PB173 Linux

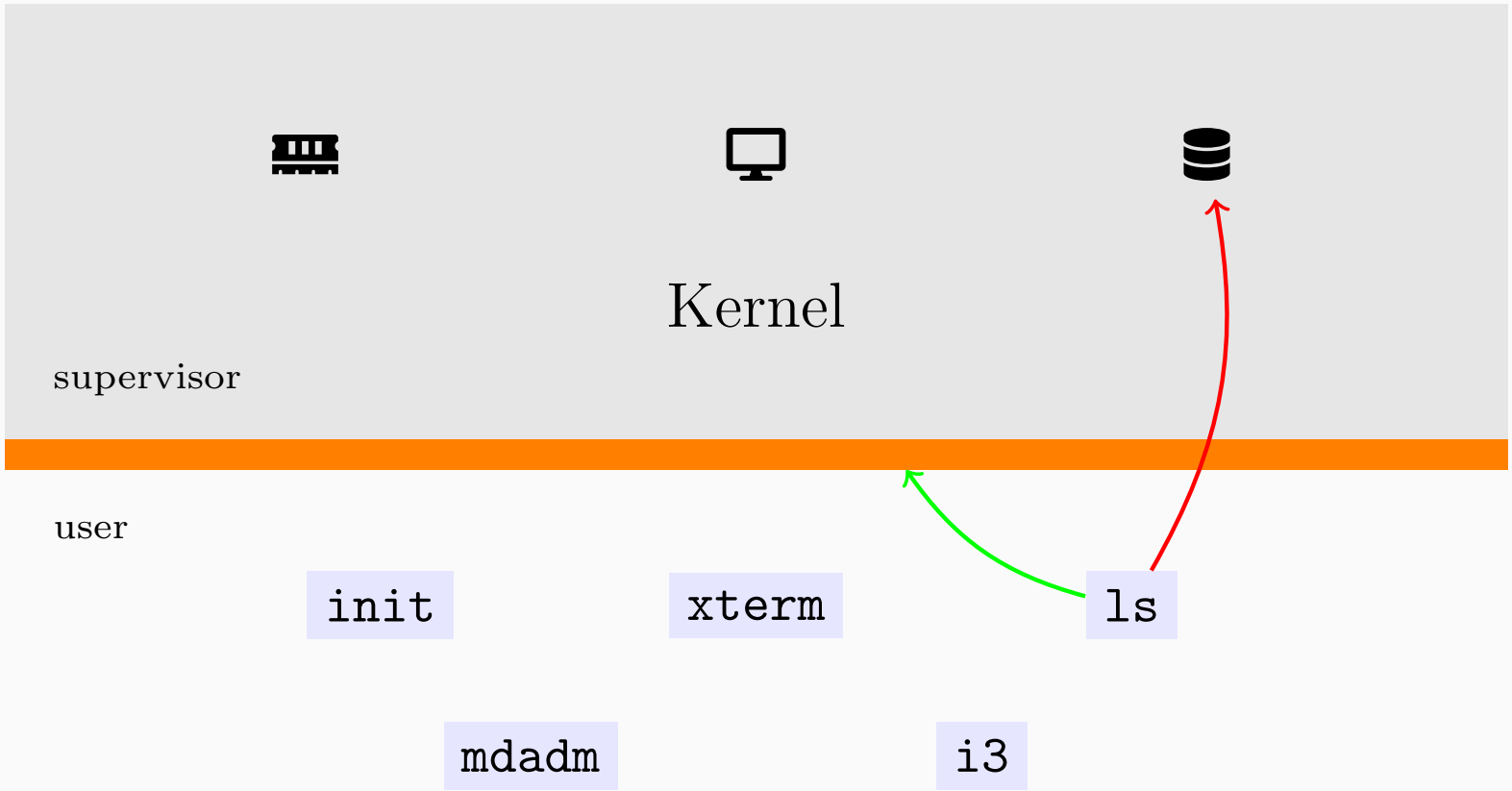
05 Procesy

Roman Lacko xlacko1@fi.muni.cz

2022-10-14

Systémové volania

Systémové volania



Systémové volania

- Služba jadra
- Nutné prepnutie do *privilegovaného režimu*
  - Prerušenie, *call gate*, špeciálne inštrukcie, ...
- `man 2 intro`
- `man 2 syscalls`

Systémové volania: Manuál

Rozdiely v manuálových stránkach:

`man 2 open` Dokumentácia systémového volania[1].

