第 5 讲：The Interface of OS

第二节：Overview of POSIX

陈渝

清华大学计算机系

yuchen@tsinghua.edu.cn

2020 年 3 月 15 日

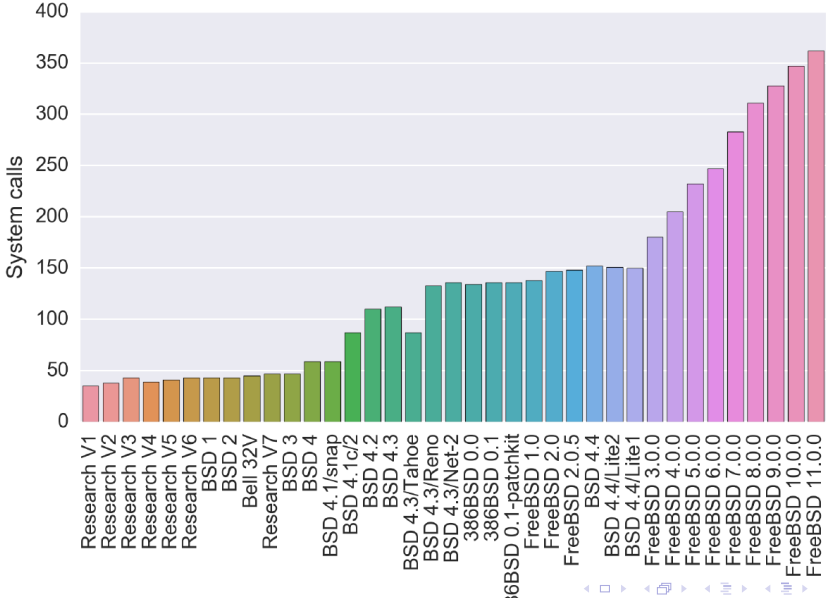
Introduction – History

First Research Edition (1971)

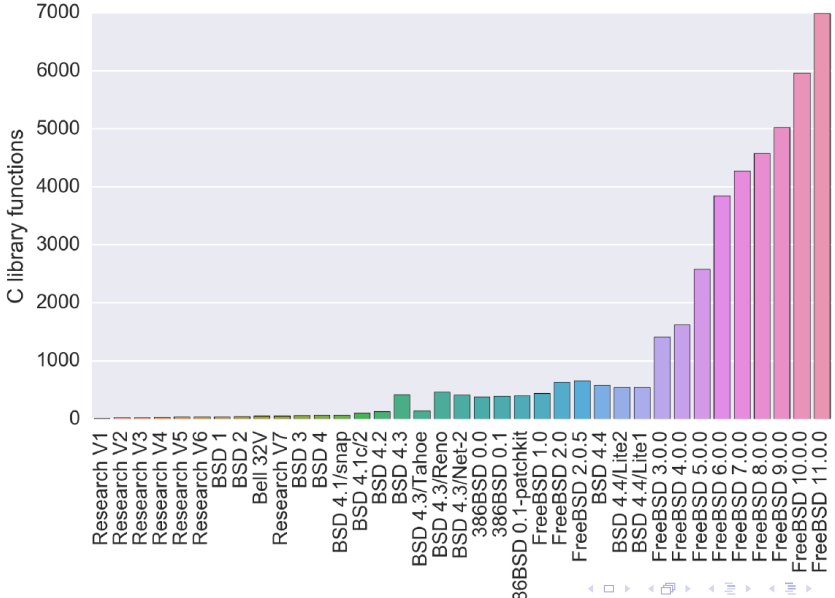
- Complete rewrite (4213 lines kernel)
- Reference architecture
  - 34 system calls
  - 18 common with PDP-7 version
  - 18 survive until today
- Binary code API
- Abstraction of standard I/O
- Devices as files

Half Century of Unix: History, Preservation, and Lessons Learned, Diomidis Spinellis, 2017
Analyzing a Decade of Linux System Calls, Mojtaba Bagherzadeh, etc. ICSE 2018

