Linux API for Software Diagnostics Accelerated

With Category Theory in View

Dmitry Vostokov
Software Diagnostics Services
Prerequisites

- Development experience

and (optional)

- Basic process core dump analysis
Training Goals

- Review fundamentals of Linux API
- Learn diagnostic analysis techniques
- See how Linux API knowledge is used during diagnostics and debugging
Training Principles

- Talk only about what I can show
- Lots of pictures
- Lots of examples
- Original content and examples
Schedule

- Review of relevant x64 and A64 disassembly
- General Linux API aspects
- Linux API formalization
- Linux API and languages
- Linux API classes
- Practical exercises
Training Idea

- Previous Accelerated Windows API training
  - Cybersecurity
  - Memory dump analysis
  - Reading Windows-based Code training

- Experience writing Linux API monitoring tools
General Linux API Aspects

- Header view
- Naming convention
- Basic type system
- Call types
- Export/import functions
- PLT and GOT
- Virtual process address space
- Calling convention
- API sequences
- API layers
- API and system calls
- API source code
- Shared Libraries
- API usage
- API internals

- Static linking
- Delayed dynamic linking
- API name patterns
- API namespaces
- API syntagms/paradigms
- Marked API
- ADDR patterns
- DebugWare patterns
- Memory analysis patterns
- API tracing
- Trace and log analysis patterns
- API and errors
- API and functional programming
- API and security
- API and versioning
Linux API Formalization

- API compositionality
- Category theory language
- A view of category theory
- Category theory square
- API category
- API functor
- API diagram
- API natural transformation
- Cross-platform API
- API adjunction
- Informal n-API
- API and trace categories
- API I/O
Linux API and Languages

- C#
- Scala Native
- Golang
- Rust
- Python
Linux API Classes

- System configuration
- File I/O
- File control
- Filesystem
- Dynamic memory
- Virtual memory
- Shared libraries
- Process
- IPC

- Job
- Signals
- Thread
- Networking
- Time
- Timers
- Tracing and logging
- Accounts
- Terminal
Links

- Core Dumps
  Included in Exercise L0

- Exercise Transcripts
  Included in this book
Exercise L0

- **Goal:** Install GDB and check if GDB loads a core dump correctly

- **Goal:** Install WinDbg or Debugging Tools for Windows, or pull Docker image, and check that symbols are set up correctly

- **Memory Analysis Patterns:** Stack Trace; Incorrect Stack Trace

- `\LAPI-Dumps\Exercise-L0-GDB.pdf`

- `\LAPI-Dumps\Exercise-L0-WinDbg.pdf`
Why Linux API?

- Development
- Malware analysis
- Vulnerability analysis and exploitation
- Reversing
- Diagnostics
- Debugging
- Monitoring
- Memory forensics
- Crash and hang analysis
- Secure coding
- Static code analysis
- Trace and log analysis
My History of Linux API

- C runtime library from 1989
- I started using Linux API in 1997 ([Old CV](#))
- Commercial application development in 2000 – 2003
- Reading of Linux API for core dump analysis since 2015
- Teaching foundations of x64 Linux debugging from 2021
  - and A64 Linux debugging from 2022, two books
- System programming using Linux API in 2022
- Linux disassembly and reversing course in 2022,
  - second edition in 2023
- Third edition of Linux core dump analysis course in 2023
- Commercial core dump analysis service in 2023

© 2023 Software Diagnostics Services
Perspectives of Linux API

- Memory analysis: dumps / live debugging
- Disassembly, reconstruction, reversing
- Trace and log analysis (strace, ltrace, perf, eBPF, procmon)
- Category theory
What Linux API?

- Source code perspective
- ABI (Application Binary Interface) perspective
  - Libraries
  - Syscalls
API and Language Levels

- Conceptual cross-platform level
- High-level and assembly language
- Machine language
x64 and A64
CPU Registers (x64)

- **RAX** $\supset$ **EAX** $\supset$ **AX** $\supseteq \{\text{AH, AL}\}

- **ALU**: **RAX**, **RDX**

- **Counter**: **RCX**

- **Memory copy**: **RSI** (src), **RDI** (dst)

- **Stack**: **RSP**, **RBP**

- **Next instruction**: **RIP**

- **New**: **R8** – **R15**, **Rx**(D|W|L)

RAX 64-bit    EAX 32-bit
Instructions: registers (x64)

- **Opcode SRC, DST # default AT&T flavour**

- **Examples:**

  - `mov $0x10, %rax`  # 0x10 → RAX
  - `mov %rsp, %rbp`  # RSP → RBP
  - `add $0x10, %r10`  # R10 + 0x10 → R10
  - `imul %ecx, %edx`  # ECX * EDX → EDX
  - `callq *%rdx`  # RDX already contains
    # the address of func (&func)
    # PUSH RIP; &func → RIP
  - `sub $0x30, %rsp`  # RSP−0x30 → RSP
    # make a room for local variables

© 2023 Software Diagnostics Services
Memory and Stack Addressing

Stack grows

Lower addresses

RSP - 0x20 \(\rightarrow\) RBP - 0x20
RSP - 0x18 \(\rightarrow\) RBP - 0x18
RSP - 0x10 \(\rightarrow\) RBP - 0x10
RSP - 0x8 \(\rightarrow\) RBP - 0x8
RSP \(\rightarrow\) RBP
RSP + 0x8 \(\rightarrow\) RBP + 0x8
RSP + 0x10 \(\rightarrow\) RBP + 0x10
RSP + 0x18 \(\rightarrow\) RBP + 0x18
RSP + 0x20 \(\rightarrow\) RBP + 0x20