`man 3 open` Dokumentácia funkcie v knižnici (libc).

`man 3p open` Dokumentácia funkcie podľa POSIXu.

[1] Presnejšie, obaľujúcej funkcie v štandardnej knižnici

Systémové volania: Chyby

Návratové kódy systémových volaní:

- ≥ 0: Úspech, prípadne nejaká hodnota
- < 0: Chyba vrátane kódu

Chybový kód

Obaľujúce funkcie chybový návratový kód zmenia na `-1`
a kód chyby uložia do `errno`.

Systémové volania: Hlásenie chýb

Chyby systémových volaní sú *očakávateľné chyby*.

Viď `patterns/01-defensive-offensive-programming`.

```
#include <errno.h>
extern int errno;
```

⚠ errno

Pri vláknach si ukážeme, že `errno` je v skutočnosti trochu zložitejší objekt.

```
#include <stdlib.h>
char *strerror(int number);
```

errno

- (Niektoré) funkcie nastavujú `errno` pri chybe.
- **Žiadne** funkcie `errno` pri úspechu nenastavujú.

Kontrolujte najprv návratový kód funkcie!

- Ak funkcia zlyhala, (väčšinou) nastaví `errno`.
- Ak je `errno` nastavené, funkcia ešte *nemusela* zlyhať.

Nulovanie `errno`

Chybu niektorých funkcií (napr. `strtol()`) je občas možné zistiť len z `errno`. Pred nimi je preto vhodnejšie `errno` vynulovať.

Systémové volania: Hlásenie chýb

```c
#include <stdio.h>
void perror(const char *s);
```

perror(s) je zhruba ekvivalentné
`fprintf(stderr, "%s: %s\n", s, strerror(errno));`

```c
#include <err.h>
void err(int eval, const char *fmt, ...);
void errx(int eval, const char *fmt, ...);
void warn(const char *fmt, ...);
void warnx(const char *fmt, ...);
```

Systémové volania: Hlásenie chýb (GNU)

```c
#define _GNU_SOURCE
#include <errno.h>
extern char *program_invocation_name;
extern char *program_invocation_name_short;

#define _GNU_SOURCE
#include <error.h>
int error(int status, int errnum, const char *format, ...);
```

Procesy

Procesy

Program

- abstraktný alebo konkrétny popis nejakej úlohy
- obvykle uložený v súbore

Proces

- inštancia programu
- vlastné prostriedky (pamäť, registre, ...)
- usporiadané v stromovej štruktúre

