Lecture 6: Process's Address Space

Initial State, Management, and Hacking

Xin Liu

Florida State University

xliu15@fsu.edu

COP 4610 Operating Systems

https://xinliulab.github.io/FSU-COP4610-Operating-Systems/

Recap

Background:

- Linux builds the entire application program world from an initial process (state machine).
- Through `fork`, `execve`, and `exit`, we can create many child processes and execute them concurrently.

Recap

This Lecture:

- Based on our state machine model, a process's state consists of memory and registers.
- Registers are well-defined and can be examined using `info registers` of `gdb`. Try: 10_miniHello.s
- What is inside the "flat" address space of a process (0 to $2^{64} - 1$)?
- Can we "invade" another process's address space?

A Process’s Memory

A Fundamental (but Difficult) Question

Registers are easy to understand (observable using `gdb` + `info registers`).

Process State Model:

- What is ”a process’s memory”?

Step 1: Printing the address of `main`

```
#include <stdio.h>
int main() {
    printf("%p\n", main);
}
```

What happens?

- Prints the memory address where the function `main` begins.
- Matches with `objdump -d a.out`:

```
0000000000001149 <main>:
    1149: f3 0f 1e fa    endbr64
```

- Also matches with `info proc mappings` in `gdb`:

```
0x555555555149
```

Step 2: Reading bytes from `main`

```
int x = *(int*)main;
printf("%x\n", x);
```

What happens?

- Casts `main` (a function pointer) to an `int*`.
- Reads the first 4 bytes of machine code at `main`.
- Example output: `fa1e0ff3`
- Matches with `objdump -d a.out`:

```
0000000000001149 <main>:
    1149: f3 0f 1e fa    endbr64
```

Step 3: Accessing an arbitrary address

```
int *q = (void*) 0x12345678LL;
int y = *q;
printf("%x\n", y);
```

What happens?

- Tries to dereference memory at `0x12345678`.
- Process does not own that address.
- Results in a **Segmentation Fault**.

What type of pointer access would NOT cause a segmentation fault?

```
char *p = random();
*p;  // Load
*p = 1;  // Store
```

How to View the Address Space of a Linux Process?

(Curious: How is `pmap` implemented?)

Process Address Space

Manual: `man 5 proc`

- `/proc/[pid]/maps`
- `pmap [pid]`
- `gdb`+`info proc mappings`

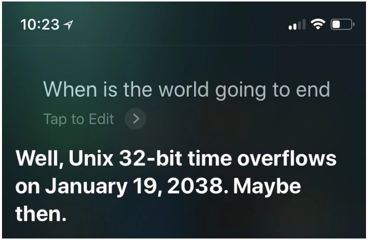
- Each segment of the process address space:
  - Address range and permissions (rwxsp)
  - Corresponding file: offset, dev, inode, pathname
  - The manual provides detailed explanations
- Verified with the information from `readelf -l`

What else can we find using `gdb miniHello` and `info proc mappings`?

- vvar (Virtual Variable Page)
- vdso (Virtual Dynamic Shared Object)
- vsyscall

vDSO: Fast time queries without syscalls

- A process has no concept of "time" by itself. It normally asks the OS via a system call.
- System calls are costly. Linux maps a read-only shared page and user-space helper functions (vDSO).
- Time data on this page is maintained by the kernel. User code can read it directly, no kernel trap.
- Examples that use vDSO when possible: `time(2)`, `gettimeofday(2)`.
- Try: `strace -e trace=gettimeofday ./vdso` to run 11_address_space/vdso.c
- If vDSO is used, you see no `gettimeofday` syscall.



VDSO and VVAR: Communication Mechanism

- We do not need syscalls.
- What we need is a communication channel between user space and the kernel.
- **Shared Memory Page:**
  - In some extreme cases, a shared page can be read and written by user programs.
- **Periodic Updates:** The OS periodically updates the shared page.
- **Synchronization:** Spinlocks are used to protect the integrity of read and write operations on this page.

Further Questions

`execve` creates the initial state of a process, including registers and segments of memory.

