The Process Model (1)

L41 Lecture 3, Part 2: Processes In Practice
Prof. Robert N. M. Watson
2022-2023
Process address space: dd(1)

- Inspect dd process address space with `procstat -v`

```
root@rp4-000:~ # procstat -v 20921

<table>
<thead>
<tr>
<th>PID</th>
<th>START</th>
<th>END</th>
<th>PRT</th>
<th>RES</th>
<th>PRES</th>
<th>REF</th>
<th>SHD</th>
<th>FLAG</th>
<th>TP</th>
<th>PATH</th>
</tr>
</thead>
<tbody>
<tr>
<td>20921</td>
<td>0x200000</td>
<td>0x203000</td>
<td>r---</td>
<td>3</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>CN---</td>
<td>vn</td>
<td>/bin/dd</td>
</tr>
<tr>
<td>20921</td>
<td>0x217000</td>
<td>0x217000</td>
<td>r-x</td>
<td>5</td>
<td>8</td>
<td>3</td>
<td>0</td>
<td>CN---</td>
<td>vn</td>
<td>/bin/dd</td>
</tr>
<tr>
<td>20921</td>
<td>0x227000</td>
<td>0x227000</td>
<td>r---</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>C----</td>
<td>vn</td>
<td>/bin/dd</td>
</tr>
<tr>
<td>20921</td>
<td>0x237000</td>
<td>0x237000</td>
<td>rw-</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-----</td>
<td>df</td>
<td></td>
</tr>
<tr>
<td>20921</td>
<td>0x4023c000</td>
<td>0x4023c000</td>
<td>r--</td>
<td>6</td>
<td>27</td>
<td>51</td>
<td>0</td>
<td>CN---</td>
<td>vn</td>
<td>/libexec/ld-elf.so.1</td>
</tr>
<tr>
<td>20921</td>
<td>0x402d4000</td>
<td>0x402d4000</td>
<td>r-x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>CN---</td>
<td>vn</td>
<td>/libexec/ld-elf.so.1</td>
</tr>
<tr>
<td>20921</td>
<td>0x402e5000</td>
<td>0x402e5000</td>
<td>rw-</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>C----</td>
<td>vn</td>
<td>/libexec/ld-elf.so.1</td>
</tr>
<tr>
<td>20921</td>
<td>0x402e7000</td>
<td>0x402e7000</td>
<td>rw-</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-----</td>
<td>df</td>
<td></td>
</tr>
</tbody>
</table>

r: read       x: execute     D: Downward growth   S: Superpage
w: write      C: Copy-on-write N: Needs copy
```

```
ELF binaries

- UNIX: Executable and Linkable Format (ELF)
- Mac OS X/iOS: Mach-O; Windows: PE/COFF; same ideas
- Inspect `dd` ELF program header using `objdump -p`:

```
root@rpi4-000:~ # objdump -p /bin/dd
/bin/dd: file format elf64-littleaarch64
```

