Homework
Part 1: Virtualization – 1) Processes

master@d08e360 (20220915-171922)

P. Mainini / E. Benoist / C. Fuhrer / L. Ith

BTI1341 / Fall 2022-23

1 Introduction to Processes

1.1 Simulation

Conduct the simulation homework from *OSTEP, Chapter 4* (`process-run.py`), found at [ost]. The tasks to be performed are described at the end of the chapter PDF.

1.2 Process Structures

Compare the process structure of a recent version of the GNU/Linux kernel (`struct task_struct`) in [lin] with the *xv6 Proc Structure* presented in class ([xv6]). You will notice that it is quite different. Why? Do you find similar parts, and if so, which? Try to explain the various parts of the structure based on your current knowledge.

1.2.1 Other OSes

Try to find information about the process structures in other operating systems (e.g. Windows, macOS, ...). What do you find? Prepare a short summary (3-5 sentences) for the discussion in class.

1.3 Process Creation

Investigate how a process is created (i.e. loaded from a program) in the GNU/Linux operating system. Have a look at the *ELF executable file format* (☞ "`man readelf`").

Find out how the memory of a GNU/Linux process is organized (☞ look at `/proc/<PID>/maps`, where `<PID>` is the process ID of one of your processes). What else can you find out?

2 Limited Direct Execution

2.1 x86 System Calls in Assembler ★

In case you don't remember, read how to make system calls in assembler on x86. Which are the *trap instructions* for x86-32 and x86-64? And how are they used?

2.2 Measure System Call Duration

Create a small C program which measures the time a syscall, for instance a 0-byte `read()` from some file, takes.

☞ Use `clock_gettime()` ("`man 2 clock_gettime`") for measuring. Call the syscall multiple times and take the average to get a more precise result.

2.3 Context Switch Duration

Think about how you would write a program which measures context switch duration. What could be the difficulties?

Can you find some figures for recent systems on the internet?

3 Process API

3.1 Understanding `fork()`

Ensure that you really have understood how the `fork()` and `exec()` syscalls work. If you are not sure, you can read about it in [ADAD]. What is the output of the following program? Why?

```c
#include <stdio.h>
#include <unistd.h>

int main(void) {
  fork();
  fork();
  fork();
  printf("%d\n", 1);
  return 0;
}
```

3.2 Understanding `exec()`

A call to `exec()` replaces the currently running process with a new one (possibly from a different executable). How does this affect its PID and parent PID? What happens if the parent process exits before the child process?

3.3 Forking in Windows?

How is a new process spawned in the Windows operating system? Is the mechanism similar to `fork()` and `exec()`?

References

[ADAD] Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau, *OSTEP Chapter 5, Interlude: Process API*, `http://pages.cs.wisc.edu/~remzi/OSTEP/cpu-api.pdf`.

[lin] *Linux kernel, sched.h*, `https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git/tree/include/linux/sched.h?h=v4.19.98`.

[ost] *GitHub.com, remzi-arpacidusseau/ostep-homework*, `https://github.com/remzi-arpacidusseau/ostep-homework`.

[xv6] *GitHub.com, mit-pdos/xv6-public: xv6 OS, proc.h*, `https://github.com/mit-pdos/xv6-public/blob/master/proc.h`.