Can we *control* the output of `pmap`?

- Modify the size of the segment in memory
  - e.g., `malloc` to change the stack size
  - `gdb` + `inferiors` to check the process
  - `!pmap [PID]` to check the stack size
- Allocate large arrays on the stack...

Managing Process Address Space

Perspective from the State Machine:

- Address space = memory segments with access permissions
  - Does not exist (inaccessible)
  - Exists but inaccessible (read/write/execute not allowed)
- Management: Add/Remove/Modify a segment of accessible memory

Question: What kind of system calls would you provide?

Memory Mapping System Calls

- Dynamically add, remove, or modify a region of a process's virtual memory.
- Two common mapping types:
  1. **Anonymous mapping**: `MAP_ANONYMOUS`, not backed by a file.
  2. **File-backed mapping**: requires a **fd** (File Descriptor). Maps file contents into the process's address space. Widely used for loaders, databases, and zero-copy I/O.

Memory Mapping System Calls

```
// Create and remove mappings
void *mmap(void *addr, size_t len, int prot, int flags,
     int fd, off_t off);
int    munmap(void *addr, size_t len);

// mprotect can change access rights (read, write,
    execute) of an existing mapping.
int    mprotect(void *addr, size_t len, int prot);
```

Example (anonymous): `mmap(NULL, len, PROT_READ|PROT_WRITE, MAP_ANONYMOUS|MAP_PRIVATE, -1, 0);`

Example (file-backed): `fd = open("data.bin", O_RDONLY);`

`mmap(NULL, len, PROT_READ, MAP_PRIVATE, fd, 0);`

Example 1: Allocating a Large Memory Space

- Instantaneous memory allocation
  - `mmap`/`munmap` provides the mechanism for `malloc`/`free`.
  - `libc`'s `malloc` directly invokes `mmap` for large allocations.
- Consider using `strace`/`gdb` to observe the behavior.

Example 2: Everything is a File

- Map a large file and access only part of it.

```
with open('/dev/sda', 'rb') as fp:
    mm = mmap.mmap(fp.fileno(),
                   prot=mmap.PROT_READ, length=128
<< 30)
    hexdump.hexdump(mm[:512])
```

Hacking Address Spaces

How to Make Mods for Games

Game Cheat 1: Hacking Address Spaces

- A process (state machine) executes on a "dispassionate instruction machine."
  - The state machine is a self-contained world.
  - **But what if a process is allowed to access the address space of another process?**
    - It implies the ability to observe or modify another program's behavior.
    - Sounds pretty cool!

Examples of "invading" address spaces:

- Debugging (gdb)
  - `!ps` or `!pmap` in `gdb a.out`
  - *gdb* allows inspecting and modifying the state of a program.
- Profiling (perf)
  - Tools like *perf* help analyze the performance bottlenecks of a program.

How gdb Interacts with ELF and Address Spaces

- **How gdb Uses ELF Files**
  - ELF contains function symbols, variable locations, and debugging metadata.
  - *gdb* reads the ELF file to get debugging symbols.
- **Accessing Another Process's Address Space**
  - *gdb* can attach to a running process.
  - It allows inspecting and modifying memory and registers.
  - Achieved through system calls (e.g., `ptrace` in Linux).

Key Concept: The OS as an API and Object

- The OS provides APIs that allow a process to debug another.
- Can these APIs ensure security and prevent unauthorized access?

