System Programming in Linux

Reda Maher

Process Overview

Processes and Programs

A process is an instance of an executing program.

A program is a file containing a range of information that describes how to construct a process at run time.

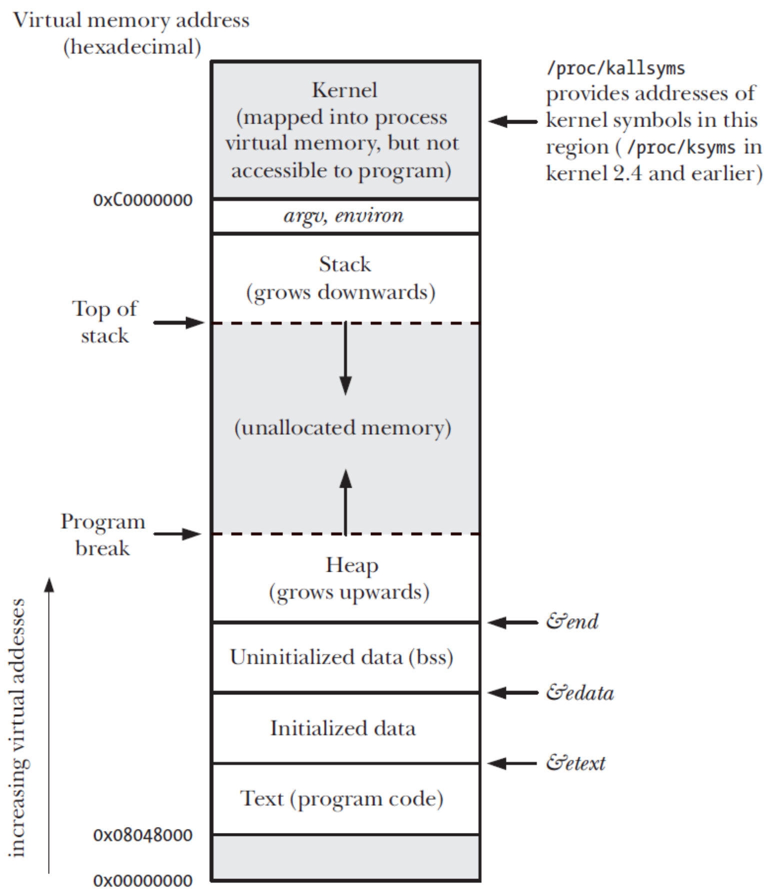
Information exists in a program file:

- Binary format identification (a.out, COFF, ELF).
- Machine-language instructions.
- Program entry-point address.
- Data (initial values and literal strings).
- Symbol and relocation tables.
- Shared-library and dynamic-linking information.

Process components

- **A process consists of:**
  - User-space memory containing program code and variables.
  - kernel data structures that maintain information about the state of the process:
    - Process IDs (/proc/sys/kernel/pid_max, getpid(), getppid()).
    - Virtual memory tables.
    - table of open file descriptors.
    - Resource usage and limits.
    - Others.

Memory Layout of a Process



Virtual memory address (hexadecimal)
Kernel (mapped into process virtual memory, but not accessible to program)
/proc/kallsyms provides addresses of kernel symbols in this region (/proc/ksyms in kernel 2.4 and earlier)
0xC0000000
argv, environ
Stack (grows downwards)
Top of stack
(unallocated memory)
Program break
Heap (grows upwards)
&end
Uninitialized data (bss)
&edata
Initialized data
&etext
Text (program code)
increasing virtual addresses
0x08048000
0x00000000

Process memory layout consists of:

- Text (read-only, sharable).
- Data.
- BSS (Block Started by Symbol).
- Stack.
- Heap (Program Break)