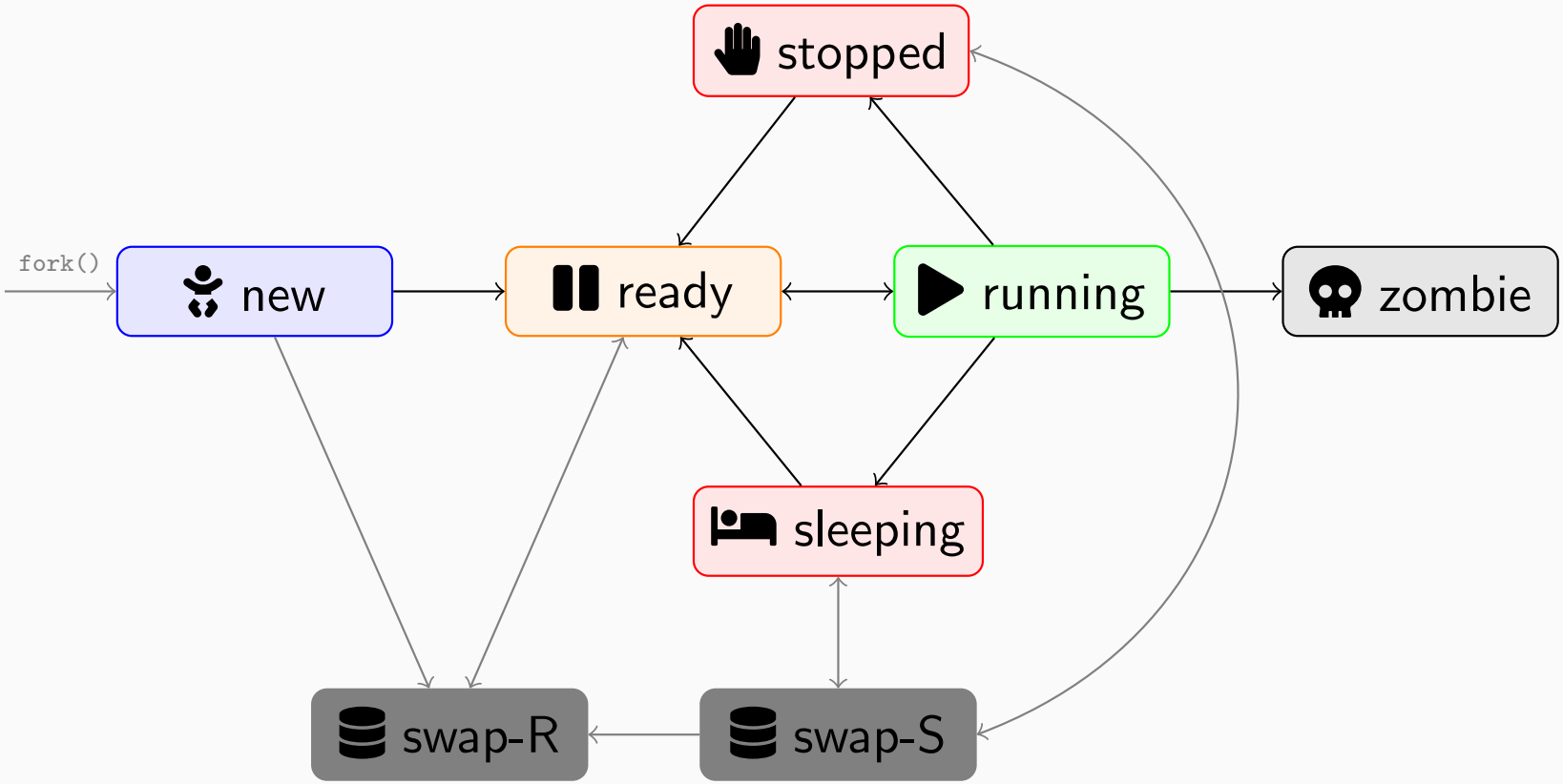
Procesy: PID

Process Identifier

- jednoznačná identifikácia procesu v systéme
- `procfs` (`/proc/$pid`), `ps`
- koreň má PID 1

```
#include <unistd.h>
pid_t getpid(void);
pid_t getppid(void);
```