Higher addresses
Instructions: memory load (x64)

- **Opcode** Offset(SRC), **DST**

- **Opcode** **DST**

**Examples:**

- `mov 0x10(%rsp), %rax`  # value at address RSP+0x10 → RAX
- `mov -0x10(%rbp), %rcx`  # value at address RBP-0x10 → RCX
- `add (%rax), %rdx`  # RDX + value at address RAX → RDX
- `pop %rdi`  # value at address RSP → RDI
  # RSP + 8 → RSP
- `lea 0x20(%rbp), %r8`  # address RBP+0x20 → R8
Instructions: memory store (x64)

- Opcode SRC, Offset(DST)
- Opcode SRC|DST

Examples:

- `mov %rcx, -0x20(%rbp)`  # RCX → value at address RBP-0x20
- `addl $1, (%rax)`  # 1 + 32-bit value at address RAX → 32-bit value at address RAX
- `push %rsi`  # RSP - 8 → RSP
  # RSI → value at address RSP
- `inc (%rcx)`  # 1 + value at address RCX → value at address RCX

© 2023 Software Diagnostics Services
Instructions: flow (x64)

- **Opcode DST**

**Examples:**

```
jmpq  0x10493fc1c  # 0x10493fc1c → RIP
     # (goto 0x10493fc1c)

jmpq  *0x100(%rip)  # value at address RIP+0x100 → RIP

callq 0x10493ff74  # RSP − 8 → RSP
0x10493fc14:  # 0x10493fc14 → value at address RSP
               # 0x10493ff74 → RIP
               # (goto 0x10493ff74)
```
# void proc(int p1, long p2);
mov  $0x1, %edi
mov  $0x2, %rsi
call proc
addr:

# void proc2();
# void proc(int p1, long p2) {
#   long local = 0;
#   proc2();
# }
proc:
push %rbp
mov  %rsp, %rbp
sub  $0x8, %rsp
mov  $0, -0x8(%rbp)
call proc2
adr2:
...
CPU Registers (A64)

- **X0 – X28, W0 – W28**
- **X16 (XIP0), X17 (XIP1)**
- Stack: **SP, X29 (FP)**
- Next instruction: **PC**
- Link register: **X30 (LR)**
- Zero register: **XZR, WZR**
- 64-bit floating point registers **D0 – D31**
- 128-bit **Q0 – Q31**
Instructions: registers (A64)

- **Opcode DST, SRC, SRC₂**

- **Examples:**

  - `mov x0, #16`  // `X0 ← 16 (0x10)`
  - `mov x29, sp`  // `X29 ← SP`
  - `add x1, x2, #16`  // `X1 ← X2+16 (0x10)`
  - `mul x1, x2, x3`  // `X1 ← X2*X3`
  - `blr x8`  // `X8 already contains the address of func (&func)`  
    // `LR ← PC+4; PC ← &func`
  - `sub sp, sp, #48`  // `SP ← SP-48 (-0x30)`  // make a room for local variables

© 2023 Software Diagnostics Services
Memory and Stack Addressing

Stack grows

Lower addresses
- \( SP - 0x20 \) → \( \leftarrow X29 - 0x20 \)
- \( SP - 0x18 \) → \( \leftarrow X29 - 0x18 \)
- \( SP - 0x10 \) → \( \leftarrow X29 - 0x10 \)
- \( SP - 0x8 \) → \( \leftarrow X29 - 0x8 \)
- \( SP \) → \( \leftarrow X29 \)
- \( SP + 0x8 \) → \( \leftarrow X29 + 0x8 \)
- \( SP + 0x10 \) → \( \leftarrow X29 + 0x10 \)
- \( SP + 0x18 \) → \( \leftarrow X29 + 0x18 \)
- \( SP + 0x20 \) → \( \leftarrow X29 + 0x20 \)

Higher addresses
Instructions: memory load (A64)

- Opcode DST, DST₂, [SRC, Offset]
- Opcode DST, DST₂, [SRC], Offset // Postincrement

Examples:

```
ldr   x0, [sp]   // X0 ← value at address SP+0
ldr   x0, [x29, #−8] // X0 ← value at address X29−0x8
ldp   x29, x30, [sp, #32] // X29 ← value at address SP+32 (0x20)
          // X30 ← value at address SP+40 (0x28)
ldp   x29, x30, [sp], #16 // X29 ← value at address SP+0
          // X30 ← value at address SP+8
          // SP ← SP+16 (0x10)
```
Instructions: memory store (A64)

- **Opcode** SRC, SRC₂, [DST, Offset]
- **Opcode** SRC, SRC₂, [DST, Offset]! // Preincrement

**Examples:**

```assembly
str  x0, [sp, #16] // x0 → value at address SP+16 (0x10)
str  x0, [x29, #-8] // x0 → value at address X29-8
stp  x29, x30, [sp, #32] // x29 → value at address SP+32 (0x20)
                              // x30 → value at address SP+40 (0x28)
stp  x29, x30, [sp, #-16]! // SP ← SP-16 (-0x10)
                              // x29 → set value at address SP
                              // x30 → set value at address SP+8
```

© 2023 Software Diagnostics Services
Instructions: flow (A64)