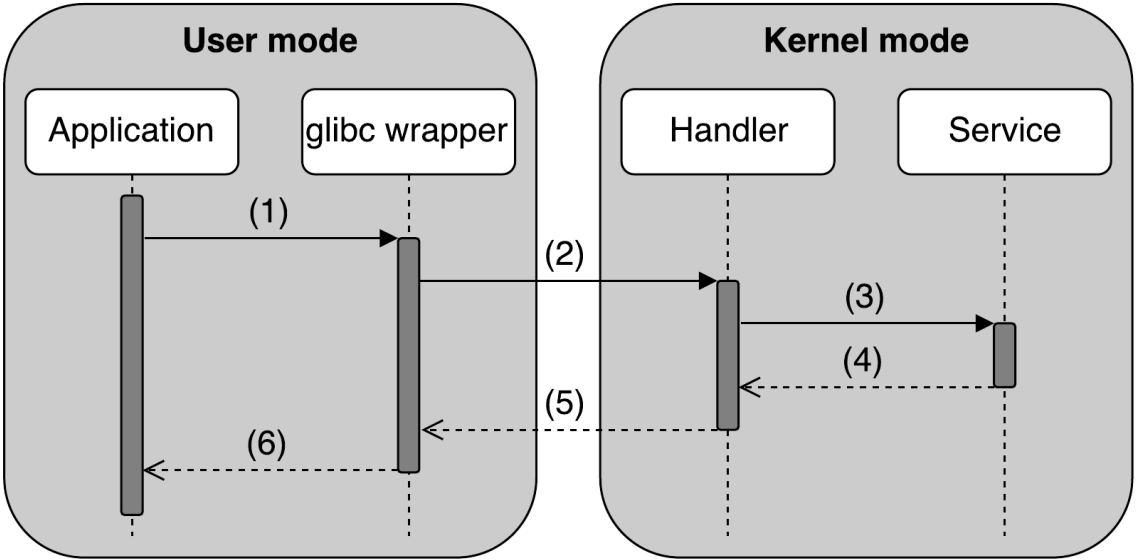
Introduction – History



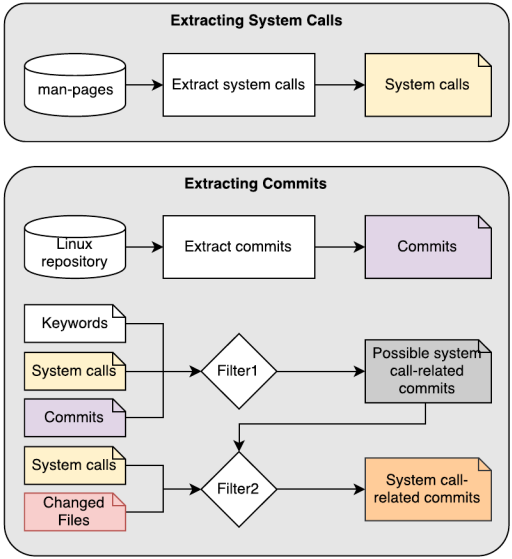
Introduction – History



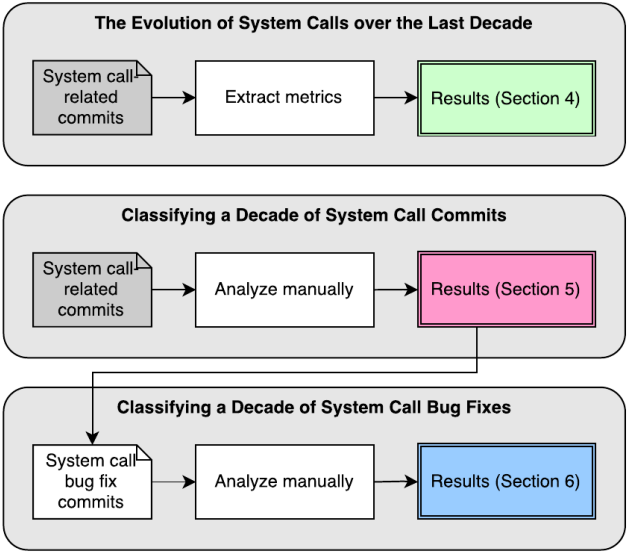
Introduction – The sequence of a system call



Introduction – An overview of syscall data collection



Introduction – An overview of syscall empirical study



Introduction – Syscall Categories

Code	Category	Example	Total
FS	File system & I/O	Reading and writing a file.	147
PM	Process management	Creating, cloning or debugging a process.	71
IPC	IPC[4] & network	Sharing memory between processes.	51
MM	Memory management	Mapping pages in memory.	27
SH	Signal handling	Killing a process.	24
TO	Time operations	Setting and querying the time.	23
SI	System info & settings	Retrieving information about the system.	21
SC	Scheduling	Thread prioritization.	14
SEC	Security & capabilities	Performing security checks.	8
MO	Modules	Loading a module.	6
All system calls			393