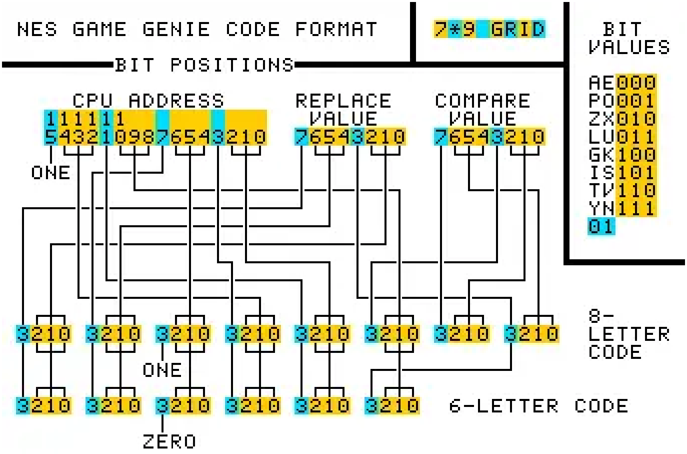
Physical Intrusion into Address Spaces

Golden Finger: Directly Manipulate Physical Memory

- Sounds distant, but it was achievable during the ”cartridge” era!

- Today, we have tools like Debug Registers and Intel Processor Trace.
- These tools assist systems in ”legally intruding” into address spaces.

Game Genie: A Look-up Table (LUT)



- Simple yet elegant: When the CPU reads address *a* and retrieves *x*, replace it with *y*.
- Technical Notes (Patents, How did it work?)

Game Genie as a Firmware

Game Genie as a Boot Loader

- Configures the Look-Up Table (LUT) and loads the cartridge code.
- Functions like a simple "Boot Loader."

The Blurring Boundaries Between I/O Dev and Comp

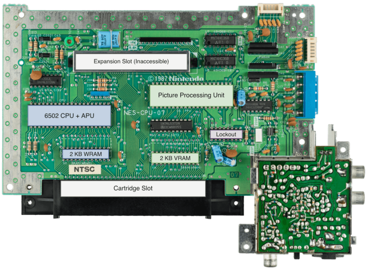
- How can we have CPUs for various tasks?

Example: Displaying Patterns

```
#include <stdio.h>

int main() {
    int H = 10;
    int W = 10;

    for (int i = 1; i <= H;
     i++) {
    for (int j = 1; j <= W;
     j++)
        putchar(j <= i ? '*
   ' : ' ');
    putchar('\n');
    }
}
```



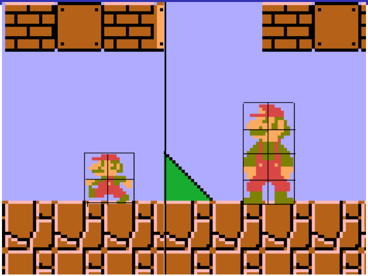
Nintendo Entertainment System (NES) Motherboard

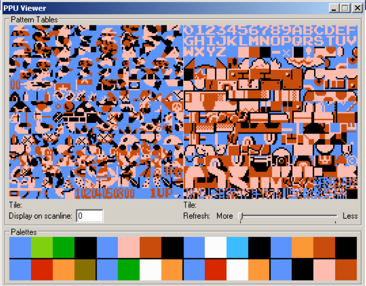
The Challenge of Performance:

NES: 6502 @ 1.79MHz; IPC = 0.43

- Screen resolution: 256 x 240 = 61K pixels (256 colors)
- 60FPS $\Rightarrow$ Each frame must complete within 10K instructions
  - How to achieve 60Hz with limited CPU computing power?

NES Picture Processing Unit (PPU)





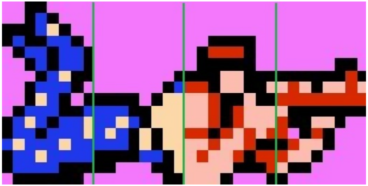
The **CPU** only describes the arrangement of 8x8 tiles

- The background is part of a larger image
  - No more than 8 foreground tiles per line
- The PPU completes the rendering
  - A simpler type of ”CPU”
- Enjoy!

7	6	5	4	3	2	1	0	
|	|	|	|	|	|	+	+	Palette
|	|	|	|	+	+	-	-	Unimplemented
|	|	+	-	-	-	-	-	Priority
|	+	-	-	-	-	-	-	Flip horizontally
+	-	-	-	-	-	-	-	Flip vertically

Why do the characters in KONAMI's Contra adopt a prone position with their legs raised?

- Video



Better 2D Game Engine

What if we have more powerful processors?