- **Opcodes**: DST, SRC

- **Examples**:
  - `adrp  x0, 0x420000`  // x0 ← 0x420000
  - `b  0x10493fc1c`  // PC ← 0x10493fc1c
    // (goto 0x10493fc1c)
  - `br  x17`  // PC ← the value of X17
  - `0x10493fc14:`  // PC == 0x10493fc14
  - `bl  0x10493ff74`  // LR ← PC+4 (0x10493fc18)
    // PC ← 0x10493ff74
    // (goto 0x10493ff74)
Function Call and Prolog (A64)

// void proc(int p1, long p2);
mov  w0, #0x1
mov  x1, #0x2
bl   proc
addr:

// void proc2();
// void proc(int p1, long p2) {
//   long local = 0;
//   proc2();
// }
proc:
stp  x29, x30, [sp, #0x18]!
mov  x29, sp
str  zxr, [x29, #0x16]
bl   proc2
adr2:
...
General Linux API Aspects
Lexicon

- IAT ↔ PLT
- DLL ↔ SO
- Module ↔ Shared Library
Manual Page and Header Views

- Manual pages (2 and 3)
  - Syscalls / overview
  - Library calls

- Headers
  - Kernel source code cross-reference
  - Manual pages 0
Naming Convention

- **Naming conventions**

- Functions, types, parameters, fields: **twowords**, **snake_case**
  - pthread_create
  - pid_t
  - tm_sec

- Constants, macros: **TWOWORDS**, **SCREAMING_SNAKE_CASE**
  - SOCK_STREAM
Basic Type System

- Basic types are typedef-ed
- inttypes.h
  - uint64_t
- sys/types.h
  - size_t, pid_t

GDB Commands

```
(gdb) info types
(gdb) info functions
```

WinDbg Commands

```
0:000> dt *!*  
0:000> x *!*  
```
Call Types (x64)

- **Direct** (same executable or shared library)

  ```
  shell_execve:
  ...
  0x000055e16508f807 <+103>: callq 0x55e1650891a0 <file_isdir>
  ```

- **Indirect**
  - **Pointer** (.got.plt)
    - **PLT** inter-module call
      ```
      shell_execve:
      ...
      0x000055e16508f89b <+251>: callq 0x55e165078aa0 <open@plt>
      ```

  ```
  open@plt:
  ...
  0x000055e165078aa0 <+0>: jmpq *0xe7aaa(%rip) # 0x55e165160550 <open@got.plt>
  ```
Call Types (A64)

- **Direct** (same executable or shared library)

  ```
  shell_execve:
  ...
  0x0000aaaabb6f71e4 <+160>:   bl 0xaaaabb742530 <file_status>
  ```

- **Indirect**
  - **Pointer** (.got)
    - **PLT** inter-module call

  ```
  shell_execve:
  ...
  0x0000aaaabb6f7210 <+204>:   bl 0xaaaabb6d1ec0 <open@plt>
  
  open@plt:
  ...
  0x0000aaaabb6d1ec0 <+0>:   adrp x16, 0xaaaabb7ff000
  0x0000aaaabb6d1ec4 <+4>:   ldr x17, [x16, #1464]
  ...
  0x0000aaaabb6d1ecc <+12>:   br x17
  ```
API as Interface

- Provided by (exported from) some .so library
- Used by (imported by) executable or .so
- Can be functional or object-oriented
Exploration Tools

- `ldd`
- `readelf`
- `objdump`

- `$ LD_DEBUG=libs bash`
Symbol Import/Export

ELF (app)

... .plt / .rela.plt
open@plt:
...

.dynsym
...
"open@GLIBC_2.2.5"
...

ELF (libc.so.6)

... .plt / .rela.plt

.dynsym
...
"open@GLIBC_2.2.5": open
...

open:
...
Procedure Linkage Table (x64)

ELF (app)

- .plt/.plt.got
- open@plt:
  - jmpq *<open@got.plt>
- ... .got/.got.plt
- open@got.plt: open
- ... .dynsym
- ... callq open@plt
- ...

ELF (libc.so.6)

- .plt/.plt.got
- .got/.got.plt
- .dynsym
- open:
  - ...

© 2023 Software Diagnostics Services
Procedure Linkage Table (A64)

ELF (app)
- .plt
- open@plt: br *<open@got>
- got
- dynsym
- callq open@plt

ELF (libc.so.6)
- .plt
- got
- dynsym
- open:
- ...
Virtual Process Address Space

ELF (app)

... plt
... open@plt:
...
...
...
callq open@plt
...

ELF (libc.so.6)

open:
...

ELF (app)

... plt
... open@plt:
...
...
...
callq open@plt
...

ELF (libc.so.6)

open:
...
Exercise L1

- **Goal:** Explore import/export information and calls to shared libraries

- **ADDR Patterns:** Call Path

- \`\LAPI-Dumps\Exercise-L1-GDB.pdf\`

- \`\LAPI-Dumps\Exercise-L1-WinDbg.pdf\`
VS Code and WSL

- **Get started**

- **Troubleshooting**

  Add to `.bashrc`

  ```
  alias code="'/mnt/c/Users/[USER]/AppData/Local/Programs/Microsoft VS Code/Code.exe'"
  ```
Calling Convention

- ioctl (from documentation)

- Actual declaration (sys/ioctl.h)

```c
extern int ioctl (int __fd, unsigned long int __request, ...) __THROW;
```

- __THROW is defined in sys/cdefs.h

- Argument passing order (simplified)

  - x64 left-to-right via RDI, RSI, RDX, RCX, R8, R9, right-to-left PUSH ...