Introduction – sibling syscalls

Type of sibling	**Pattern**	**# of calls**	**Example**
Parameter extension	**[1..4]*	12	`dup()`, `dup2()`
Architecture	**[32|64]*	32	`truncate()`, `truncate64()`
Working directory	**at*	14	`open()`, `openat()`
Backwards compatibility	**old*	6	`vm86()`, `vm86old()`
Real time	*rt**	8	`sigreturn()`, `rt_sigreturn()`
Others	-	30	`waitpid()`, `wait4()`
Total number		102	

Introduction – new syscalls

Functionality	**# of system calls**	**Example**
Monitoring	8	`inotify()`, `getcpu()`
Synchronization	7	`eventfd()`, `signalfd()`
Hardware-specific	6	`cacheflush()`, `move_pages()`
Message passing	5	`process_vm_readv()`, `tee()`
Security	3	`bpf()`, `seccomp()`
Other[1]	7	`setns()`, `clock_adjtime()`
Total number	36	

Introduction – Classifying a Decade of System Call Commits

- **Add/remove**: The commit was made to add or remove one or more system calls.
- **Bug fix**: The commit was made to fix a bug.
- **Improvement**: The commit was made to make an improvement.
- **Restructuring**: The commit was made to conduct code restructuring, such as cleaning up comments or refactoring.

Introduction – The system calls with the most commits

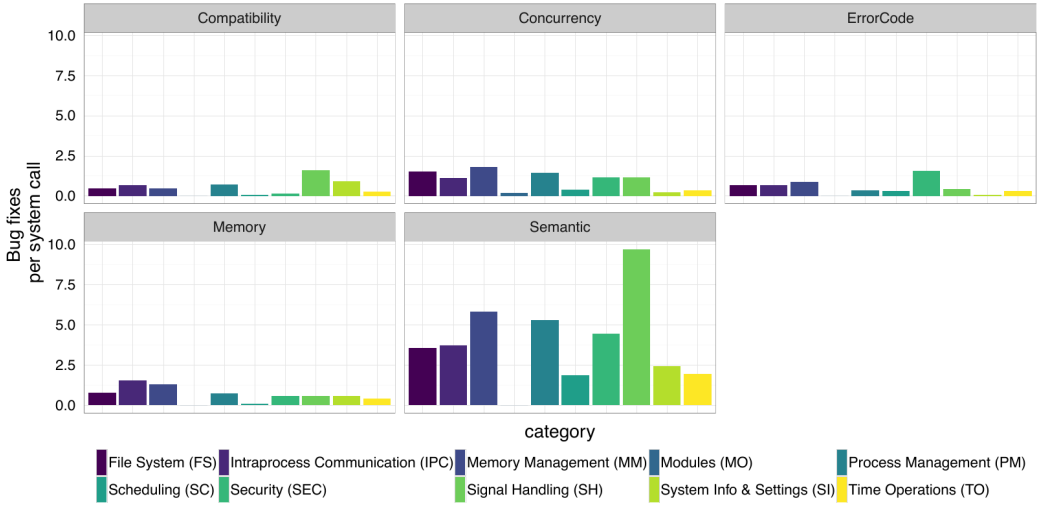
System call	ALL	Restructuring[1]	Bug fix	Improvement
`ptrace()`	743	46%	35%	21%
`signal()`	714	53%	33%	18%
`ioctl()`	438	44%	32%	25%
`futex()`	257	35%	43%	23%
`ipc()`	253	51%	23%	30%
`mmap()`	213	30%	43%	31%
`perf_event_open()`	199	10%	46%	45%
`readdir()`	169	46%	41%	14%
`splice()`	166	30%	40%	25%

Introduction – Classifying Bug Fixes for Sysalls

- **Compatibility**: Compatibility-related bugs are caused by compatibility issues between architectures (e.g., 32-bit versus 64-bit).
- **Concurrency**: Concurrency-related bugs are caused by issues with atom- icity, execution order, synchronization or locking, and lead to problems such as deadlock or race conditions.
- **Error code**: Error code-related bugs are caused by returning the wrong error code or handling a returned error code incorrectly.
- **Memory**: Memory-related bugs are caused by incorrect usage of the mem- ory, thereby introducing an issue such as a memory leak.
- **Semantic** : Semantic bugs are bugs in the implementation of the system call-specific behaviour, such as the logic of the service provided by the system call.

Signal handling system calls have the highest number of semantic (9.33) and compatibility-related (1.70) bug fixes per system call.

Introduction – Classifying Bug Fixes for Sysalls



Signal handling system calls have the highest number of semantic (9.33) and compatibility-related (1.70) bug fixes per system call.