```
Program Header:
PHDR off 0x0000000000000040 vaddr 0x0000000000000040 paddr 0x0000000000000040 align 1
filesz 0x0000000000000268 memsz 0x0000000000000268 flags r--
INTERP off 0x00000000000002a8 vaddr 0x00000000000002a8 paddr 0x00000000000002a8 align 2**0
  filesz 0x0000000000000015 memsz 0x0000000000000015 flags r--
LOAD off 0x0000000000000000 vaddr 0x0000000000000000 paddr 0x0000000000000000 align 2**16
  filesz 0x00000000000034a4 memsz 0x00000000000034a4 flags r-x
  DYNAMIC off 0x00000000000001a8 vaddr 0x00000000000001a8 paddr 0x00000000000001a8 align 2**16
  filesz 0x00000000000001a8 memsz 0x00000000000001a8 flags rw-
  RELRO off 0x000000000000063e0 vaddr 0x000000000000063e0 paddr 0x000000000000063e0 align 2**0
  filesz 0x00000000000001e8 memsz 0x00000000000001e8 flags r--
```

Actual loaded content
Virtual memory (quick but painful primer)

• **Memory Management Unit (MMU)**
  • Transforms virtual addresses into physical addresses
  • Memory is laid out in **virtual pages** (4K, 2M, 1G, ...)
  • Control available only to the supervisor (historically)
  • Software handles failures (e.g., store to read-only page) via **traps**

• **Page tables**
  • SW-managed **page tables** provide virtual-physical mappings
  • Access permissions, page attributes (e.g., caching), dirty bit
  • Various configurations + traps implement BSS, COW, sharing, ...

• **Translation Look-aside Buffer (TLB)**
  • Hardware cache of entries – avoid walking pagetables
  • Content Addressable Memory (CAM); 48? 1024? entries
  • TLB **tags**: entries **global** or for a specific **address-space ID (ASID)**
  • Software- vs. hardware-managed TLBs

• **Hypervisors and IOMMUs**:
  • I/O performs **direct memory access (DMA)** via virtual address space
Virtual memory (quick but painful primer)

- A fixed partition between user and kernel address space makes checks quick and easy to implement.
- On some architectures (e.g., ARMv8-A), this point is configurable.
- The kernel also needs substantial address space. It’s a squeeze in 32 bits, and fine with 64.

- Pages will be zero filled on demand – e.g., for BSS or heap memory.

Memory mappings from program binaries include:
- Read-write (COW) demand-zeroed pages (BSS)
- Read-write (COW) mappings of data
- Read-execute mappings of program text (COW)

- Kernel address space is also managed using the MMU.
- Unified global kernel address space (with certain exceptions).
- Kernel mappings may “borrow” pages from userspace.
- “Direct map region” provides quick, efficient mapping of physical addresses.

The kernel will opportunistically promote and demote superpages.
This requires physical, and not just virtual, alignment and contiguity.
Role of the run-time linker (rtld)

- **Static linking**: program, libraries linked into one binary
  - Process address space laid out (and fixed) at compile time

- **Dynamic linking**: program, libraries in separate binaries
  - Shared libraries avoid code duplication, conserving memory
  - Shared libraries allow different update cycles, ABI ownership
  - Program binaries contain a list of their **library dependencies**
  - The run-time linker (rtld) loads and links libraries
  - Also used for plug-ins via `dlopen()`, `dlsym()`

- Three separate but related activities:
  - **Load**: Load ELF segments at suitable virtual addresses
  - **Relocate**: Rewrite **position-dependent code** to load address
  - **Resolve symbols**: Rewrite inline/PLT addresses to other code

- The run-time linker also plays a role in debugging
  - Its internal state is inspected and understood by the debugger
Starting a binary (and dependencies)

- When the `execve` system call starts the new program:
  - ELF binaries name their **interpreter** in ELF metadata
  - Kernel maps `rtld` and the application binary into memory
  - Userspace starts execution in `rtld`
  - `rtld` loads and links dynamic libraries
  - `rtld` runs library and application binary constructors
  - `rtld` calls `main()`

- Optimisations:
  - **Lazy binding**: don’t resolve all function symbols at load time
  - **Prelinking**: relocate, link in advance of execution
  - Difference is invisible – but surprising to many programmers

```bash
root@rpi4-000:~ # ldd /bin/dd
/bin/dd:
    libutil.so.9 => /lib/libutil.so.9 (0x402a3000)
    libc.so.7 => /lib/libc.so.7 (0x402e7000)
```

Arguments and ELF auxiliary arguments

• C-program arguments are argc, argv[], and envv[]:

```
root@rpi4-000:~ # procstat -c 20921
  PID COMM          ARGS
20921 dd           dd if=/dev/zero of=/dev/null bs=1k
```

• The run-time linker also accepts arguments from the kernel:

```
root@rpi4-000:~ # procstat -x 20921
  PID COMM AUXV         VALUE
20921 dd  AT_PHDR      0x200040
20921 dd  AT_PHENT     56
20921 dd  AT_PNUM      11
20921 dd  AT_PAGESZ    4096
20921 dd  AT_FLAGS     0
20921 dd  AT_ENTRY     0x213148
20921 dd  AT_BASE      0x40236000
20921 dd  AT_EHDRFLAGS 0
20921 dd  AT_EXCPATH   0xffffffffffed8
20921 dd  AT_OSRELDATE 1300138
20921 dd  AT_CANARY    0xffffffffef98
20921 dd  AT_CANARYLEN 64
20921 dd  AT_NCPUS     4
20921 dd  AT_PAGESIZES 0xfffffffffff80
20921 dd  AT_PAGESIZESLEN 24
20921 dd  AT_TIMEKEEP  0xffffffffff1c0
20921 dd  AT_STACKPROT NONEXECUTABLE
20921 dd  AT_HWCAP     0x83
20921 dd  AT_HWCAP2    0
20921 dd  AT_BSDFLAGS  0x1
20921 dd  AT_ARGC      4
20921 dd  AT_ARGV      0xfffffffff0a68
20921 dd  AT_ENVC      24
20921 dd  AT_ENVV      0xfffffffff0a90
20921 dd  AT_PS_STRINGS 0xfffffffff1e0
```

- Address of binary’s ELF program header
- Entry address for binary
- Base address of binary (or rtld if used)
- Command-line arguments and environment above stack
Wrapping up

• In this lecture, we have talked about:
  • The basics and history of the process model
  • A few gory implementation details

• Our next lecture, also on the process model, will explore:
  • Traps and system calls
  • Ideas about isolation, security, and reliability
  • More gory details of the VM system

• Readings for the next lecture:
  • Paper - Navarro, et al. 2002. (L41 only)