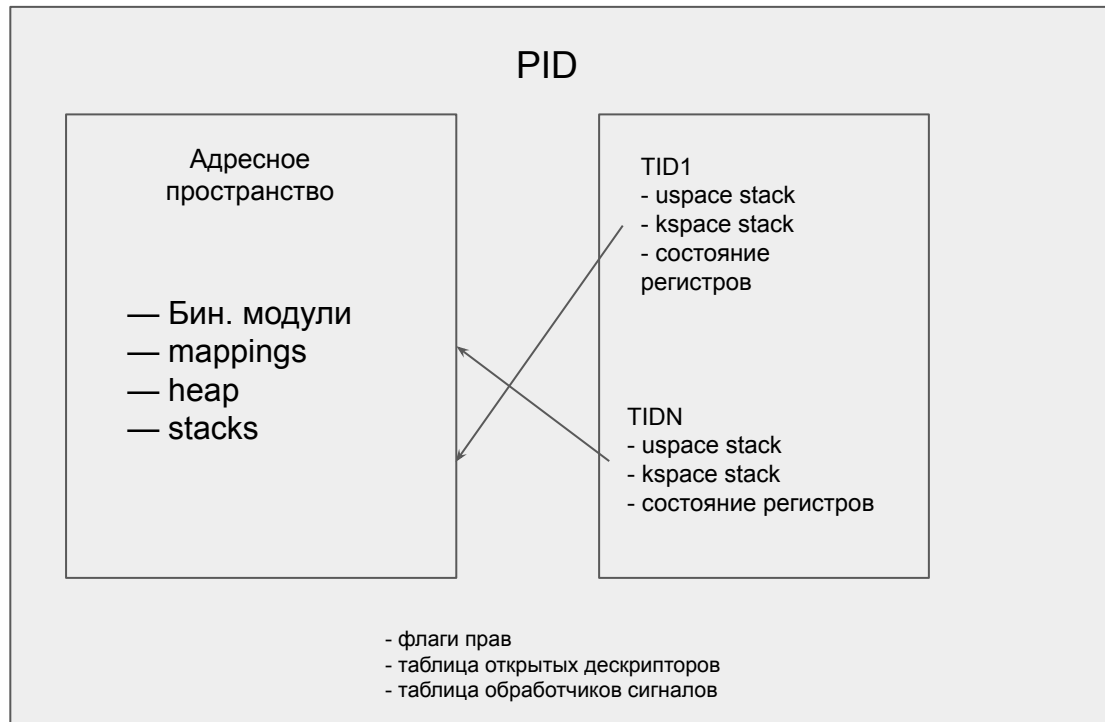


Программирование в Linux

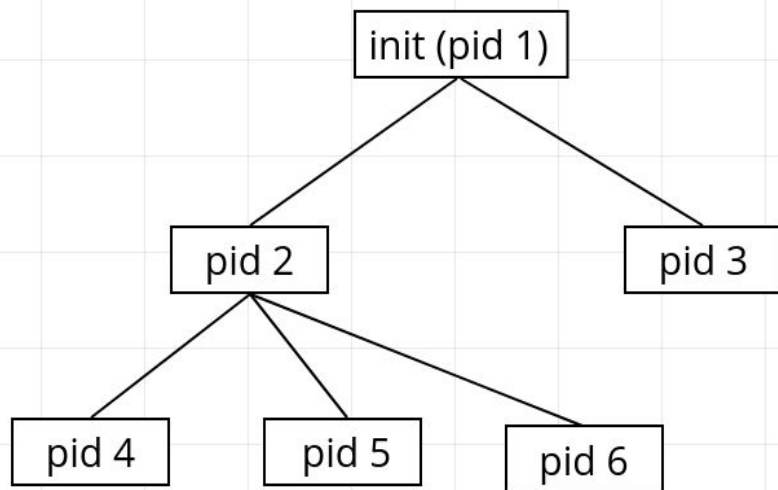


Процессы и права

Процесс ~ контейнер. Thread — единица исполнения



Порождение процессов



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

```
int clone(int (*fn)(void *), void *stack, int flags, void *arg, ...  
        /* pid_t *parent_tid, void *tls, pid_t *child_tid */ );
```

```
#include <sys/types.h>  
#include <unistd.h>
```

```
pid_t fork(void);
```

```
#include <sys/types.h>  
#include <unistd.h>
```

```
pid_t vfork(void);
```

