NTNU | Norwegian University of Science and Technology

Operating Systems

Lecture overview and Q&A Session 3 – 31.01.2022

Michael Engel

Lectures 3 and 4

Challenges and tasks of an operating system

- Operating system abstractions
- Tasks
- Problems
- Challenges

Processes

- Unix process hierarchy
- Shells and I/O redirection
- Unix philosophy
- Process creation and related syscalls
- Details of Unix processes

Operating system abstractions

- Processes as an abstraction – different definitions
  - "A process is a program in execution" – precise enough?
  - Process **context** in **PCB** (process control block):
    CPU state, memory, files, meta info (owner, mode, …)
- Different process models
  - multiprogramming
  - concurrent processes
  - CPU multiplexing
- Process states and transition diagram
  - Basic model: Running, Ready, Blocked
  - Which transitions are permitted and which are not?

Operating system abstractions

- CPU scheduling
  - process synchronisation
  - Inter-process communication (IPC)
- Memory hierarchy
  - from fast and small to large and slow:

    registers → cache → RAM → disk
  - Main memory management
    - Virtual memory
  - Disk management
    - disk organisation
    - file systems
    - access control

Challenges

- Modern computer architecture
  - No longer the simple machines from the 1980s…
- Multi- and manycore computers
  - How to enable load distribution and good utilisation?
- NUMA – non unified memory access
  - Main memory with different access latencies and throughputs
- Hardware virtualization
  - Share a computer between different operating systems, not only different processes
- Cloud computing
  - Use virtualization to provide highly available, adaptive services

The Unix process hierarchy

- Simple process state diagram revisited
- The Unix process hierarchy
  - process IDs
  - process inheritance: parent/child processes
  - the **init** process and its role
- Unix shells and process management (job control)
  - shell commands
  - I/O channels of processes
    - stdin, stdout, stderr
  - I/O redirection and shell pipelines

Unix philosophy

- "do one thing and do it well"
  - small programs designed to do a single task
  - *Is this still the case today?* Check the `ls` command…

```
usage: ls [-@ABCFGHILOPRSTUWabcdefghiklmnopqrstuvwxy1%,] [--color=when] [-D format] [file ...]
```

- "work together"
  - processes can use the output of other processes as input without any specific considerations
  - Pipelines avoid the creation of temporary files
- "handle text streams"
  - a simple, but universal interface for exchanging data between processes
  - *Is this good enough for today's requirements?*

Process creation and related syscalls

- The concept of system calls
  - they look like regular function calls…
- System modes: user and kernel
  - syscalls are *controlled transitions*

- Syscalls for process control
  - create, destroy, wait, terminate, check status

When Unix functionality is mentioned, the number in brackets gives the section of the man pages the functionality is described in, here section 2 (syscalls)

- Creating the process hierarchy: `fork(2)`
  - one process calls, but `fork` returns twice!
  - sharing of code and data between parent and child process
  - copy-on-write mechanism for efficiency

Details of Unix processes

- Process termination
  - `exit`, the function that never returns…
- Process termination and orphaned child processes
  - The role of the init process
- Process interaction
  - `wait` syscalls
  - zombie processes
- Executing another program: the `exec` family
  - Different variants for different use cases
  - All use the same `exec` syscall, variants handled by libc
- Discussion about combining `fork` and `exec` syscalls
- More realistic Unix process state diagram

Further organisation

- First theoretical exercise
  - Handout: last Friday (28.1.)
  - Submission deadline: this Friday (4.2.)

- First practical exercise
  - Mandatory
  - Handout: today (31.1.)
  - Submission deadline: + 3 weeks: 21.2.

- Hints and questions for the first practical exercise also in the live session next Monday

Overview Theoretical Exercise 1

1.1 Unix processes and the shell

- Revisiting the init process and its role
- Man page reading – finding relevant information
- Shell command behaviour

1.2 fork

- Unrestricted resource access can be problematic
- Behaviour of `fork` and its return values

1.3 Process execution ~~order~~

- Parent-child process relation

Overview Practical Exercise 1

- Build an alarm clock!
  - …using the Unix system calls you know from the lectures, e.g. fork, exec, wait…
- Learn about time in Unix
  - It's not quite simple
- Write a complex and (somewhat) useful C program for Unix
  - Do it in some smaller steps

- This is a complex task – you have three weeks time (and are supposed to work in a team), so please use the time!

The forum question

- We are currently discussing setting up a Discourse server
  - open source solution (https://github.com/discourse/discourse)
  - the maths department seems to use it…

- More on this later this week – sorry for the further delay