Procesy: Životný cyklus



Procesy: Subproces

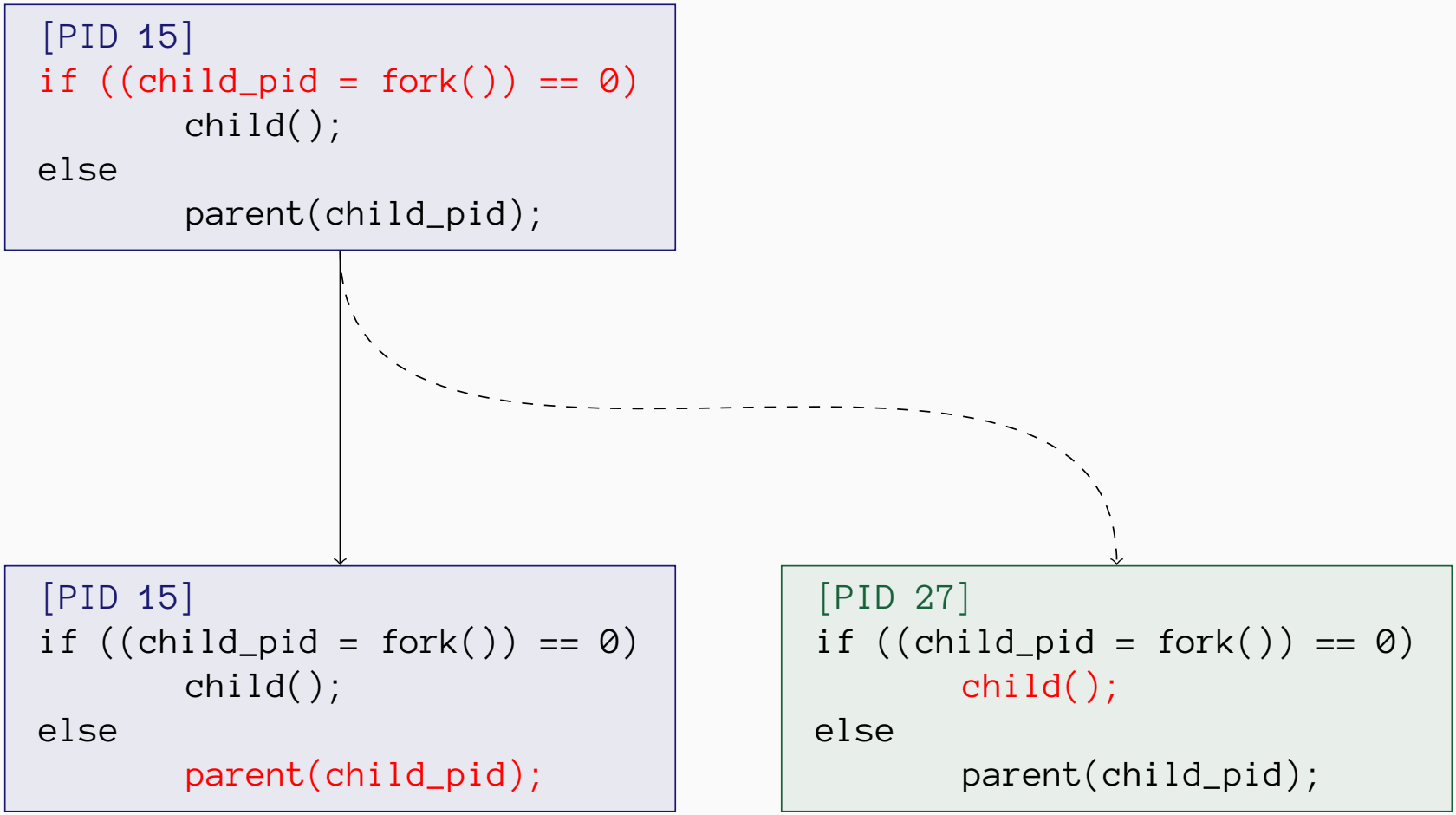
```c
#include <stdlib.h>
int system(const char *command);
```

- Spustenie programu ako `sh -c "COMMAND"`
- Problémy s prenositeľnosťou
- Problémy s bezpečnosťou

Ak nemusíte, radšej nepoužívajte.