fork vs vfork

```
6 int main(int argc, char* argv) {
7
8     int x = 42;
9
10    pid_t child = fork();
11
12    if (child < 0) {
13        return EXIT_FAILURE;
14    }
15
16    // parent and child run simultaneously
17    if (child == 0) {
18        printf("Hello from child, %d\n", x);
19    } else {
20        x = 43; // ok!
21        printf("Hello from parent of %d, answer = %d\n", child, x);
22    }
23 }
```

```
dmis@dmis-MS-7A15:~/LinuxEgs/fokrs$ ./fork_eg
Hello from parent of 160767, answer = 43
Hello from child, 42
```

```
5
6 int main(int argc, char* argv) {
7
8     int x = 42;
9
10    pid_t child = vfork();
11
12    if (child < 0) {
13        return EXIT_FAILURE;
14    }
15
16    // parent is blocked until child runs exec or exit
17    if (child == 0) {
18        // x = 43; // UB!
19        printf("Hello from child, %d\n", x); // UB!!!!
20    }
21    execlp("cat", "dog", "--help", NULL);
22 } else {
23     printf("Hello from parent of %d, answer = %d\n", child, x);
24 }
25 }
```

```
dmis@dmis-MS-7A15:~/LinuxEgs/fokrs$ ./vfork_eg
Hello from parent of 161553, answer = 42
dmis@dmis-MS-7A15:~/LinuxEgs/fokrs$ Usage: dog [OPTION]... [FILE]...
Concatenate FILE(s) to standard output.
```

fork делает CoW копию адресного пространства. vfork — нет

Замена бинарного модуля (exec)

```
1 #include <unistd.h>
2
3 int main() {
4     execlp("cat", "dog", "--help", NULL);
5     // Unreachable
6     return 1;
7 }
```

```
#include <unistd.h>

extern char **environ;

int execl(const char *pathname, const char *arg, ...
          /* (char *) NULL */);
int execlp(const char *file, const char *arg, ...
          /* (char *) NULL */);
int execl_e(const char *pathname, const char *arg, ...
            /*, (char *) NULL, char *const envp[] */);
int execl_v(const char *pathname, char *const argv[]);
int execl_vp(const char *file, char *const argv[]);
int execl_vpe(const char *file, char *const argv[],
              char *const envp[]);
```

PID сохраняется

Адресное пространство новое

Открытые дескрипторы сохраняются

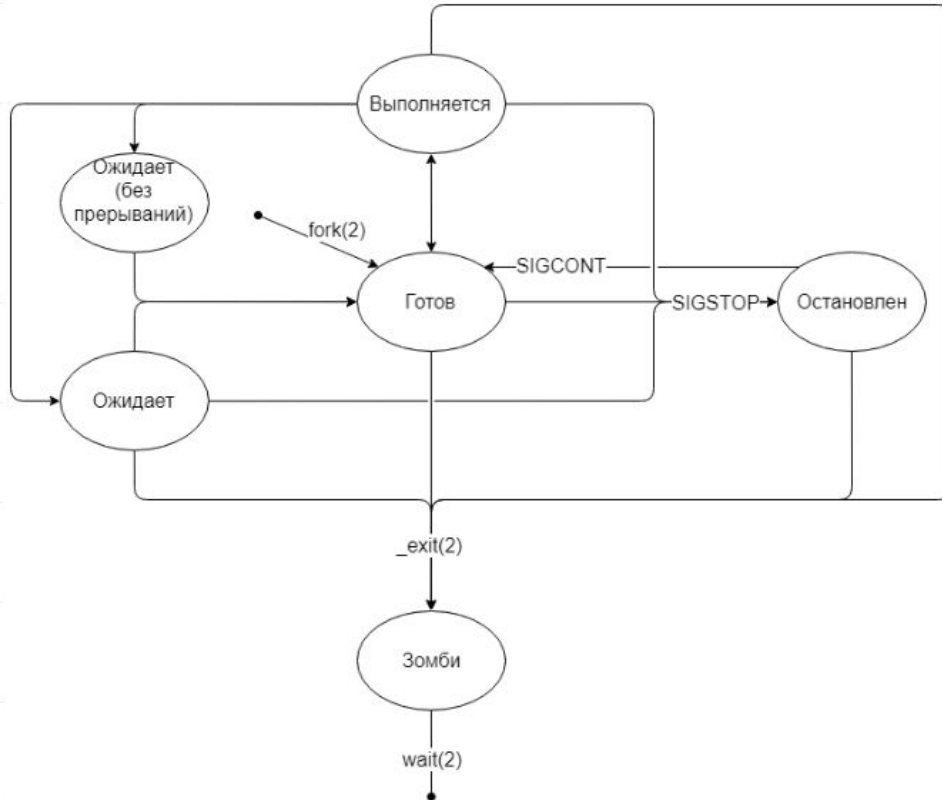
*p-версии умеют запускать скрипты (#!)