- The NES PPU is essentially a "tile-based" system aligned with the coordinate axes.
  - It only requires addition and bitwise operations to work.
- Greater computational power = More complex graphics rendering.

2D Graphics Accelerator: Image "Clipping" + "Pasting"

- Supports rotation, material mapping (scaling), post-processing, etc.

Achieving 3D

- Polygons in 3D space are also polygons in the visual plane.
  - Thm. Any polygon with n sides can be divided into $n - 2$ triangles.

Simulated 3D with Clipping and Pasting

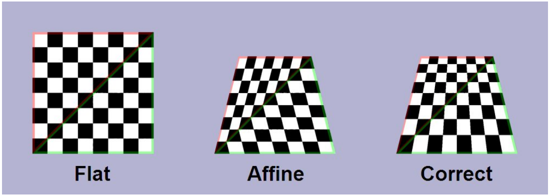
GameBoy Advance

- 4 background layers; 128 clipping objects; 32 affine objects
  - CPU provides the description; GPU performs the rendering (acting as a ”program-executing” CPU)

V-Rally; Game Boy Advance, 2002

Triangles in 3D space require correct rendering

- Modeling at this stage includes:
    - Geometry, materials, textures, lighting, etc.
- Most operations in the rendering pipeline are massively parallel



"Perspective correct" texture mapping (Wikipedia)

Solution: Full PS (Post-Processing)

Example: GLSL (Shading Language)

- Enables ”shader programs” to execute on the GPU
  - Can be applied at various rendering stages: vertex, fragment, pixel shaders
  - Functions as a ”PS” program to calculate lighting changes for each part
    - Global illumination, reflections, shadows, ambient occlusion, etc.

Modern GPU: A General-Purpose Computing Device

A complete multi-core processing system

- Focuses on massively parallel similar tasks
  - Programs are written in languages like OpenGL, CUDA, OpenCL, etc.
- Programs are stored in memory (video memory)
  - `nvcc` (LLVM) compiles in two parts
    - Main: Compiles/links to a locally executable ELF
    - Kernel: Compiles to GPU instructions (sent to drivers)
- Data is also stored in memory (video memory)
  - Can output to video interfaces (DP, HDMI, ...)
  - Can also use DMA to transfer to system memory

Example: PyTorch and Deep Learning

What is a "Deep Neural Network"?
How do we "train"?