  - A64 left-to-right via $x0 - x7, [SP], [SP+8], [SP+16], ...

- System V Application Binary Interface: AMD64 Architecture Processor Supplement
Test8params(int p1, int p2, int p3, int p4, int p5, int p6, int p7, int p8);

**Caller**

EDI (p1)
ESI (p2)
EDX (p3)
ESX (p4)
R8D (p5)
R9D (p6)

**Callee**

EDI (p1)
ESI (p2)
EDX (p3)
ESX (p4)
R8D (p5)
R9D (p6)

RSP: 0`
p7
RSP+0x8: 0`
p8

RSP: return address
RSP+0x8: 0`
p7
RSP+0x10: 0`
p8
Parameter Passing (A64)

Test10params(int p1, int p2, int p3, int p4, int p5, int p6, int p7, int p8, int p9, int p10);

<table>
<thead>
<tr>
<th>Caller</th>
<th>Callee</th>
</tr>
</thead>
<tbody>
<tr>
<td>W0 (p1)</td>
<td>W0 (p1)</td>
</tr>
<tr>
<td>W1 (p2)</td>
<td>W1 (p2)</td>
</tr>
<tr>
<td>W2 (p3)</td>
<td>W2 (p3)</td>
</tr>
<tr>
<td>W3 (p4)</td>
<td>W3 (p4)</td>
</tr>
<tr>
<td>W4 (p5)</td>
<td>W4 (p5)</td>
</tr>
<tr>
<td>W5 (p6)</td>
<td>W5 (p6)</td>
</tr>
<tr>
<td>W6 (p7)</td>
<td>W6 (p7)</td>
</tr>
<tr>
<td>W7 (p8)</td>
<td>W7 (p8)</td>
</tr>
</tbody>
</table>

call

SP: 0`p9
SP+8: 0`p10

SP: 0`p9
SP+8: 0`p10
Exercise L2

- **Goal:** Compare calling conventions on x64 and A64 platforms

- **ADDR Patterns:** Call Prologue; Call Parameter; Function Prologue

- \LAPI-Dumps\Exercise-L2-GDB.pdf

- \LAPI-Dumps\Exercise-L2-WinDbg.pdf
Static Linkage

Example

- `core.18256` from `/LAPI/x64/static-example`

```c
main:
...

callq 0x43c1f0 <sleep>
...

callq 0x43d6e0 <ioctl>
...
```
Shared Libraries

WinDbg Commands
0:000> lm
0:000> x mpattern!pattern
0:000> x *!pattern

GDB Commands
(gdb) info sharedlibrary
(gdb) info files
(gdb) info functions pattern

© 2023 Software Diagnostics Services
Modules and Analysis Patterns

- Module memory analysis patterns
  - Module Collection
  - Coupled Modules
  - Duplicated Module

- Namespace malware analysis pattern
Exercise L3

- **Goal:** Explore shared libraries and their dependencies

- **Memory Analysis Patterns:** Module Collection; Coupled Modules; Value References

- **Malware Analysis Patterns:** Namespace

- \LAPI-Dumps\Exercise-L3-GDB.pdf

- \LAPI-Dumps\Exercise-L3-WinDbg.pdf
API Usage

- Module usage (static analysis)
  - Hidden Module

- Function usage (dynamic analysis)

GDB Commands

- (gdb) break function
- (gdb) rbreak regex
Exercise L4

- **Goal:** Find usage of specific Linux API functions

- **Debugging Implementation Patterns:** Code Breakpoint; Breakpoint Action

- \`\`\`LAPI-Dumps\`\`\`\`Exercise-L4-GDB.pdf
Delayed Dynamic Linking

- **First call:**
  
  open@plt:
  
  ```
  0x0000555555581aa0 <+0>: jmpq *0xe7aaa(%rip) # 0x555555669550 <open@got.plt>
  0x0000555555581aa6 <+6>: pushq $0xa7
  0x0000555555581aab <+11>: jmpq 0x5555555581020
  ```

- **Subsequent calls:**
  
  open@plt:
  
  ```
  0x0000555555581aa0 <+0>: jmpq *0xe7aaa(%rip) # 0x5555555569550 <open@got.plt>
  0x0000555555581aa6 <+6>: pushq $0xa7
  0x0000555555581aab <+11>: jmpq 0x5555555581020
  ```
Exercise L5

- **Goal**: Explore the delayed dynamic linking
- **Debugging Implementation Patterns**: Code Breakpoint; Break-in
- **ADDR Patterns**: Call Path

\LAPI-Dumps\Exercise-L5-GDB.pdf
API Sequences (Prescriptive)

- open, ..., close
- malloc, ..., free
- pthread_create, ..., pthread_join
- socket, ..., bind, ..., listen, ..., accept
- wl_surface_damage, ..., wl_surface_attach, ..., wl_surface_commit
API Sequences (Descriptive)

- Horizontal
  - Code disassembly
  - Traces and logs (*Thread of Activity* analysis pattern)

- Vertical
  - Stack trace
  - Traces and logs (*Fiber Bundle* analysis pattern)
API Internals

- Memory analysis patterns:
  - Hooked Functions (User Space)
  - Module patterns
    - Hooked Modules

