

# Vorige week

- Organisatie computersystemen.
- Een besturingsstelsel omvat in het algemeen de kernel + systeemprogramma's.
  - Hoofdtaken: abstractie/afscherming & resource allocation.
- Storage hierarchy & caching.
- Dual-mode operation.
- System calls.
- Structuren: monolithisch, microkernel, hybride kernel.

# Boekdelen

- Vorige week: deel 1: introductie & structuur.
- In de rest van de hoorcolleges behandelen we nog 4 delen:
  - Processes.
  - Memory Management.
  - File Storage.

# Process state

- Demo: top, activity monitor.

# Linux "pstree"

```
$ pstree
init--acpid
  |-auditd---{auditd}
  |-automount---4*[{automount}]
  |-avahi-daemon
  |-console-kit-dae---64*[{console-kit-da}]
  |-cron
  |-cupsd
  ...
  |-rpc.statd
  |-rpcbind
  |-rsyslogd---4*[{rsyslogd}]
  |-screen---2*[tcsh]
  |-ssh-agent
  |-sshd--+-2*[sshd---sshd---bash]
  |   |   |-sshd---sshd---tcsh---less
  |   |   |-sshd---sshd---tcsh---telnet
  |   |   |-sshd---sshd---bash---pstree
  |   |   `--sshd---sshd
  |-udev---2*[udev]
  |-udisks-daemon--+-udisks-daemon
  |   |   `--{udisks-daemon}
  |-upowerd---{upowerd}
  `--ypbind---2*[{ypbind}]
```

# fork() system call

- fork() creates a new process.
  - “The child process is an exact copy of the calling process.”
  - “Except for process ID, parent process ID”.
- Return value of fork():
  - $< 0$ : operation failed.
  - $= 0$ : returned to the child process.
  - $> 0$ : returned to the parent process, indicates process ID of child.

# exec() system call

- exec(): “replace the process image”.
  - Text, data segment, stack, heap.
  - File descriptor state not modified!
- So, for instance, load the program “/bin/ls” in memory.

# Remote Procedure Calls

- Easy way to do IPC.
- Instead of local procedure call, call a function on a different machine.
- Transfer of function arguments, return value all handled for you.
- Structured messages, structure already defined.
- Also frequently used to implement “web services”: XML-RPC, SOAP, JSON-RPC.

# Chapter 4: Threading





# Vorige week

- Proces vs. programma: actief vs. passief.
- Context switch.
- `fork()`: maakt duplicaat van gehele proces.
- `fork()`, `exec()`, `wait()`, `exit()`, `pipe()`.

# Threading

- Van oudsher heeft een proces 1 instructiestroom.
  - We slaan ook maar 1 set registers, 1 program counter, 1 stack op.
- We zeggen: het proces is “single-threaded”.
- Als een proces een blocking system call doet om bijvoorbeeld te wachten op I/O, dan blokkeert het proces.
  - GUI programma dat van de disk leest of een netwerkverbinding opzet.

# Testing & Debugging

- Popular tweet under programmers:
  - “A programmer had a problem. He thought to himself, "I know, I'll solve it with threads!". has Now problems. two he”

# HyperThreading

- One CPU core is exposed to the Operating System as two CPU cores.
- If one process gets stuck, e.g.:
  - It needs to use a functional unit that is occupied.
  - Blocks on a memory operation.then the other process takes over.
- Fully transparent to the Operating System and end-user.

# Many-to-one (2)

- One can wonder whether this model ever had advantages.
  - Consider that in the past, kernels did not support threads.
  - Additionally, multi-core/CPU systems were not widespread.
- On some systems, user-space threads switch faster than kernel-space threads.
  - Another argument: you can control the scheduling of threads yourself.
- Blocking I/O problem can (mostly) be solved by non-blocking I/O.

# Many-to-one (3)

- Threads sometimes preferred over event-based programming, allows more natural programming.
  - (Though subject of debate in the past).
- Java Green Threads were used to simulate a multi-threaded system.
  - Through an alternative I/O API, the problematic blocking I/O calls can be hidden from the user by automatically using asynchronous I/O instead.
- Do note that pure many-to-one implementations have for the most part been superseded, because with this model no advantage can be taken of multi-core systems.

# GCD example

## BEFORE

```
for (i = 0; i < count; i++) {
    results[i] = do_work(data, i);
}
total = summarize(results, count);
```

## AFTER

```
dispatch_apply(count, dispatch_get_global_queue(0, 0),
    ^(size_t i){
        results[i] = do_work(data, i);
    });
total = summarize(results, count);
```

Source: [https://en.wikipedia.org/wiki/Grand\\_Central\\_Dispatch](https://en.wikipedia.org/wiki/Grand_Central_Dispatch)

# Example signals

- SIGSEGV – Segmentation violation
- SIGBUS – Bus Error
- SIGPIPE – Broken Pipe
- SIGFPE – Floating Point Exception
- SIGTERM – Terminate (default of kill command)
- SIGKILL – Non-ignorable kill
- SIGALRM – Alarm Clock
- See also “kill -l”