assignment 2
Writing Your Own System Call

Aditi Sejal
2019228

The system call sh_task_info() takes two parameters - the PID of a process and the name of the file into which the process details are written. It prints out all the fields corresponding to it on the kernel log and also saves it in a file with the given name.

Description

The code starts with checking for valid values of PID and then returns an appropriate error number if invalid PID is received. This is followed by reading the file name passed as parameter to the function using the strncpy_from_user function which takes two strings (one to which data is copied and the latter from which data is copied) and the size of the former string. This is done because we cannot directly read data from a string defined in user space from the kernel space. The data is read and copied into another character array and if this fails, an appropriate error number is returned by the system call.

A PID struct corresponding to the integer PID passed as parameter is obtained using the find_get_pid function which takes the PID (integer or pid_t type) as input. This PID struct is used to get the corresponding task_struct which is a structure, an interface to the scheduler, used to store details of processes in the linux kernel. This is done using the pid_task function which takes the pointer to the PID structure and returns the corresponding task_struct for the process with the given pid. In case no process with given pid exists, an error number is returned accordingly.

The task_struct has fields which describe the process and these are accessed and printed. Some of the fields printed are:

1. **Process Name**
   This is stored in the comm field of task_struct and is the name of the process with.
2. **Process PID**
   This is an unique identifier for active processes used by the linux kernel. It is of pid_t type which is essentially an integer and is stored in the pid field.
3. **Process State**

The state of a process stored in the state field is defined using preprocessor macros which have meanings as below:

```
#define TASK_RUNNING          0
#define TASK_INTERRUPTIBLE    1
#define TASK_UNINTERRUPTIBLE  2
#define TASK_ZOMBIE           4
#define TASK_STOPPED          8
```

4. **Priority**
   Stored in the prio field, priority of a process denotes the dynamic priority.
5. **Static Priority**
   Stored in the static_prio field, priority of a process denotes the static priority which is the priority which was allotted to the process when it was started.
6. **Normal Priority**
   Stored in the normal_prio field, priority of a process denotes the dynamic priority.
7. **Policy**
   Stored in the policy field, it holds the scheduling policy applied to the process. Linux supports five possible values**:** sched_normal, sched_batch, sched_idle, sched_rr, sched_fifo
8. **Virtual Runtime**
   This is stored in the se field of task_struct. It is present as vruntime, a field in the sched_entity type struct. It is the time in nanoseconds spent by a process on the processor for the CFS scheduler.
9. **PIDs of Children Processes**
   Some processes have child processes and these are stored as a list in the task_struct and can be obtained using the list_entry function on the list_head for all the entries.

The printk function is used to print these to the kernel log. These details for the process are also stored in a file which is opened using the filp_open function in write only and created if it does not already exist with mode 0644 and returns the file pointer. Data is written to the file using the kernel_write function which takes the file pointer, string to be written and its size, pointer within file as arguments. This is followed by closing the file using filp_close by sending a file pointer as argument and ending the definition of the syscall.

Inputs

The system call takes two arguments, a PID of type integer and the filename of the file to which the data has to be written. If this is an existing file, it is truncated and data is

then overwritten. In case it does not exist, a new file of the name is created. The PID should be valid and of a running process for correct output otherwise error is reported.

Outputs

The system call returns 0 on successful execution and otherwise returns the error number for occurred error and to which the global variable errno from is also set. It prints the details mentioned above on the kernel log which can be seen using dmesg or dmesg|tail command in linux. It also writes these details to the file in the <Field>: <Field Value> format. The interpretations of these fields can be seen as from the description of the fields above. The interpretations of the error number as described below.

Error Handling

The following errors are handled by the system call sh_task_info()

1. **Error 3**
   ESRCH 3 /* No such process */
   This means that the process with given pid does not exist and thus its corresponding task_struct can not be found by the kernel. Give PID of an existing process to resolve the error.
2. **Error 14**
   EFAULT 14 /* Bad address */
   If there is insufficient memory the kernel cannot copy data from the user space to read using the strncpy_from_user function and thus this error occurs.
3. **Error 22**
   EINVAL 22 /* Invalid argument */
   All PIDs for processes are greater than 0 and have upper limit as pid_max value which is used as a safe bound in the syscall implementation. If this error occurs provide a value between o and 131072 to resolve it.
4. **Error 21**
   EISDIR 21 /* Is a directory */
   This is encountered when an invalid filename like “.” or “..” is given which is recognised as a directory and thus the syscall does not execute properly.
5. **Error 13**
   EACCES 13 /* Permission denied */
   This occurs when we supply the root directory as location for file. The kernel cannot access the root directory generally. To resolve this error provide another valid path for the file.

The IS_ERR macro is used to check if the file pointer obtained after trying filp_open should be considered an error value and above errors are then handled accordingly.

Sample Outputs