- Malware analysis patterns:
  - Patched Code

GDB Commands

- (gdb) disassemble function
- (gdb) x/<n>i address

WinDbg Commands

- 0:000> !for_each_module
- 0:000> u fname
- 0:000> uf /c fname
API and System Calls

- **Syscall numbers and arguments**

- **API that does not require syscalls**
  - `strlen`
  - `malloc`
  - `fread`

- **API that requires kernel services**
  - `getpid`
  - `mmap`
  - `read`
Goal: Explore API layers and internals of specific API functions; check whether the selected API functions use system calls

ADDR Patterns: Function Skeleton; Call Path
API Name Patterns

- `create/open/delete/close`
- `pthreads`
- `display/surface/window`
- `alloc/free`
- `read/write`

GDB Commands

```text
(gdb) info functions pattern
```

WinDbg Commands

```text
0:000> x module!fpattern
```
API Namespaces

- Functions required to accomplish a particular task
  - Example: network communication
API Syntagms/Paradigms

- **Syntagms / syntagmatic analysis**
- **Paradigms / paradigmatic analysis**
Marked API

- **Marked Message** trace and log analysis pattern
- Points to the presence or absence of activity

Example:

- `execve` [-]
- `socket` [+]
- `connect` [+]
- `create` [-]
ADDR Patterns

- From Accelerated Disassembly Deconstruction Reversing

- List of pattern names

- Pattern descriptions
DebugWare Patterns

- Patterns for troubleshooting and debugging tools

- **API Query**

  Periodic or asynchronous query of the same set of APIs and logging of their input and output data.
**Patterns vs. Analysis Patterns**

**Diagnostic Pattern**: a common recurrent identifiable problem together with a set of recommendations and possible solutions to apply in a specific context.

**Diagnostic Problem**: a set of indicators (symptoms, signs) describing a problem.

**Diagnostic Analysis Pattern**: a common recurrent analysis technique and method of diagnostic pattern identification in a specific context.

**Diagnostics Pattern Language**: common names of diagnostic and diagnostic analysis patterns. The same language for any operating system: Windows, macOS, Linux, ...
Memory Analysis Patterns

- User space
  - Process memory dumps

- Function analysis patterns
  - Stack Trace Collection
  - Well-Tested Function
  - False Function Parameters
  - String Parameter
  - Small Value / Design Value
  - Stack Trace
  - Execution Residue
  - Hidden Parameter
  - Parameter Flow
  - Data Correlation
# Thread and Adjoint Thread

<table>
<thead>
<tr>
<th>PID</th>
<th>PPID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>22992</td>
<td>1</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>2368</td>
<td>124</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>23900</td>
<td>22992</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>2368</td>
<td>124</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>23909</td>
<td>22992</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>2368</td>
<td>124</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>23917</td>
<td>22992</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>2368</td>
<td>124</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>23918</td>
<td>23917</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>2368</td>
<td>124</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>23925</td>
<td>23917</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>2368</td>
<td>124</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PID</th>
<th>PPID</th>
<th>USER</th>
<th>PR</th>
<th>NI</th>
<th>VIRT</th>
<th>RES</th>
<th>SHR</th>
<th>S</th>
<th>%CPU</th>
<th>%MEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>8298</td>
<td>8297</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>7636</td>
<td>4480</td>
<td>3284</td>
<td>S</td>
<td>0.0</td>
<td>0.1</td>
</tr>
<tr>
<td>8297</td>
<td>8296</td>
<td>root</td>
<td>20</td>
<td>0</td>
<td>2348</td>
<td>120</td>
<td>0</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2526</td>
<td>2525</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>326880</td>
<td>324772</td>
<td>26196</td>
<td>S</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>2517</td>
<td>2516</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>326880</td>
<td>34576</td>
<td>26260</td>
<td>S</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>4859</td>
<td>4858</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>1867316</td>
<td>823340</td>
<td>148868</td>
<td>S</td>
<td>0.3</td>
<td>10.2</td>
</tr>
<tr>
<td>8972</td>
<td>8971</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>5918712</td>
<td>497800</td>
<td>31664</td>
<td>S</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>2347</td>
<td>2346</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>326880</td>
<td>324772</td>
<td>26196</td>
<td>S</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>2198</td>
<td>2197</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>445928</td>
<td>61892</td>
<td>32144</td>
<td>S</td>
<td>0.8</td>
<td></td>
</tr>
<tr>
<td>2236</td>
<td>2235</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>1021380</td>
<td>161120</td>
<td>37840</td>
<td>S</td>
<td>0.3</td>
<td>2.0</td>
</tr>
<tr>
<td>2247</td>
<td>2246</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>326880</td>
<td>34576</td>
<td>26260</td>
<td>S</td>
<td>0.4</td>
<td></td>
</tr>
<tr>
<td>2187</td>
<td>2186</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>946236</td>
<td>188696</td>
<td>34664</td>
<td>S</td>
<td>0.3</td>
<td>1.3</td>
</tr>
<tr>
<td>2183</td>
<td>2182</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>2388</td>
<td>756</td>
<td>688</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2178</td>
<td>2177</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>2388</td>
<td>756</td>
<td>688</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2198</td>
<td>2197</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>2388</td>
<td>756</td>
<td>688</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2236</td>
<td>2235</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>2388</td>
<td>756</td>
<td>688</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2247</td>
<td>2246</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>2388</td>
<td>756</td>
<td>688</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2187</td>
<td>2186</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>2388</td>
<td>756</td>
<td>688</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2183</td>
<td>2182</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>2388</td>
<td>756</td>
<td>688</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2178</td>
<td>2177</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>2388</td>
<td>756</td>
<td>688</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2177</td>
<td>2176</td>
<td>coredump</td>
<td>20</td>
<td>0</td>
<td>2388</td>
<td>756</td>
<td>688</td>
<td>S</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

