1044414: Advanced Operating Systems and Virtualization **Spring 2018**

Lecture 12: April 17

Lecturer: Alessandro Pellegrini *Scribe: Anxhelo Xhebraj*

12.1 Traditional System V Initialization

System V

A set of text files known as run levels or targets configures systemd to run some services.

The first level is single user where only one user is allowed to enter the system, root.

The second level is multi user with no networking

The third multiuser full

Fourth not used and fallbacks to the third

others.

File conventions pp 95.

inittab tells the services needed to be started in a specific runlevel. Id identifies the entry, rl tells the run level, action tells the kind of command sent to the specific script, process is the process command line.

12.2 Systemd

Systemd acts as init therefore when a process becomes orphan (parent dies) then it becomes child of one of the systemd runlevels therefore if sysd crashes then everything else does.

A unit or target is a service or set of services needed to be launched at start.

In unit files there are multiple unit types. A service is a unit type for daemons.

The file is split in sections. Unit to which it belongs, description, require tells the dependencies needed to be executed before. Wants is like require but not necessary. Conflicts tells that this unit conflicts with another. Before means run it before the others.

Service section.

Install section. Wanted by determines when to start, which runlevel.

12.3 Program execution

Once init is spawned, thanks to systemd/runlevels a plethora of services/virtual terminals are launched.

As traditional unix when bash launches a program it forks itself and then performs an execve. execve is performed which will return in the case that the file is not a binary and therefore, the shebang will be checked in this case.

Bash will have performed a setjump before and through longjump we don't follow the return paths from nested function calls but directly go to the point of setjmp. Setjump saves only callee save registers.

The kernel level implementation of the fork is `do_fork` which is used when new userspace process must be created. Allocates a fresh PCB from the slab and setups kernel stack.

Flags allow to determine what resources are shared between the parent and child parent. Exec just changes the program file, wipes the memory of the calling process, loads the new file to memory, initializes the cpu state.

`linux_binprm` bridges the gap btw VFS and binary files. do execve is the only implementation of the exec* present in kernel code. Open exec navigates and opens the file.

12.3.1 Binary Handlers

Each program is organized according to a specific structure that carries a magic number put at the very beginning of files. load elf binary will load the program image in memory and setup the permissions according to the header.

Traditional file format was `a.out`. There were up to 7 section in this format. What is a section? In a program there can be code, data etc. and we would like them put together in a portion of the file. The exec header tells how large are each section and which sections there are. A text segment is put in a piece of memory that is put in executable memory. Relocation sections. Symbol table is used to identify for a given instruction the symbol that is referenced. The string table contains the symbols of the program. This file format was problematic since it was not good for cross-compiling, dynamic linking etc. Lack of iniitalizers and finalizers of programs. The object file format was then replaced with the Executable and Linkable Fromat (ELF). Debugging format called dwarf.

12.3.2 ELF

An elf file can be relocatable (created by compilers and assemblers that cannot be run). Executable. Shared object can be linked. Core file is the complete content of the program when it crashed.

There can be as many sections as we like. Segments tell which are the parts of the file and the mapping to sections is one to one or many to one etc. The section header in the file image tells where each section starts etc. The elf header tells were to find the section and program header, also the magic number to tell this is an elf file. Segments group sections into types of stuff that is found in sections (pieces of the file treated homogeneously).

struct Elf32 Ehdr. e entry holds the virual address of the first instruction to be executed. Program header is kept in e phof and section header kept in e shof. There are entries of the same size in the program header and section header describing each section/segment. String table index speeds up the resolution and lookup of linking.

By default there can't be data and instructions in the same section. Also constant variables are in another segment with respect to normal variables. Symbols are identified by their offset from the start of the section they live in.

Section header, there is the name (index in the string table), type and flags. Size tells the length of the section.

Types and flags in the section header. Progbits code, data, debug info. Progbits combined with execintrs tells if the section is code. NOBITS combined with ALLOC tells the loader that even if the section contains no information in the image a size of ALLOC should be initialized of zeros in memory.

If the symbol refers to the first null it has no name, while if is another null then its name is empty.

Symbol table.

Static relocation table connects references to symbols.