Introduction to Naming

COS 316 Lecture 5

Amit Levy
<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.princeton.edu">www.princeton.edu</a></td>
<td>hostname</td>
</tr>
<tr>
<td><a href="mailto:aalevy@cs.princeton.edu">aalevy@cs.princeton.edu</a></td>
<td>email</td>
</tr>
<tr>
<td>r1</td>
<td>ARM register name</td>
</tr>
<tr>
<td>main</td>
<td>procedure name</td>
</tr>
<tr>
<td><a href="http://www.princeton.edu/news">http://www.princeton.edu/news</a></td>
<td>URL</td>
</tr>
<tr>
<td>(609) 258-3000</td>
<td>phone number</td>
</tr>
<tr>
<td>140.180.223.42</td>
<td>IP address</td>
</tr>
</tbody>
</table>

Systems manipulate and manage resources either by value or by name

- (but values are also usually just names at a lower level)
Why use names?
Why use names?

- Sharing
- Retrieval
- User friendly
- Hiding
- Indirection
Choosing a naming scheme is often the first step in designing a system
Naming Schemes Components

- Set of all possible names (i.e. the *namespace*)
- Set of all possible values
- Allocation algorithm: creates a new mapping
- Translation algorithm: translates a name to a value
Example Naming Scheme: UNIX Pipes

Names
- (PID, non-negative integer)
  - i.e. names are local to each process
- Shared with other file descriptors

Values
- (in-kernel buffer, in/out?)
- Buffer needs to have associated data structures (e.g. semaphore, cursor index, etc)
Example Naming Scheme: UNIX Pipes

Allocation
Invariant: keep track of max file descriptor per process

int pipe(int pipefd[2])

1. Allocate an in-kernel buffer: newbuf = new_kernel_pipe(...);

2. Increment max file descriptor by 2 and use:

(PID, newmaxfd − 1) : (newbuf, in) (PID, newmaxfd) : (newbuf, out)
Example Naming Scheme: UNIX Pipes

Allocation
Invariant: keep track of max file descriptor per process

int dup(int oldfd)

1. Increment max file descriptor by 1 and use:

\[(PID, \text{newmaxfd}) : resolve(PID, oldfd)\]
**Example Naming Scheme: UNIX Pipes**

**Allocation**
Invariant: keep track of max file descriptor per process

```c
int dup(int oldfd)
```

1. Increment max file descriptor by 1 and use:

```
(PID, newmaxfd) : resolve(PID, oldfd)
```

Alternative?
Example Naming Scheme: UNIX Pipes

**Allocation**
Invariant: keep track of max file descriptor per process

```c
int dup(int oldfd)
```

1. Increment max file descriptor by 1 and use:

```
(PID, newmaxfd) : resolve(PID, oldfd)
```

Alternative?

```
(PID, newmaxfd) : (λ → resolve(PID, oldfd))
```
Example Naming Scheme: UNIX Pipes

**Translation**  
Maintain a table per process

<table>
<thead>
<tr>
<th>FD</th>
<th>Pipe</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>(buf1, in)</td>
</tr>
<tr>
<td>4</td>
<td>(buf1, out)</td>
</tr>
<tr>
<td>5</td>
<td>(buf1, in)</td>
</tr>
<tr>
<td>12</td>
<td>(buf2, out)</td>
</tr>
</tbody>
</table>
Virtual Memory

Figure 1: Virtual Memory
What does virtual memory give us?

• Isolation
• Flexibility in memory management, e.g., defragment memory dynamically
• Overprovisioning
• Abstraction over storage media
What does virtual memory give us?

• Isolation

• Flexibility in memory management
  • E.g. defragment memory dynamically

• Overprovisioning

• Abstraction over storage media
Virtual Memory as a Naming Scheme

- Names?
- Values?
- Allocation?
- Translation?
Virtual Memory: Names

Pointer-sized (e.g. 32-bit or 64-bit) addresses and process identifiers (\( P \_I \_D \), virtual address) e.g. (3487, 0xdeadbeef)
Virtual Memory: Names

Pointer-sized (e.g. 32-bit or 64-bit) addresses and process identifiers

\[(PID, \text{virtual\_address})\]

e.g.

\[(3487, 0xdeadbeef)\]
Virtual Memory: Values

Could be any of:

- Physical memory address (i.e. 32-bit or 64-bit address up to size of RAM)
- On-disk file and offset in file
- Some hardware registers (e.g. a network card configuration registers)
- Remote memory
Virtual Memory: Values

Could be any of:

- Physical memory address (i.e. 32-bit or 64-bit address up to size of RAM)
- On-disk file and offset in file
- Some hardware registers (e.g. a network card configuration registers)
- Remote memory
Virtual Memory: Allocation

int sbrk(intptr_t increment)

- Name is given by user—each 4KB “page” of virtual memory between old break and new break
- For values, keep a free list of physical 4KB memory pages
- Add mapping to “page table”—a data structure understood by the virtual memory hardware that maps virtual addresses to physical addresses
int sbrk(intptr_t increment)

- Name is given by user—each 4KB “page” of virtual memory between old break and new break
- For values, keep a free list of physical 4KB memory pages
- Add mapping to “page table”—a data structure understood by the virtual memory hardware that maps virtual addresses to physical addresses
Virtual Memory: Translation

Figure 2: Two-level page table structure in x86
Virtual Memory: Translation

Figure 2: Two-level page table structure in x86

*) 32 bits aligned to a 4-KByte boundary
Virtual Memory: Translation

Figure 3: Virtual-to-physical translation
Virtual Memory: Translation

For this to work, the OS needs to do some housework when context switching:

- Set CR3 register to point to process’s page table
- Invalidate the TLB
  - Mark entire TLB as invalid—simple but can cause unnecessary slow down
  - Associate process IDs with each TLB entry
Virtual Memory: Alternative naming schemes

- Segmentation
  - Coarser grain

- Single shared address space (identity mapping)
  - Still protect with hardware, better performance but less flexibility

- Swap out all memory for one process at a time (original UNIX)

- Language-based memory isolation - runtime maps variables to physical address
  - Generally slower to translate compared to hardware paging
Takwaways: Virtual Memory

• Naming scheme influences:
  • Performance
  • Resource allocation flexibility
  • Isolation

• Going from design to practical implementation can take a long time
1. Two types of file systems:
   - The UNIX file system
   - Content addressable storage: Git

2. Naming in Networking (Prof. Rexford)

Assignment 2
An HTTP request routing library.

Why? URL paths name resources (pages, form handlers, etc) on a web server.