Procesy: Vznik

```
#include <unistd.h>
pid_t fork(void);
```



Procesy: Nahradenie

```c
#include <unistd.h>
int execv (const char *path, char *const argv[]);
int execve (const char *path, char *const argv[],
        char *const env[]);
int execvp (const char *file, char *const argv[]);
int execvpe(const char *file, char *const argv[],
        char *const env[]);
int execl (const char *path, const char *arg, ..., /* NULL */);
int execlp (const char *file, const char *arg, ..., /* NULL */);
int execle (const char *path, const char *arg, ..., /* NULL */,
        char *const env[]);
```

Z týchto volaní je len execve() skutočné systémové volanie.

Procesy: Nahradenie

`exec[vl](pe)()`:

- **`v`** Parametry sú v poli `argv[]`, posledný prvok `NULL`.
- **`l`** Parametry sú v elipse, posledný prvok `NULL`.
- **`p`** Namiesto celej cesty berie názov programu a hľadá ho v `PATH`.
- **`e`** Prostredie ako samostatný parameter.

Procesy: Nahradenie (Linux)

```c
#define <unistd.h>
int execveat(int dirfd, const char *path, char *const argv[],
        char *const env[], int flags);

#define AT_FDCWD /* implementation-specific (probably -100) */

int fexecve(int fd, char *const argv[], char *const envp[]);
```

Procesy: Čakanie

```c
#include <sys/wait.h>
pid_t wait(int *wstatus);
pid_t waitpid(pid_t pid, int *wstatus, int options);
pid_t waitid(idtype_t idtype, id_t id, siginfo_t *sig,
        int options);

#define WNOHANG /* ... */

#define WIFEXITED(STATUS) /* ... */
#define WEXITSTATUS(STATUS) /* ... */

#define WIFSIGNALED(STATUS) /* ... */
#define WTERMSIG(STATUS) /* ... */
```

Procesy: Čakanie (BSD)

```c
#define _DEFAULT_SOURCE
#include <sys/wait.h>
pid_t wait3(int *wstatus, int options, struct rusage *rusage);
pid_t wait4(pid_t pid, /* wstatus, options, rusage */);
```

Procesy: Ukončenie

```c
#include <stdlib.h>
void exit(int status);
```

```c
#include <stdlib.h>
int atexit(void (*f)(void));
```

```c
#define _DEFAULT_SOURCE
#include <stdlib.h>
int on_exit(void (*f)(int, void *), void *arg);
```

Procesy: Ukončenie

Pozor, nepoužívajte `_exit()` ani `_Exit()` (ak nemusíte)!

```c
#include <unistd.h>
void _exit(int status);

#include <stdlib.h>
void _Exit(int status);
```

Vlastnosti

Vlastnosti: 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*)

Rôzne systémy môžu mať iné ďalšie poverenia.

```
man 7 credentials
```

Vlastnosti: Identifikácie

EUID, EGID (Effective ...)

- Zdedené pri `fork()`, môže sa zmeniť pri `execve()`:
  - EUID na UID programu, ak má nastavený Set-UID bit
  - EGID na GID programu, ak má nastavený Set-GID bit
  - ďalšie obmedzenia viď `man 2 execve`
- Používajú sa na vyhodnocovanie práv
- Bežný proces môže meniť na Real alebo Saved

Vlastnosti: Identifikácie

RUID, RGID (Real User ID, Real Group ID)

- Zdedené pri `fork()`
- ID, ktoré identifikuje vlastníka procesu

Saved-UID, Saved-GID

- Kópia EUID resp. EGID pri `execve()`

Supplementary GID

- Zoznam ďalších skupín procesu
- Prístup k súborom a zdrojom

Vlastnosti: Proces vs. používateľ

⚠ Pozor na rozdiel

Práva procesu a práva používateľov sú všeobecne nezávislé!

Práva používateľa sa berú do úvahy (väčšinou) len pri

- prihlásení do systému,
- spustení `su USER` a `sudo -u USER`.

Vlastnosti: Proces vs. program

Program na disku (ako súbor) má UID a GID.
Na práva spusteného procesu majú efekt:

- pri kontrole prístupu k súboru (napr. pri `execve()`),
- ak majú Set-UID alebo Set-GID bit.