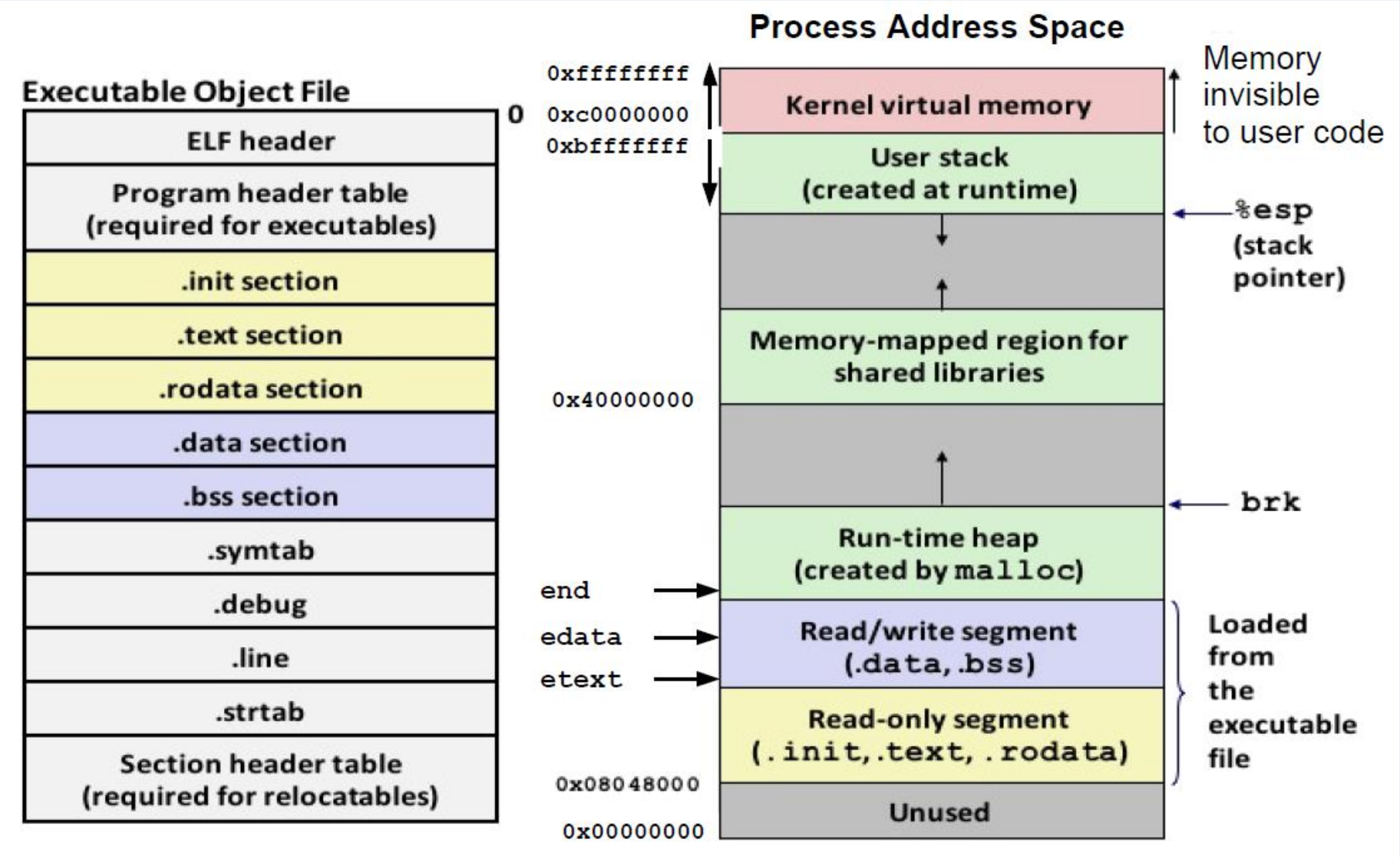
Notes:

→ Size command.

→ etext, edata, end.

→ Reentrant functions.

