PB173 Linux

06 Signály

Roman Lacko xlacko1@fi.muni.cz

2022-10-21

Obsah

Vlastnosti procesov: Pokračovanie

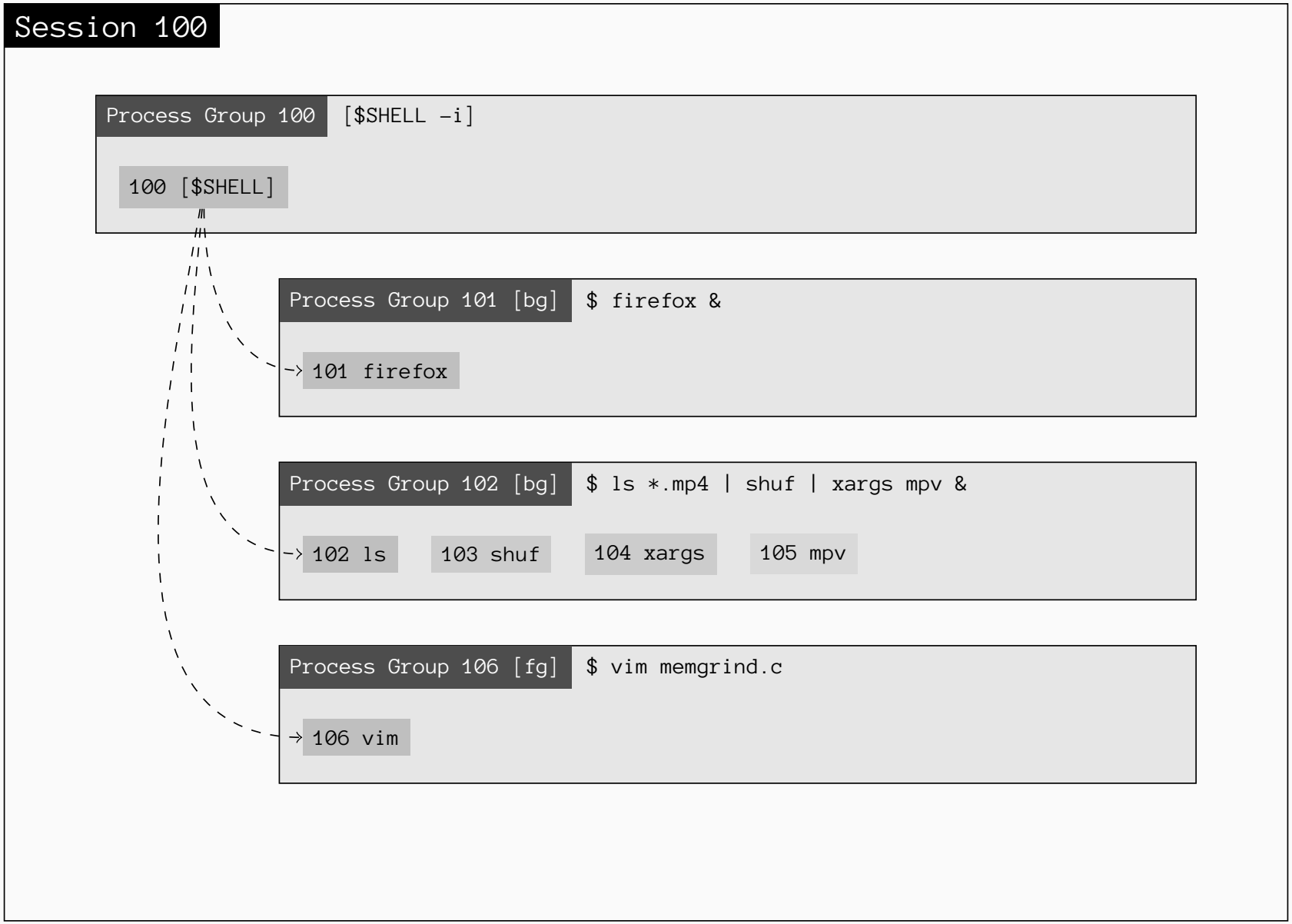
Poverenia

Poverenia (credentials)

Vlastnosti, ktoré jadro používa na kontrolu práv procesu.