- Requires computationally intensive tasks

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512), nn.ReLU(),
            nn.Linear(512, 512), nn.ReLU(),
            nn.Linear(512, 10), nn.ReLU(),
        )
...
model = NeuralNetwork().to('cuda')
```

Dark Silicon Age and Heterogeneous Computing

Many components can perform the ”same task”

- The key is to choose the component with the most suitable power/performance/time trade-off!

Examples of Components:

- CPU, GPU, NPU, DSP, DSM/RDMA

Game Cheat 2: Expanding Game Exploration

Address Space: Where is the "Gold"?

- Includes dynamically allocated memory, with varying addresses every time.
- Insight: **Everything is a state machine.**
  - By observing the trace of state changes, you can identify the valuable addresses.

Search + Filter

- Enter the game: `exp = 4610`.
- Perform an action: `exp = 5370`.
- Match the memory locations where `4610 → 5370` occurs.
  - These memory locations are very few.
- Once found, you're satisfied! Demo

Repeating Fixed Tasks at Scale (e.g., 1 second, 5370 shots)

Enjoy!

- Example shown demonstrates automating repetitive actions with precise timing.
- Such tools enable consistent execution of predefined tasks without manual intervention.

Implementing Precision Automation

Sending Keyboard/Mouse Events to Processes

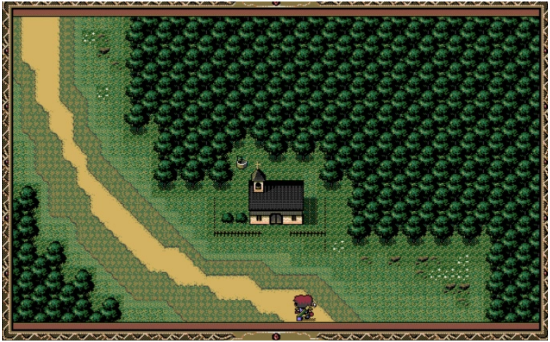
- Developing Drivers (e.g., custom keyboard/mouse drivers)
- Leveraging System Window Manager APIs
  - xdotool: Useful for testing, including plugins for VSCode
  - ydotool
  - evdev: Commonly used for live streaming or scripting key sequences

Application in 2024: Implementing AI Copilot Agent

- Automating workflows: Text/Image Capture → AI Analysis → Execute Actions

Adjusting the Game's Logic Update Speed

- For example, a certain mysterious company's game is so slow that both map traversal and combat feel unbearable.
- The gaming industry today has become so competitive that if a new player's progression path isn't smooth, the game will be heavily criticized.



Principle of Speed Modification: Theory

Program = State Machine

- ”Compute instructions” are inherently unaware of time.
- Using count for timing can lead to issues where the game becomes unplayable on faster machines.
- **Syscalls** are the only way for a program to perceive time.

”Hijacking” Time-Related Syscall/Library Functions

- `gettimeofday`, `sleep`, `alarm`
- Replacing the system call’s code with our own code allows us to alter the program’s perception of time.
- Similar to adjusting a clock to make it appear faster or slower.

Code Injection: Hooking Functions with Code

- Using a piece of code to **hook** the execution of a function.
- Allows tampering with the program's logic and gaining control.

Hooking in Game Cheats

How Hooking is Used in Game Cheats

- Hooking intercepts and modifies game functions to manipulate game behavior.
- Commonly used in ESP (Extra Sensory Perception) cheats, Aimbots, and Wallhacks.

Methods of Hooking:

- DirectX/OpenGL Hooking: Modifies rendering functions like `D3D11Present` to draw ESP overlays.
    - System Call Hooking: Alters time-related functions (e.g., `gettimeofday`) to manipulate game physics.
    - Memory Hooking: Modifies in-game variables (e.g., `hp = 9999`) in real-time.

Example: ESP Wallhack

- Hooks rendering APIs to bypass depth checks.
- Modifies enemy rendering to make them visible through walls.

The Essence of "Hijacking Code" is Debugger Behavior

- A game is also a program, and a state machine.
- A cheat tool is essentially a `gdb` designed specifically for the game.

Example: Locking Health Points

- Create a thread to spin and modify:

