CS342: Operating Systems Lab

Department of Computer Science and Engineering,
Indian Institute of Technology, Guwahati
North Guwahati, Assam 781 039

Exercise UP-03

OS Lessons: System calls for file operations

Rating: Moderate

Last update: 20 February 2017

Implementing PintOS code related to the operations on the file-system and files in the system is not very difficult under this project as PintOS provides the underlying implementation with a well-described interface. Students only need to make wise calls to the provided functions.

However, some students may find the exercise laborious and time consuming. The reason being the need to safeguard the kernel from the mistakes and deliberate attempts to sabotage the kernel by the user programs.

Any way a system call can harm the kernel must be denied. The sanity checks on addresses and values mentioned in the previous exercise need to be applied with enthusiasm. The memory references made by the kernel, using the user program provided arguments to the system calls, need to be tested in the system call specific manners before being allowed.

Two opposing pressures affect us. If a check is made earlier in the call, it applies to many divergent courses the system call service routines may take. On the other hand, a sanity check made at a later stage (near the actual memory reference) allows us to not prohibit arguments that may be acceptable in some circumstances.

For example, if a string is given for writing, we need to clearly establish that it starts in the user virtual address space. Do we need to ensure that the last address is also in the same virtual address space? We do not know for sure. The string may be not long enough to be a problem. Or, the number of bytes asked to be written may not be causing any problem. The data-sanity checkers are not difficult to write, but have many special cases to attend to.

Section 3.1.5 *Accessing User memory* hints at the challenges you need to overcome before you claim victory in this exercise of ensuring sanity of every argument that a user program may provide in a system call.

A simple but important decision you have to make is about the per-process open file table. Two issues of importance are its size and location. In our implementation for this project we have a table of 128 entries and is located in `struct thread`. The later projects may call for revision of this decision.

Results of `make check` Command Execution:

As this exercise was quite simple, you must make sure that your implementation has progressed to the levels matching the results below.

The only requirements from User Program PintOs project that remains unimplemented at this stage are: `exec()`, `wait()` and *Denying Writes to Executables*. These excluded functionality shall be our final exercise in project User Program. They are listed as a separate exercise because the implementation will require significant planning and efforts.

```
[vmm@progsrv build]$ make check
pass tests/userprog/args-none
pass tests/userprog/args-single
pass tests/userprog/args-multiple
pass tests/userprog/args-many
pass tests/userprog/args-dbl-space
pass tests/userprog/sc-bad-sp
pass tests/userprog/sc-bad-arg
pass tests/userprog/sc-boundary
pass tests/userprog/sc-boundary-2
pass tests/userprog/halt
pass tests/userprog/exit
pass tests/userprog/create-normal
pass tests/userprog/create-empty
pass tests/userprog/create-null
pass tests/userprog/create-bad-ptr
pass tests/userprog/create-long
pass tests/userprog/create-exists
pass tests/userprog/create-bound
pass tests/userprog/open-normal
pass tests/userprog/open-missing
pass tests/userprog/open-boundary
pass tests/userprog/open-empty
pass tests/userprog/open-null
pass tests/userprog/open-bad-ptr
pass tests/userprog/open-twice
pass tests/userprog/close-normal
pass tests/userprog/close-twice
pass tests/userprog/close-stdin
pass tests/userprog/close-stdout
pass tests/userprog/close-bad-fd
pass tests/userprog/read-normal
pass tests/userprog/read-bad-ptr
pass tests/userprog/read-boundary
pass tests/userprog/read-zero
pass tests/userprog/read-stdout
pass tests/userprog/read-bad-fd
pass tests/userprog/write-normal
pass tests/userprog/write-bad-ptr
pass tests/userprog/write-boundary
```

```
pass tests/userprog/write-zero
pass tests/userprog/write-stdin
pass tests/userprog/write-bad-fd
FAIL tests/userprog/exec-once
FAIL tests/userprog/exec-arg
FAIL tests/userprog/exec-multiple
FAIL tests/userprog/exec-missing
pass tests/userprog/exec-bad-ptr
FAIL tests/userprog/wait-simple
FAIL tests/userprog/wait-twice
FAIL tests/userprog/wait-killed
pass tests/userprog/wait-bad-pid
FAIL tests/userprog/multi-recurse
FAIL tests/userprog/multi-child-fd
FAIL tests/userprog/rox-simple
FAIL tests/userprog/rox-child
FAIL tests/userprog/rox-multichild
pass tests/userprog/bad-read
pass tests/userprog/bad-write
pass tests/userprog/bad-read2
pass tests/userprog/bad-write2
pass tests/userprog/bad-jump
pass tests/userprog/bad-jump2
FAIL tests/userprog/no-vm/multi-oom
pass tests/filesys/base/lg-create
pass tests/filesys/base/lg-full
pass tests/filesys/base/lg-random
pass tests/filesys/base/lg-seq-block
pass tests/filesys/base/lg-seq-random
pass tests/filesys/base/sm-create
pass tests/filesys/base/sm-full
pass tests/filesys/base/sm-random
pass tests/filesys/base/sm-seq-block
pass tests/filesys/base/sm-seq-random
FAIL tests/filesys/base/syn-read
pass tests/filesys/base/syn-remove
FAIL tests/filesys/base/syn-write
15 of 76 tests failed.
make: *** [check] Error 1
```

Contributing Authors:

Vishv Malhotra, Gautam Barua, Rashmi Dutta Baruah