- PID, PPID
- PGID, SID
- používateľké ID (RUID, EUID, Saved-UID)
- skupinové ID (RGID, EGID, Saved-GID, Supplementary GID)
- schopnosti (*capabilities*)

Sedenia


Session 100
Process Group 100
[$SHELL -i]
100 [$SHELL]
Process Group 101 [bg]
$ firefox &
101 firefox
Process Group 102 [bg]
$ ls *.mp4 | shuf | xargs mpv &
102 ls
103 shuf
104 xargs
105 mpv
Process Group 106 [fg]
$ vim memgrind.c
106 vim

Sedenia

Sedenie

(ang. *session*) je množina skupín procesov.

- SID Session ID
- Môže mať *kontrolný terminál*
- Číslo je rovnaké ako PID prvého procesu *Session Leader*

Odpojením procesu od sedenia s kontrolným terminálom vznikne **démon**.

Sedenia

```c
#include <unistd.h>
pid_t getsid(pid_t pid);
pid_t setsid(void);
```

Ak chce proces vytvoriť nové sedenie, nesmie byť lídrom skupiny (*Process Group Leader*)

Skupiny procesov

Skupina procesov

(ang. *process group*) je množina príbuzných procesov.

- PGID Process Group ID
- Číslo rovnaké ako PID prvého procesu *Process Group Leader*

Typy skupín:

- *Foreground* Process Group
- *Background* Process Group
- *Orphaned* Process Group

Skupiny procesov

```c
#include <unistd.h>
int setpgid(pid_t pid, pid_t pgid);
pid_t getpgid(pid_t pid);
```

V Linux viacero rozhraní:

```c
pid_t getpgrp(void);                    /* POSIX.1 */
pid_t getpgrp(pid_t pid);               /* BSD */

int setpgrp(void);                      /* System V */
int setpgrp(pid_t pid, pid_t pgid); /* BSD */
```

Démoni

Démoni

Daemon ≠ demon

- Proces bežiaci na pozadí
- Žiadna priama interakcie s používateľom
- Riadený IPC (signály, sokety, ...)
- Obvykle názov končí `d`
  `sshd`, `syslogd`, `systemd`, ...

```
#include <unistd.h>
int daemon(int nochdir, int noclose);
```

Schopnosti

Privilegovaný proces podľa POSIX:

- EUID = 0
- EGID = 0

Privilegovaný proces v Linuxe:

- Neprázdna podmnožina schopností (*capabilities*)
- Vyššia granularita práv v systéme

Schopnosti

`CAP_SET_UID` Ľubovoľné zmeny v `*UID`

`CAP_SET_GID` Ľubovoľné zmeny v `*GID`

`CAP_CHOWN` Ľubovoľné zmeny vo vlastníkoch súborov

`CAP_NET_ADMIN` Sieťové operácie

… `man 7 capabilities`

Schopnosti

File capabilities

- Proces s `CAP_SETFCAP` môže udeliť schopnosti spustiteľnému súboru
- Schopnosti sa pri spustení nastavia
  `setcap cap_α,cap_β,…+pe FILE`

Nástroje

- `setcap`, `getcap`, `netcap`, `pscap`, …
- C API vid' `man 7 capabilities`

Signály

Signály

Asynchronné notifikácie o udalosti

Seminár 10

Udalosťami riadené programovanie

Všeobecný princíp ošetrenia signálu

1. Doručenie signálu
2. Prerušenie činnosti procesu
3. Obsluha
4. Obnova činnosti alebo koniec

Jednoduché signály

Jednoduché signály (POSIX Reliable Signals)

- Tiež *Ordinary* alebo *Standard Signals*
- Kódy závislé na platforme
  `man 7 signal`

Nepoužívajte číselné hodnoty, ale makrá s predponou `SIG`.

Reliable?