! Neexistujúce ID

UID a GID programu na disku nemusí odkazovať
na skutočného používateľa či skupinu v systéme. Prečo?

Proces: User ID

```c
#include <unistd.h>
uid_t getuid(void);
int setuid(uid_t uid);

uid_t geteuid(void);
int seteuid(uid_t euid);

int setreuid(uid_t uid, uid_t euid);
```

Rozšírenie (GNU)

```c
#define _GNU_SOURCE
#include <unistd.h>
int getresuid(uid_t *uid, uid_t *euid, uid_t *suid);
int setresuid(uid_t uid, uid_t euid, uid_t suid);
```

Proces: Group ID

```c
#include <unistd.h>
gid_t getgid(void);
int setgid(gid_t gid);

gid_t getegid(void);
int setegid(gid_t egid);

int setregid(gid_t gid, gid_t egid);
```

Rozšírenie (GNU)

```c
#define _GNU_SOURCE
#include <unistd.h>
int getresgid(gid_t *gid, gid_t *egid, gid_t *sgid);
int setresgid(gid_t gid, gid_t egid, gid_t sgid);
```

Proces: Zmeny v ID

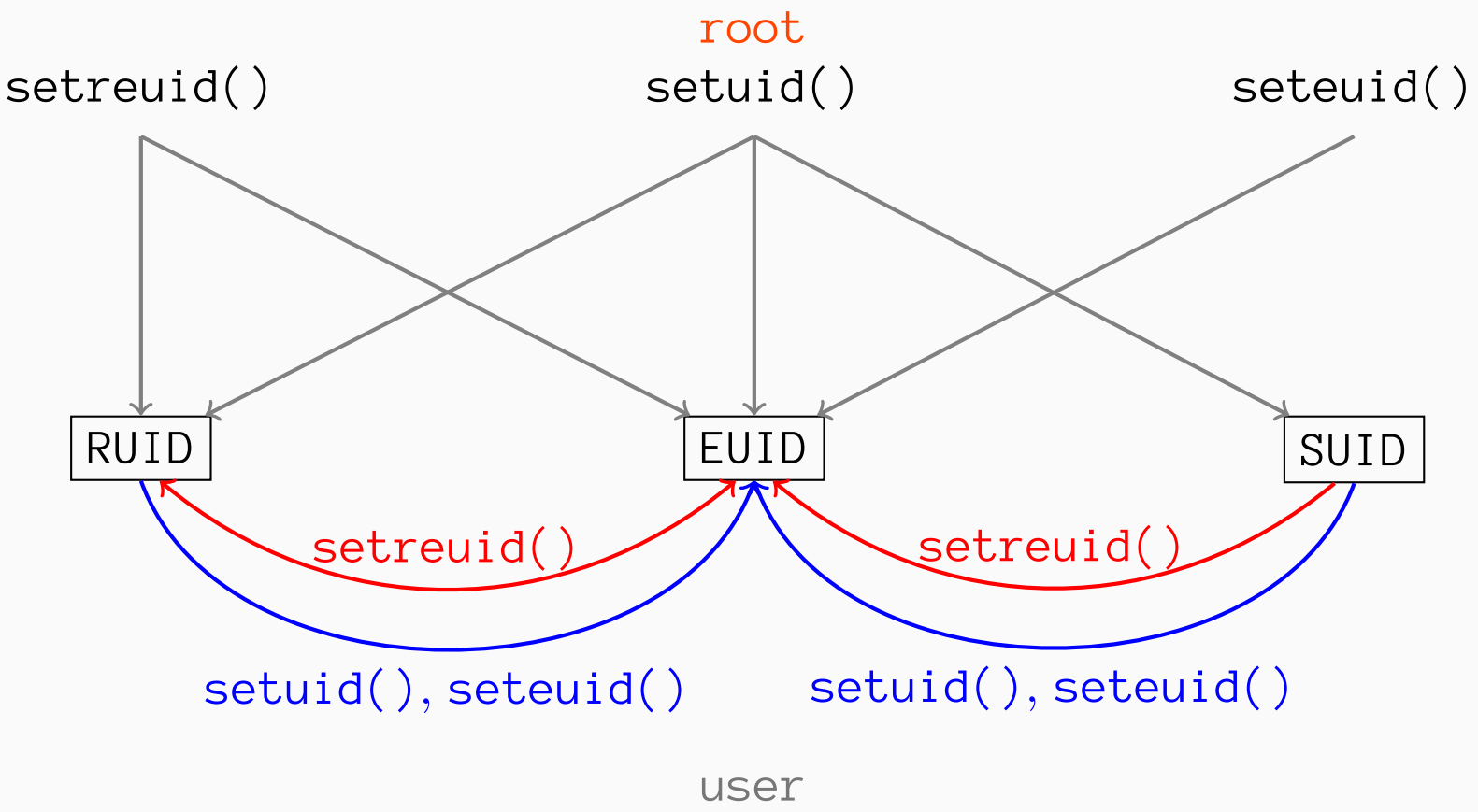
Proces, ktorý má EUID rovné 0 alebo `CAP_SETUID` schopnosť, môže robiť ľubovoľné zmeny vo svojich User ID[2].