© 2023 Software Diagnostics Services
Trace and Log Analysis Patterns

- Function calls:
  - Thread of Activity
  - Fiber of Activity
  - Adjoint Thread of Activity
  - Strand of Activity
  - Discontinuity
  - Fiber Bundle
  - Weave of Activity
API and Errors

- `errno.h` *(values)*
- `errno`
- `rtld_errno`

GDB needs an executable linked with `-pthread`

- FS (x64) and TPIDR_EL0 (ARM64) `TLS_ABI`
Exercise L7

- **Goal:** Explore error handling implementation in Linux API
- **ADDR Patterns:** Call Epilogue
- **Memory Analysis Patterns:** Last Error Collection

- \LAPI-Dumps\Exercise-L7-GDB.pdf
- \LAPI-Dumps\Exercise-L7-WinDbg.pdf

© 2023 Software Diagnostics Services
API and Functional Programming

- Referential transparency

- `strlen`

  \[
  \text{strlen}(s), \text{strlen}(s) \to n, n
  \]

- `read`

  \[
  \text{read}(fd, buf, len), \text{read}(fd, buf, len) \to n1, n2
  \]

- Side effects
API and Security

- Maliciousness: What, When, Where
  - SecQuant: Quantifying Container System Call Exposure
  - System call risk level classification / syscalls / masks

- Vulnerability: How
  - SAST
  - Static code analysis tools
API and Versioning

- Windows: *Ex*-suffix, longer descriptive function names
  
  CopyFile/CopyFileEx
  CreateThread/CreateRemoteThread

- Linux: numbering, shorter prefixes/suffixes
  
  openat/openat2
  accept/accept4
  read/pread/readv/preadv/preadv2
Linux API Formalization

Ideas from Conceptual Mathematics
API Compositionality

- **Principle of compositionality**

![Diagram showing API calls and code glue connections]
Category Theory Language

- **Category**
  - Objects
  - Arrows between objects (must be transitive, if $A \to B$ and $B \to C$ then $A \to C$)

- **Functor**
  - Arrow between categories (can be the same category)
  - Maps objects to objects and arrows to arrows

- **Natural Transformation**
  - Arrows between functors in a category of functors

- **Adjunction**
  - Relationship between functors, change of perspective, back translation
A View of Category Theory
Category Theory Square

Category \rightarrow \text{Functor} \leftarrow \text{Adjunction} \rightarrow \text{Natural Transformation}
API Category

- API as objects, glue code as arrows
- API as arrows, glue code as objects fails at composition
- **Initial** and **terminal** API objects in subcategories

![Diagram showing the relationship between open, read, write, and close with initial and terminal objects in subcategories.]
API Functor

- Translates between API layers (different API)
- Stack trace as functor
- Translates between different API sequences
- Endofunctor – between the same API
- Translates between different code implementations
API Diagram

- Indexed set → diagram
- Functor from a shape (pattern)
API Natural Transformation

- Maps between different vertical API sequences (stack traces)
- Maps between different code translations
- Diagnostics and debugging as natural transformation
Cross-platform API

- Windows API / Linux API
- Similar diagrams
- Cross-platform development as a natural transformation
API Adjunction

- Navigation between different API sequences
- Call and return stack trace sequences, callbacks (when stack traces correspond to vertical API sequences)
- Back translation between traces/logs (when traces correspond to API horizontal sequences)
Informal n-API

- Arrows between arrows
- 1-API – normal API usage
- 2-API – diagnostics, debugging
- 3-API – higher diagnostics, debugging (debugging the debugging)
- ∞-API – for homework 😊
API and Trace Categories

- 1-category API *(semigroup)*
- 2-category of traces and logs

© 2023 Software Diagnostics Services
API I/O

- Categories – one input, one output
- Operads – many inputs, one output
- Properads – many inputs, many outputs

```c
int socketpair(
    int domain,
    int type,
    int protocol,
    int sv[2]
);
```
Linux API and Languages
API and C#

- Installation
- Platform Invoke
Goal: Install .NET environment and write a simple program that uses Linux API

LAPI-Dumps\Exercise-L8.pdf
API and Scala Native

- Documentation
- Scala Native
- Native code interoperability
Exercise L9

- **Goal:** Install Scala Native environment and write a simple program that uses Linux API

- \LAPI-Dumps\Exercise-L9.pdf
API and Golang

- **unix**

- **Example:**

```go
package main

import (
    "golang.org/x/sys/unix"
)

func main() {
    unix.Exec("/bin/ps", nil, nil)
}
```
API and Rust

- Unsafe: **libc**
- Safe: **nix**, **rustix**
- Example:

```rust
use libc::{c_char, execve};

fn main() {
    unsafe {
        unsafe {
            execve("/bin/ps".as_ptr() as *const c_char, std::ptr::null(), std::ptr::null());
        }
    }
}
```
API and Python