Execution View



Process Address Space

Executable Object File

0

ELF header

Program header table
(required for executables)

.init section

.text section

.rodata section

.data section

.bss section

.symtab

.debug

.line

.strtab

Section header table
(required for relocatables)

0xffffffff

0xc0000000

0xbfffffff

0x40000000

end

edata

etext

0x08048000

0x00000000

Kernel virtual memory

User stack
(created at runtime)

Memory-mapped region for
shared libraries

Run-time heap
(created by malloc)

Read/write segment
(.data, .bss)

Read-only segment
(.init, .text, .rodata)

Unused

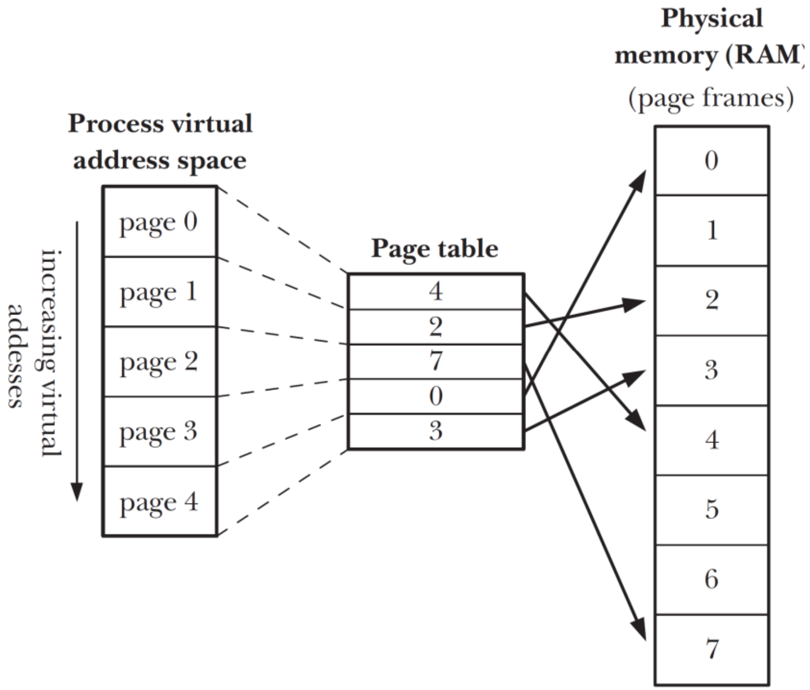
Memory
invisible
to user code

%esp
(stack
pointer)

brk

Loaded
from
the
executable
file

Virtual Memory Management



Locality of reference: a typical property of most programs.

- Spatial locality: reference memory address nearby.
- Temporal locality: access same address in near future.

We can execute a program while only part of its address space in RAM.