std::system

```
...  
int exit_code1 = std::system("echo hello1");  
printf("exit_code1 %d\n", exit_code1);
```

```
... auto child = vfork();  
... if (child < 0) {  
...     perror("vfork");  
...     return EXIT_FAILURE;  
... }  
... if (child == 0) {  
...     execl("/bin/echo", "/bin/echo", "hello2", NULL);  
... } else {  
...     int exit_code2 = 0;  
...     waitpid(child, &exit_code2, WEXITED);  
...     printf("exit_code2 %d\n", WEXITSTATUS(exit_code2));  
... }
```

Жизненный цикл процесса



Зомби — метаинформация о статусе
завершенного процесса

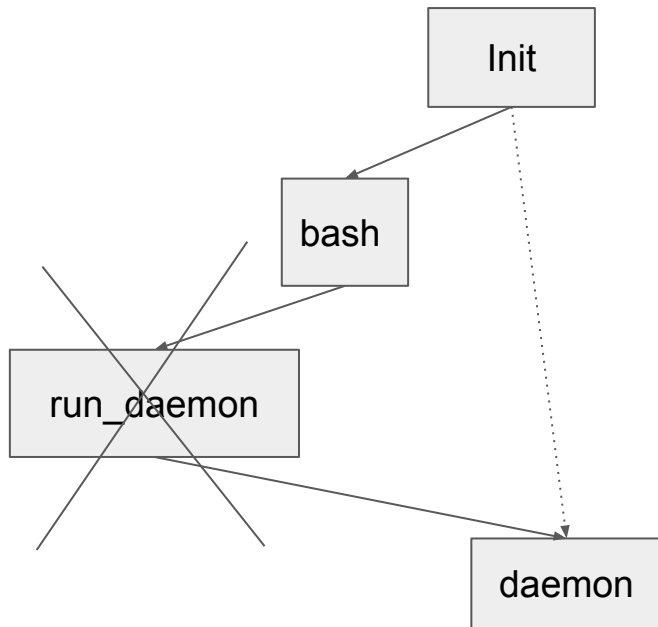
- Занимает место
- Освобождается после обслуживания (`wait`) от родителя
- Если родитель мертв, его заменяет `init`
- `Init` обслуживает всех своих (приемных) детей

“Демонение” процесса

```
#include <unistd.h>

int daemon(int nochdir, int noclose);
```

1. Делаем fork
2. Убиваем родителя
3. Потомок становится потомком init
4. Потомок закрывает все файловые дескрипторы
5. Потомок больше не привязан к терминалу и работает в фоне



Механизмы прав доступа

Пользователи и группы. Доступ и владение файлами

```
shum@sol:~$ ls -l
total 20
drwx----- 2 shum      staff    4096 Jan 16 22:04 Mail
drwx----- 3 shum      staff    4096 Jan 16 14:15 csc128
drwxr-xr-x  2 shum      staff    4096 Jan 13 16:42 public
drwxr-xr-x  2 shum      staff    4096 Jan 16 14:07 public_html
-rw-r--r--  1 shum      staff    628 Jan 15 20:04 verse
```

Diagram illustrating the components of the `ls -l` output:

- file type**: Indicated by the first character of the permissions (e.g., `d` for directory, `-` for regular file).
- permissions**: Indicated by the next nine characters (e.g., `rw-r--r--`).
- number of hard links**: Indicated by the number following the permissions (e.g., `2`).
- user (owner) name**: Indicated by the user name (e.g., `shum`).
- group name**: Indicated by the group name (e.g., `staff`).
- size**: Indicated by the file size in bytes (e.g., `4096`).
- date/time last modified**: Indicated by the date and time (e.g., `Jan 16 22:04`).
- filename**: Indicated by the file name (e.g., `Mail`).

Legend for permissions:

- rwx**: Readable, Writeable, Executable
- executable**: Indicated by the `x` character.
- writeable**: Indicated by the `w` character.
- readable**: Indicated by the `r` character.