```
while (1) hp = 9999;
```

- However, conditions like `hp < 0` (e.g., instant death) may still occur.
- Solution: Patch the code that checks `hp < 0` (soft dynamic updates).

Code Injection (cont'd)

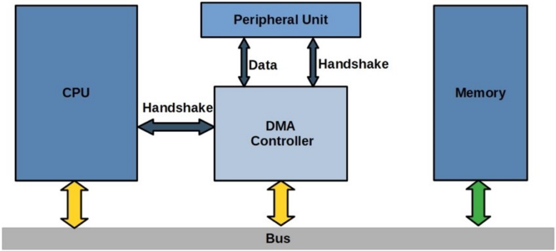
"I heard that Devil Fruits are the incarnations of sea demons. Eating one grants devil-like abilities, but in return, the sea will reject the user."

Enjoy!

Game Cheat 5: DMA

DMA (Direct Memory Access): A dedicated CPU for executing "memcpy" operations

- Adding a general-purpose processor is too costly
- A simple controller is a better solution
- Supported types of memcpy:
  - memory → memory
  - memory → device (register)
  - device (register) → memory
    - Practical implementation: Directly connect the DMA controller to the bus and memory
    - Intel 8237A



More on DMA

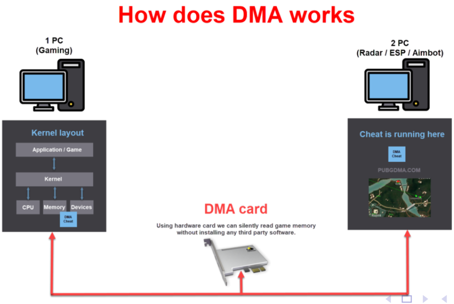
- CPU is not involved in copying data
- A process cannot access in-transit data
- PCI bus supports DMA
  - Handles a large number of complex tasks

Why Does DMA Cheating Exist?

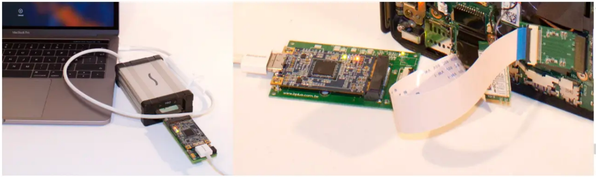
- Modern anti-cheat methods rely on detecting memory modifications.
- Kernel-level anti-cheat software (e.g., Vanguard, BattleEye) prevents direct process memory access.
- Reading memory via software (e.g., external cheats) is highly detectable.
- DMA bypasses all software-based detection because it directly accesses memory **without CPU intervention**.

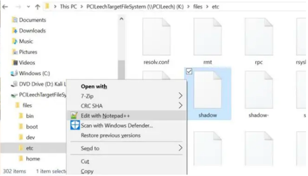
How DMA Cheats Work

1. A second computer with a **DMA capture card** is used.
2. The card is installed in the main gaming PC via **PCIe**.
3. The DMA card **reads game memory** and extracts relevant data (e.g., player positions).
4. The extracted data is sent to the second PC for processing.
5. The second PC renders an **ESP (extra-sensory perception) overlay**, giving the player an unfair advantage.
6. Since the main PC runs no cheat software, anti-cheat solutions fail to detect it.

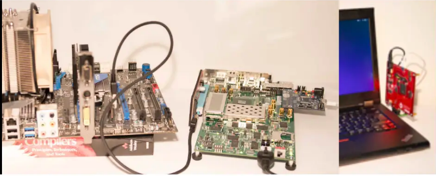


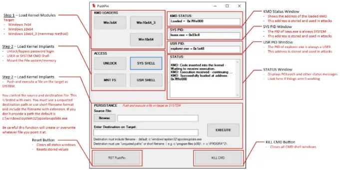
Cheat Example: PCILeech






```
 Current Action: Dumping Memory
 Access Mode:    KMD (kernel module assisted DMA)
 Progress:       8678 / 8678 (100%)
 Speed:          173 MB/s
 Address:        0x000000021E600000
 Pages read:     2050967 / 2221568 (92%)
 Pages failed:   170601 (7%)
Memory Dump: Successful.
```





Why Is It Hard to Detect?

- **No modification of game memory** (only reading).
- **No injected code**, unlike traditional hacks.
- **Appears as a legitimate PCIe device**, making it difficult to blacklist.

Current Anti-Cheat vs. DMA

Anti-Cheat Method	Effectiveness Against DMA
Signature Scanning	Ineffective (DMA is external)
Kernel-Level Hooks	Ineffective (DMA doesn't use system calls)
Code Integrity Checks	Ineffective (No code modification)
Behavior Analysis	Partially Effective (Detecting unnatural move

Future of Anti-DMA Methods

- **Hardware-based solutions**: Restricting PCIe device access via BIOS/firmware.
- **AI-based detection**: Tracking suspicious player behavior.
- **Encrypted memory**: Preventing DMA from extracting useful data.
- Currently, **no effective universal countermeasure exists.**

Takeaways: On Cheats and Code Injection

Cheats Can Also Serve "Good" Purposes:

- Live Kernel Patching: Enable "hot" updates without stopping the system.
- Techniques, whether in computing systems, programming languages, or artificial intelligence, are meant to provide benefits to humans — for example, debugging tools and even cheats can help game developers or testers improve performance.

Ethics of Technology:

- Strong technology always has both "good" and "bad" applications.
- Any misuse of technology to harm others is a violation of integrity. Similarly, if cheats are used for malicious purposes in games, we should also consider the moral implications and use tools responsibly.