- Nie je garantované poradie doručenia
- Viacnásobné signály sa môžu stratiť
- Len dva signály na vlastné účely (`USR1`, `USR2`)

Jednoduché signály

Jednoduché signály v C99:

`SIGABRT` Zlyhanie spôsobené `abort()`

`SIGFPE` Chyba na ALU

`SIGILL` Neplatná inštrukcia

`SIGINT` Prerušenie

`SIGSEGV` Chyba dereferencie ukazovateľa

`SIGTERM` Ukončenie procesu

Jednoduché signály

POSIX má navyše 22 signálov

`SIGKILL` Vražda procesu

`SIGSTOP`, `SIGCONT` Zastavenie resp. obnovenie behu procesu

`SIGALRM` Prebudenie z časovača

`SIGTRAP` *Breakpoint*

`SIGPIPE` Prelomené potrubie

Jednoduché signály: C99

Rozhranie pre signály v C99

```
#include <signal.h>
void (*signal(int sig, void (*handler)(int)))(int);

#define SIG_DFL /* … */
#define SIG_IGN /* … */

int raise(int signal);
typedef /* … */ sig_atomic_t;
```

`signal()` používajte len s `SIG_DFL` a `SIG_IGN`

Jednoduché signály: Blokovanie

Rozhranie pre signály v POSIX

```
#include <unistd.h>
int pause(void);

#define _POSIX_C_SOURCE
#include <signal.h>
int sigsuspend(const sigset_t *sigmask);
```

Jednoduché signály: Maskovanie

```c
#include <signal.h>
typedef /* … */ sigset_t;

int sigemptyset(sigset_t *set);
int sigfillset(sigset_t *set);
int sigaddset(sigset_t *set, int signum);
int sigdelset(sigset_t *set, int signum);
int sigismember(const sigset_t *set, int signum);

#define SIG_BLOCK /* … */
#define SIG_UNBLOCK /* … */
#define SIG_SETMASK /* … */
int sigprocmask(int how, const sigset_t *new_mask,
                         sigset_t *old_mask);
```

Jednoduché signály: Obsluha

```
struct sigaction {
    void     (*sa_handler)(int);
    void     (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t   sa_mask;
    int        sa_flags;
};

#define SA_SIGINFO /* … */   // → sa_sigaction
#define SA_RESETHAND /* … */ // → signal()
#define SA_RESTART /* … */

int sigaction(int signum, const struct sigaction *new_act,
                                struct sigaction *old_act);
```

Nepoužívajte `signal()`

Na nastavenie obsluhy signálu používajte vždy `sigaction()`!

Jednoduché signály: Obsluha

Ako písať obslužné funkcie

- Čo najjednoduchší a reentrantný kód
  - Nemeňte globálne premenné, ak nemusíte
  - Ak musíte, tak `volatile` a `sig_atomic_t`.
- Používajte ideálne len bezpečné funkcie
  `man 7 signal-safety`
- Nemeňte `errno`

Špeciality

- `fork()` obsluhu zdedí, `execve()` nie
- `longjmp()` môže obsluhu signálu ukončiť,
  ale môže viesť k nedefinovanému správaniu

Jednoduché signály: Odosielanie

```c
#include <signal.h>
int kill(pid_t pid, int sig);
```

- `raise(Σ)` ≈ `kill(getpid(), Σ)`
- `abort()` ≈ `kill(getpid(), SIGABRT)`

Špeciálne funkcie

```c
#include <abort.h>
void abort(void);  /* SIGABRT */

#include <unistd.h>
unsigned alarm(unsigned seconds); /* SIGALRM */
```

Spoľahlivé signály

Spoľahlivé signály (POSIX Real-Time Signals)

- Nie sú pomenované
- Nemajú špeciálny význam
- `SIGRTMIN` $\longrightarrow$ `SIGRTMAX`

! Nepoužívajte číselné hodnoty

Definujte si makrá:

```
#define SIGRTQEMPTY (SIGRTMIN + 0)
#define SIGRTQFULL  (SIGRTMIN + 1)
```

Spoľahlivé signály

- Signal Queue s kapacitou `SIGQUEUE_MAX`
- Poradie doručenia je deterministické
- Môžu niesť nejakú hodnotu od odosielateľa

```c
#define _POSIX_C_SOURCE 199309L /* 200809L */
#include <signal.h>
int sigqueue(pid_t pid, int sig, const union sigval value);
```

Synchrónna obsluha signálov

```
int sigwait(const sigset_t *set, int *sig);
int sigwaitinfo(const sigset_t *set, siginfo_t *info);
int sigtimedwait(const sigset_t *set, siginfo_t *info,
        const struct timespec *timeout);

int sigpending(sigset_t *set);
```

Ďalšie funkcie

```c
#define _POSIX_C_SOURCE 200809L
#include <signal.h>
char *strsignal(int sig);

#define _DEFAULT_SOURCE
#include <signal.h>

#define NSIG /* … */

void psignal(int sig, const char *s);
void psiginfo(const siginfo_t *pinfo, const char *s);

const char *sigdescr_np(int sig);
const char sigabbrev_np(int sig);
```

_np znamená *non-portable*.