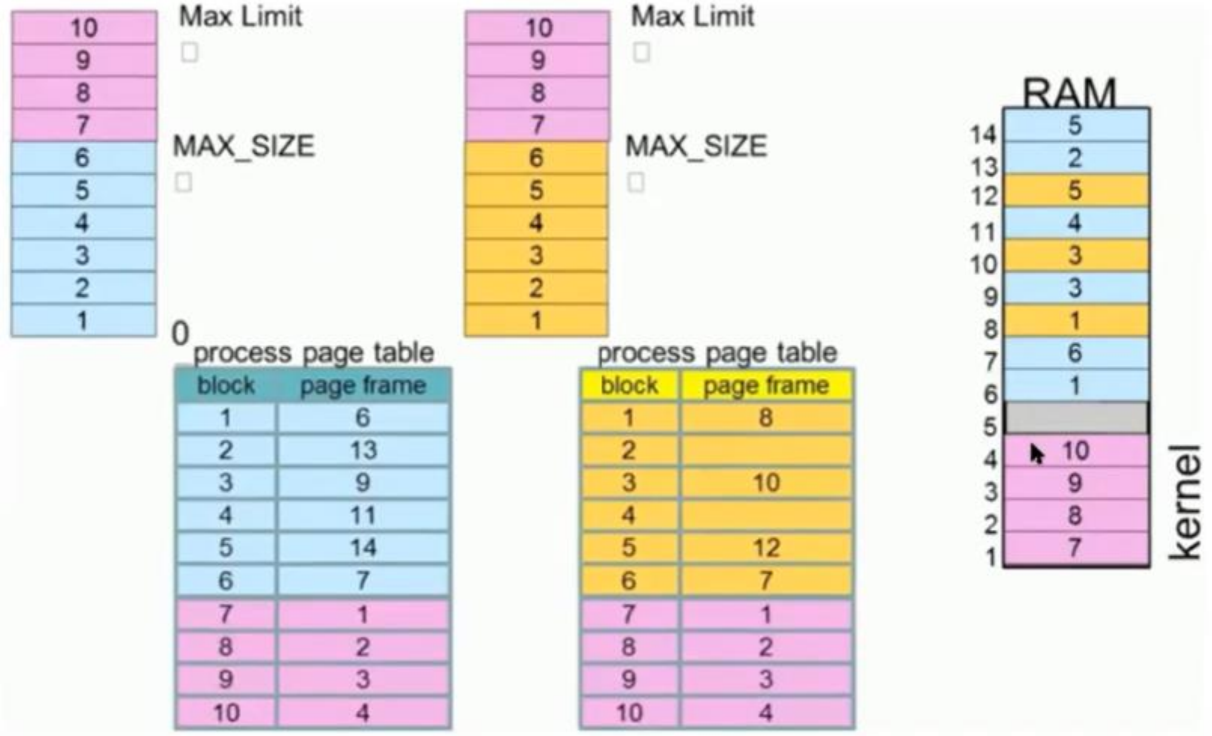
How does it work?

- Memory is divided into pages.
- Kernel maintains a page table for each process.
- Vary over program lifetime (stack, heap, shared memory).
- Page faults.
- Swap area.
- MMU.

Virtual Memory Advantages

- Process isolation (from other process and from the kernel).
- Memory sharing.
  - Executing the same program.
  - IPC.
- Memory Protection (read-only, execute-only, RW).
- Compiler and linker don’t need to be concerned with the physical layout.
- Loading programs faster.
- Better CPU utilization.