Legend for permissions:

- other (everyone) permissions**: Indicated by the last three characters (e.g., `r--`).
- group permissions**: Indicated by the middle three characters (e.g., `-r--`).
- user permissions**: Indicated by the first three characters (e.g., `rw-`).

```
#include <sys/stat.h>
```

```
int chmod(const char *pathname, mode_t mode);
int fchmod(int fd, mode_t mode);
```

```
int chown(const char *pathname, uid_t owner, gid_t group);
int fchown(int fd, uid_t owner, gid_t group);
int lchown(const char *pathname, uid_t owner, gid_t group);
```

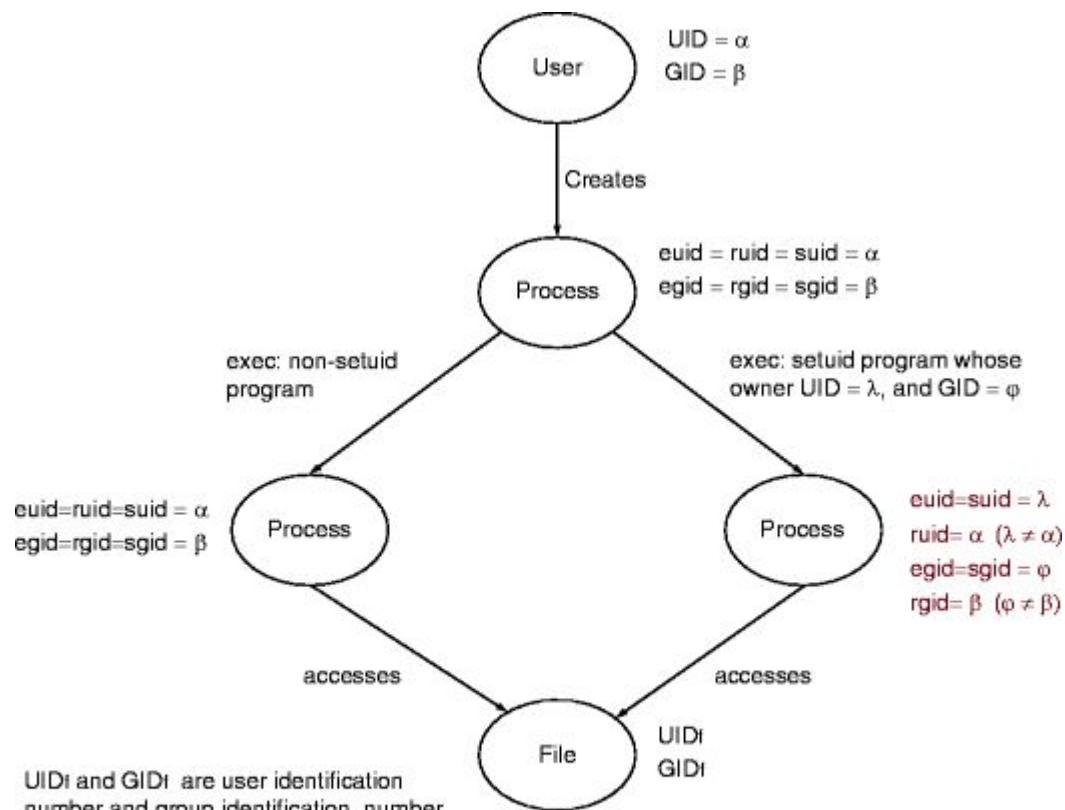
Пользователи и группы. Процессы

Исполнимый файл: UID, GID, SUID-bit. SGID-bit

Запущенный процесс:

- Real user ID (RUID) — кто запустил процесс?
- Effective user ID (EUID) — от чьего имени процесс выполняет запросы?
- Saved user ID (SUID) — кем можно временно прикинуться?

Аналогично для групп



UID_i and GID_i are user identification number and group identification number for file f .

Let $P_i(x)$ denote access permissions for file f , where $x \in \{\text{owner, group, others}\}$.

Accessible set F of files for process:

$$\begin{aligned}
 F = \{ & f \mid [(euid = UID_f) \wedge (P_i(\text{owner}))] \vee \\
 & [(euid \neq UID_f) \wedge (egid = GID_f) \wedge (P_i(\text{group}))] \vee \\
 & [(euid \neq UID_f) \wedge (egid \neq GID_f) \wedge (P_i(\text{others}))] \}
 \end{aligned}$$