Pokročilé operácie s procesmi

Process Control

```c
#include <sys/prctl.h>

#define PR_SET_CHILD_SUBREAPER /* … */ // prctl(a, bool)
#define PR_SET_PDEATHSIG /* … */       // prctl(a, int)
#define PR_SET_NAME /* … */            // prctl(a, char *)
#define PS_SET_SECCOMP /* … */

int prctl(int option, unsigned long arg2, … arg5);
```

Trasovanie

Trasovanie

Sledovanie správania iného procesu

tracer Proces, ktorý sleduje iný

tracee Sledovaný proces

Stavy z pohľadu trasovania:

- `ptrace-stop`: Čaká na príkazy `tracer` procesu
- `running`: Normálna činnosť procesu

Trasovanie

```c
#include <sys/ptrace.h>

#define PTRACE_SET_OPTIONS /* … */
#define PTRACE_TRACEME /* … */ /* Trace child */
#define PTRACE_ATTACH /* … */  /* Attach other and stop */
#define PTRACE_SEIZE /* … */   /* Attach other, do not stop */
#define PTRACE_DETACH /* … */

#define PTRACE_INTERRUPT /* … */
#define PTRACE_CONT /* … */
#define PTRACE_SYSCALL /* … */
#define PTRACE_SINGLESTEP /* … */
#define PTRACE_{PEEK,POKE}{DATA,TEXT,USER} /* … */
#define PTRACE_{GET,SET}REGS /* … */

int ptrace(enum __ptrace_request op, pid_t pid,
        void *addr, void *data);
```

Sandbox

- Oddelenie procesov medzi sebou
- Obmedzenie dostupných operácií
- Skúmanie nedôveryhodných programov

Techniky

- Menné priestory (`namespaces`)
  a kontrolné skupiny (`cgroups`)
- Zmena koreňového adresára
- Audit systémových volaní (`seccomp`)
- Bezpečnostné moduly jadra (SELinux, ...)
- Kontajnerizácia, virtualizácia

Sandbox: Menné priestory

Abstrakcia zdrojov systému:

`user ns`	Schopnosti a oprávnenia
`pid ns`	Strom procesov
`mount ns`	Súborové systémy
`net ns`	Sieťové rozhrania

...

```
man 7 namespaces
```

Sandbox: Menné priestory

Manipulácia s mennými priestormi vyžaduje `CAP_SYS_ADMIN`, okrem *user namespace*.

User namespace

Prepnutie do User NS dá procesu všetky schopnosti na zdroje **v tomto priestore**, vrátane `CAP_SYS_ADMIN`.

```
#include _GNU_SOURCE
#include <sched.h>

#define CLONE_NEWUSER /* … */
#define CLONE_NEWPID /* … */
…

int unshare(int flags);
```

Existuje tiež `man 1 unshare`

Sandbox: Menné priestory

Okrem prepnutia je možné proces v priestore vyrobit.

Clone

Systémové rozhranie `clone()` v Linuxe implementuje `fork()`, `pthread_create()` a podobné volania.

```
#include <linux_sched.h>
#include <sched.h>
#include <sys/syscall.h>

struct clone_args {
    u64 flags;
    u64 exit_signal;
    …
};

long syscall(SYS_clone3, struct clone_args *args, size_t size);
```

Sandbox: Chroot

```c
#include <unistd.h>
int chroot(const char *path);
```

Sandbox: Seccomp

```c
#include <linux/seccomp.h>
#define SECCOMP_SET_MODE_STRICT /* … */
#define SECCOMP_SET_MODE_FILTER /* … */
int seccomp(unsigned int operation, unsigned int flags,
        void *args);

#include <seccomp.h>
scmp_filter_ctx seccomp_init(unit32_t def_action);
int seccomp_rule_add*(scmp_filter_ctx ctx, uint32_t action,
        int syscall, ...);
int seccomp_load(scmp_filter_ctx ctx);
```