Ostatné procesy môžu nastaviť svoje EUID
na RUID alebo Saved-UID.

- Proces sa môže „ponížiť” nastavením všetkých svojich ID na nenulové číslo a zrušením schopností.
- Bežný proces sa nikdy nemôže „povýšiť“.

[2] Pre Group ID analogicky

Proces: Zmeny v ID



Analogicky pre Group ID.

Proces: Doplnkové skupiny

```c
#include <unistd.h>
int getgroups(int size, gid_t list[]);

#include <grp.h>
int setgroups(size_t size, const git_t list[]);
```

Zmeny v pomocných skupinách môže robiť len proces s EGID rovné 0 alebo so schopnosťou `CAP_SETGID`.

Používatelia a skupiny

Používatelia v systéme obvykle majú

- Login
- UID a GID (primárna skupina)
- Môžu mať doplnkové skupiny
- Domovský adresár

Užitočné programy:

- `id`
- `getent passwd`, `getent group`
- `newgrp`

Používatelia a skupiny

Databáza používateľov a skupín

`/etc/passwd` Zoznam používateľov

`/etc/shadow` Heslá používateľov (hashe)

`/etc/group` Zoznam skupín

`/etc/gshadow` Heslá skupín (hashe)

Vzdialená databáza

Záznamy o používateľoch môžu pochádzať aj z iného systému napr. LDAP, SSSD, NSS, ...

Používatelia a skupiny

```c
#include <pwd.h>

struct passwd {
    char *pw_name;
    char  pw_uid, pw_gid;
    char *pw_dir;
    ...
};

struct passwd *getpwnam(const char *name);
struct passwd *getpwuid(uid_t uid);
/* getpwnam_r(), getpwuid_r() */

struct passwd *getpwent(void);
void setpwent(void);
void endpwent(void);
```

Používatelia a skupiny

```c
#include <grp.h>

struct group {
    char  *gr_name;
    char   gr_gid;
    char **gr_mem;
    ...
};

struct group *getgrnam(const char *name);
struct group *getgrgid(gid_t gid);
/* getgrnam_r(), getgrgid_r() */

struct passwd *getgrent(void);
void setgrent(void);
void endgrent(void);
```

Používatelia a skupiny

Suplementárne skupiny (BSD)

```
#define _DEFAULT_SOURCE
#include <grp.h>
int getgrouplist(const char *user, gid_t group, gid_t *groups,
        int *ngroups);
```

Záver

Záver

Procesy

- The Linux Process Model

Poverenia

- Set-UID Demystified
- Revising “Set-UID Demystified”