- os
- ctypes

Example:

```python
import ctypes

c ctypes.CDLL("libc.so.6").execve("/bin/ps".encode("utf-8"), 0, 0)
```
Linux API Classes
Tracing

- **strace** (syscalls, signals)
- **ltrace** (libraries, syscalls, signals)

**Examples**

- `strace bash`
- `ltrace bash`
- `ltrace -S bash`
Classification

- Classification and Grouping of Linux System Calls
- A study of modern Linux API usage and compatibility (slides)
- A study of modern Linux API usage and compatibility (paper)
System Configuration API

- **Library**
  - Invariant and increasable `sysconf`
  - Pathnames `(f)` `pathconf`

- **Syscalls**
  - Identification `uname`
File I/O API

- **File I/O Essentials**

- **Layers**
  - Stream-based: `f(open|seek|read|write|close)`
  - Syscall-based: `creat|open|lseek|read|write|ioctl|close` on `fd`

- **Buffering**
  - Block-based: `f(read|write)`
  - Line-based: `(f)puts,(f)printf`
  - Unbuffered: `(p)read(v),(p)write(v)`

- **Multiplexing**
  - `(p)select, poll, epoll_(create|ctl|wait)`
File Control API

- **Library**
  - Mixing `FILE*` and `fd` `fileno, fdopen`
  - Buffering `set(v)buf, fflush`
  - Temp `mkstemp, tmpfile`

- **Syscalls**
  - Buffering `f(data)sync`
  - Access `posix_fadvise`
  - Control `fcntl`
  - Locking `flock`
  - Truncation `(f)truncate`
Filesystem API

- **Library**
  - Directories: (n)ftw, (fd)(open|read)dir, remove, get(c)wd
  - Paths: realpath, (dir|base)name

- **Syscalls**
  - Mounting: (u)mount
  - Metadata: (f|l)stat(at|vfs)
  - Attributes: (list|get|set)(f|l)xattr
  - Permissions: (l|f)ch(own|mod), umask
  - Time: utime(s)
  - Links: (un)link, rename
  - Symlinks: (sym|read)link
  - Directories: (mk|rm|ch)dir, chroot
  - Monitoring: inotify_(init|(add|rm)_watch)
Dynamic Memory API

- **Library**
  - Control `mall(opt|info), memalign`
  - Allocation `(m|c|re)alloc`
  - Deallocation `free`
  - Debugging `m(un)trace, m(check|probe)`
  - Stack `alloca`

- **Syscalls**
  - Adjustment `(s)brk`
Virtual Memory API

- Library
  - Usage `posix_madvise`

- Syscalls
  - Mapping `m((un|re)map|sync), remap_file_pages`
  - Protection `mprotect`
  - Locking `m(un)lock(all)`
  - Residence `mincore`
  - Usage `madvise`
Shared Libraries API

- **Library**
  - Loading `dlopen`
  - Unloading `dlclose`
  - Errors `dlerror`
  - Symbols `dl(addr|sym)`
Process API

- **Library**
  - Execution `exec(v(p)|l(e|p)), fexecve, system`
  - Termination `exit`
  - Exit `(at|on_)exit`
  - Capabilities `cap_((get_|set_)proc|free)`

- **Syscalls**
  - Creation `fork, clone`
  - Execution `execve`
  - Termination `_exit`
  - Waiting `wait(pid|id|3|4)`
  - Resources `acct, getrusage, (get|set)rlimit`
  - Priority `(get|set)priority, sched_get_priority_((min|max))`
  - Scheduling `sched_((set|get)(scheduler|param)|yield|rr_get_interval)`
  - Affinity `sched_(set|get)affinity`
  - Capabilities `prctl`
IPC API

- **Library**
  - Pipes `p(open|close) FILE*`
  - FIFO `mkfifo`
  - Keys `ftok`
  - POSIX `(mq|sem|shm)(open|unlink|*)`

- **Syscalls**
  - Pipes `pipe`
  - Handles `dup(2), close`
  - Message queues `msg(get|snd|rcv|ctl)`
  - Semaphores `sem(get|ctl|op)`
  - Shared memory `shm(get|at|ctl|dt)`
Job API

- **Library**
  - Terminal `ctermid, tc(get|set)pgrp`

- **Syscalls**
  - Groups `(get|set)pgid`
  - Sessions `(get|set)sid`
Signals API

- **Library**
  - Sets \( \text{sig}((\text{empty} | \text{fill} | \text{add} | \text{del} | \text{and} | \text{or} | \text{isempty}) \text{set}) | \text{ismember}) \)
  - Sending \text{raise}, \text{killpg}, \text{abort}
  - Description \text{strsignal}, \text{psignal}
  - Waiting \text{pause}
  - BSD \text{sig}(\text{vec} | \text{block} | (\text{get} | \text{set}) \text{mask} | \text{pause})

- **Syscalls**
  - Disposition \text{signal}, \text{sigaction}
  - Mask \text{sig}(\text{procmask} | \text{pending})
  - Sending \text{kill}, \text{sigqueue}
  - Waiting \text{sig}(\text{suspend} | \text{waitinfo}), \text{signalfd}
  - Stack \text{sigaltstack}
Thread API