Virtual address



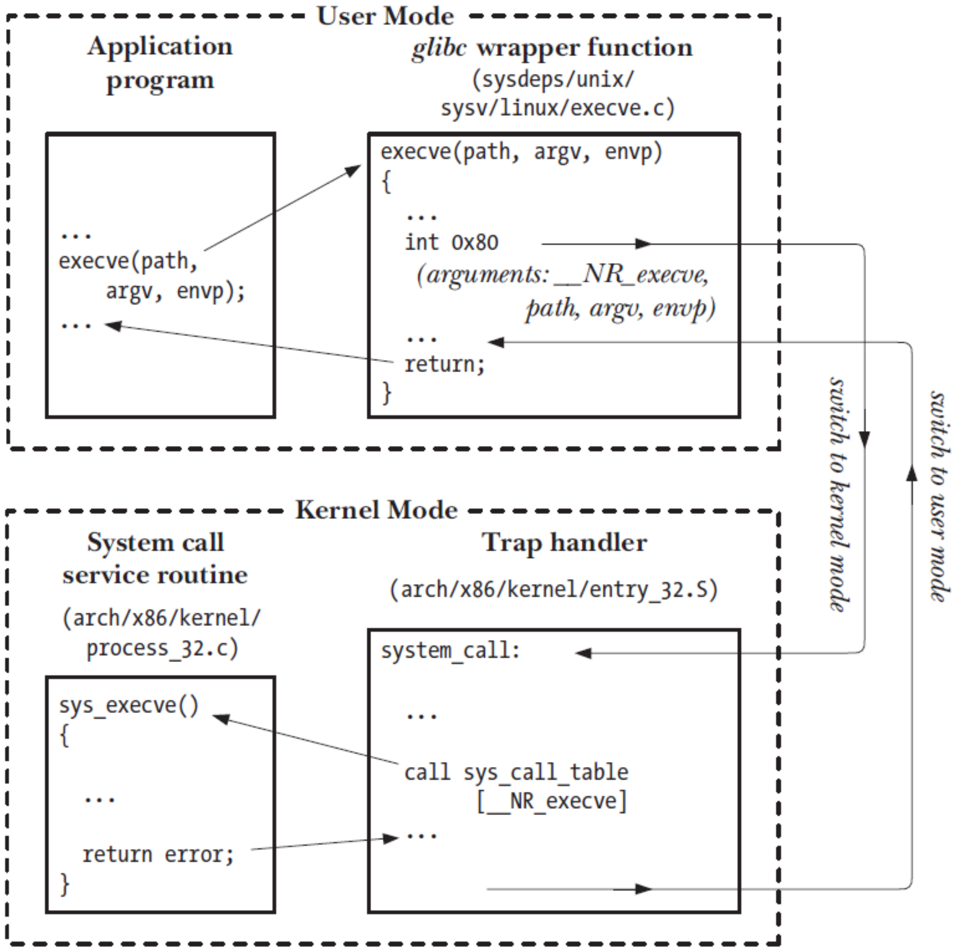
process page table

block	page frame
1	6
2	13
3	9
4	11
5	14
6	7
7	1
8	2
9	3
10	4

process page table

block	page frame
1	8
2	
3	10
4	
5	12
6	7
7	1
8	2
9	3
10	4

System Calls


User Mode
Application program
glibc wrapper function
(sysdeps/unix/sysv/linux/execve.c)
switch to kernel mode
switch to user mode
Kernel Mode
System call service routine
(arch/x86/kernel/process_32.c)
Trap handler
(arch/x86/kernel/entry_32.S)

```
...
execve(path,
    argv, envp);
...
```

```
execve(path, argv, envp)
{
  ...
  int 0x80
   (arguments: __NR_execve,
             path, argv, envp)
  ...
  return;
}
```

```
sys_execve()
{

  ...

  return error;
}
```

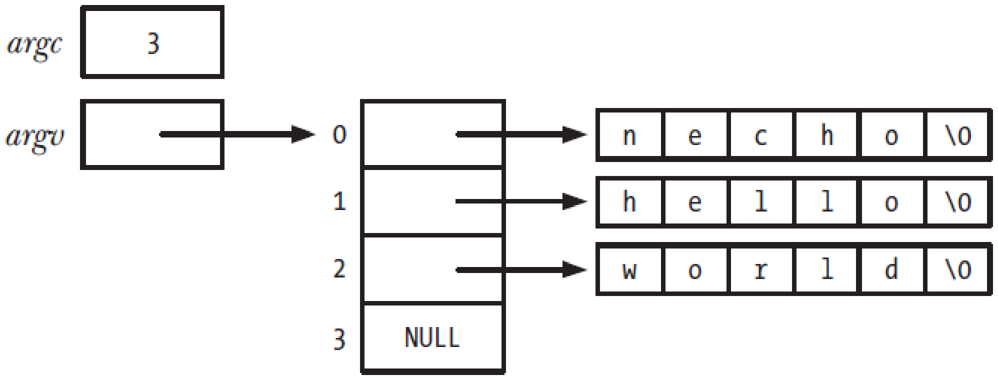
```
system_call:

  ...

  call sys_call_table
       [__NR_execve]
  ...
```

- **System call is** a controlled entry point into the kernel, allowing a process to request that the kernel perform some action on the process’s behalf.

Command line arguments



The kernel puts the array of the process arguments and their count on the process stack at the process execution.

Thank you

Reda Maher