Library

- One-time initialization `pthread_once`
- Creation `pthread_(create|atfork)`
- Termination `pthread_exit`
- Identification `pthread_(self|equal)`
- Wait `pthread_(join|detach)`
- Cancellation `pthread_(set|test)cancel(state|type)`
- Cleanup `pthread_cleanup_(push|pop)`
- Signals `pthread_(sigmask|kill|sigqueue), sigwait`
- Attributes `pthread_attr_(init|set*|destroy)`
- Synchronization `pthread_mutex__(init|(un|try|timed)lock|destroy)|attr_(init|set*|destroy))`  
  `pthread_cond_(init|signal|broadcast|(timed)wait|destroy)`
- Data `pthread_(key_create|(set|get)specific)`
Networking API

- **Library**
  - Addresses: `getaddrinfo`, `gai_strerror`
  - Names: `getnameinfo`

- **Syscalls**
  - Sockets: `socket`, `bind`, `listen`, `accept`, `connect`, `close`, `shutdown`
  - Streaming: `write`, `send`, `read`, `recv`
  - Datagrams: `recvfrom`, `sendto`
  - Messages: `send`, `recv`, `msg`
  - Files: `sendfile`
  - Domain: `socketpair`
  - Addresses: `get(sock|peer)name`
  - Options: `get|setsockopt`
Time API

- **Library**
  - Conversion (c|asc|strf|strp|get|local|mk)time
  - Zones tzset
  - Locales setlocale
  - Correction adjtime
  - Process clock

- **Syscalls**
  - Calendar (get|set)timeofday, (s)time
  - Process times
Timers API

- **Library**
  - Low-res `sleep`
  - Clock `(clock|pthread)_getcpuclockid`

- **Syscalls**
  - Intervals `(get|set)itime, timer_(create|settime|delete)`
  - Repeating `timer_getoverrun`
  - Once `alarm`
  - File `timerfd_(create|(get|set)time)`
  - Clock `clock_(get|set)(time|res)`
  - Hi-res `(clock_)nanosleep`
Tracing and Logging API

- **Library**
  - Writing `(open|v)sys|close)log`
  - Filtering `setlogmask`

- **Syscalls**
  - Process `ptrace`
  - Performance and monitoring `perf_event_open`
  - eBPF `bpf`
Accounts API

- **Library**
  - Records `getpw(nam|uid)`, `getgr(nam|gid)`
  - Scanning `getspnam`, `(set|get|end)spent`
  - Groups `(init|get|set)groups`
  - Encryption `crypt(_r)`

- **Syscalls**
  - ID `(get|set)((r)e)(s)(u|g)id`
  - Filesystem `setfs(u|g)id`
Terminal API

- **Library**
  - Identification: `isatty`, `ttymname`
  - Attributes: `tc(get|set)attr`
  - Speed: `cf(get|set)(to|ti)speed`
  - Control: `tc(sendbreak|drain|flush|flow)`
  - Windows: `ioctl`
  - Pseudo: `(posix_open|grant|unlock)pt, ptsname`
References and Resources
Resources (Construction)

- Windows System Programming, Fourth Edition (Appendix B)
- Programming with POSIX Threads
- The Linux Programming Interface
- Hands-On System Programming with Linux
- Linux System Programming Techniques
- Advanced Programming in the UNIX Environment
- Effective TCP/IP Programming
- UNIX Systems Programming
- Low Level X Window Programming
- Wayland Architecture / Wayland Book
Resources (Postconstruction)

- WinDbg Help / [WinDbg.org](http://WinDbg.org) (quick links)
- [DumpAnalysis.org](http://DumpAnalysis.org) / [SoftwareDiagnostics.Institute](http://SoftwareDiagnostics.Institute) / [PatternDiagnostics.com](http://PatternDiagnostics.com)
- [Debugging.TV](http://Debugging.TV) / [YouTube.com/DebuggingTV](http://YouTube.com/DebuggingTV) / [YouTube.com/PatternDiagnostics](http://YouTube.com/PatternDiagnostics)
- Foundations of Linux Debugging, Disassembling, and Reversing
- Foundations of ARM64 Linux Debugging, Disassembling, and Reversing
- Software Diagnostics Library
- Encyclopedia of Crash Dump Analysis Patterns, Third Edition
- Trace, Log, Text, Narrative, Data
- Memory Dump Analysis Anthology (Diagnomicon)
Resources (Training)

- Accelerated Linux Core Dump Analysis, Third Edition
- Accelerated Linux Debugging
- Accelerated Linux Disassembly, Reconstruction, and Reversing, Second Edition
- Accelerated Software Trace Analysis, Revised Edition, Part 1: Fundamentals and Basic Patterns
Resources (Category Theory)

Applied category theory books that have chapters explaining category theory:

- Conceptual Mathematics: A First Introduction to Categories
- The Joy of Abstraction: An Exploration of Math, Category Theory, and Life
- Category Theory for Programmers
- Categories for Software Engineering
- An Invitation to Applied Category Theory: Seven Sketches in Compositionality
- Life Itself: A Comprehensive Inquiry Into the Nature, Origin, and Fabrication of Life
- Category Theory for the Sciences
- Conceptual Mathematics and Literature: Toward a Deep Reading of Texts and Minds
- Diagrammatic Immanence: Category Theory and Philosophy
- Mathematical Mechanics: From Particle To Muscle
- Memory Evolutive Systems; Hierarchy, Emergence, Cognition
- Mathematical Structures of Natural Intelligence
- Sheaf Theory Through Examples
- Visual Category Theory
Q&A

Please send your feedback using the contact form on PatternDiagnostics.com
Thank you for attendance!