Process Control

System Programming

2019 여름 계절학기

한양대학교 공과대학 컴퓨터소프트웨어학부
홍석준

Changing User IDs and Group IDs

```
#include <unistd.h>
int setuid(uid_t uid);
int setgid(gid_t gid);
```

- ❑ **A superuser process can set the real UID, effective UID, and saved set-user-ID to *uid*.**
- ❑ **If *uid* equals either the real UID or the saved set-user-ID, `setuid` sets only the effective UID to *uid*.**
- ❑ **Otherwise, `errno` is set to EPERM.**

Changing User IDs and Group IDs

ID	exec		setuid(*uid*)	
	set-user-ID bit off	set-user-ID bit on	superuser	unprivileged user
real user ID	unchanged	unchanged	set to *uid*	unchanged
effective user ID	unchanged	set from user ID of program file	set to *uid*	set to *uid*
saved set-user ID	copied from effective user ID	copied from effective user ID	set to *uid*	unchanged

- **Only a superuser process can change the real user ID.**
- **The effective UID is set by the `exec` function, only if the set-user-ID bit is set for the program file. We can call `setuid` at any time to set the effective UID to either the real UID or the saved set-user-ID.**
- **The saved set-user-ID is copied from the effective UID by `exec`.**

Changing User IDs and Group IDs

- **saved set-user-ID feature**

- **Assuming that the `man` program file is owned by the user name `man` and has its <u>set-user-ID bit</u> set**

1. When we `exec` it,
   - real user ID = our user ID
   - effective user ID = `man`
   - saved set-user-ID = `man`

2. The `man` program accesses the required configuration files and manual pages (owned by the user name `man`.)

3. Before `man` runs any command on our behalf, it calls `seteuid(getuid())` to safely execute filter programs.
   - real user ID = our user ID (unchanged)
   - <u>effective user ID = our user ID</u>
   - saved set-user-ID = `man` (unchanged)

Changing User IDs and Group IDs

4. When the filter is done, `man` calls `seteuid(maneuid)`. This call is allowed because `maneuid` equals the saved set-user-ID.

   - real user ID = our user ID (unchanged)
   - effective user ID = `man`
   - saved set-user-ID = `man` (unchanged)

5. The `man` program can now operate on its files, as its effective UID is `man`.

- **Extra privileges at the beginning and end, but our normal privilege most of the time.**

Changing User IDs and Group IDs

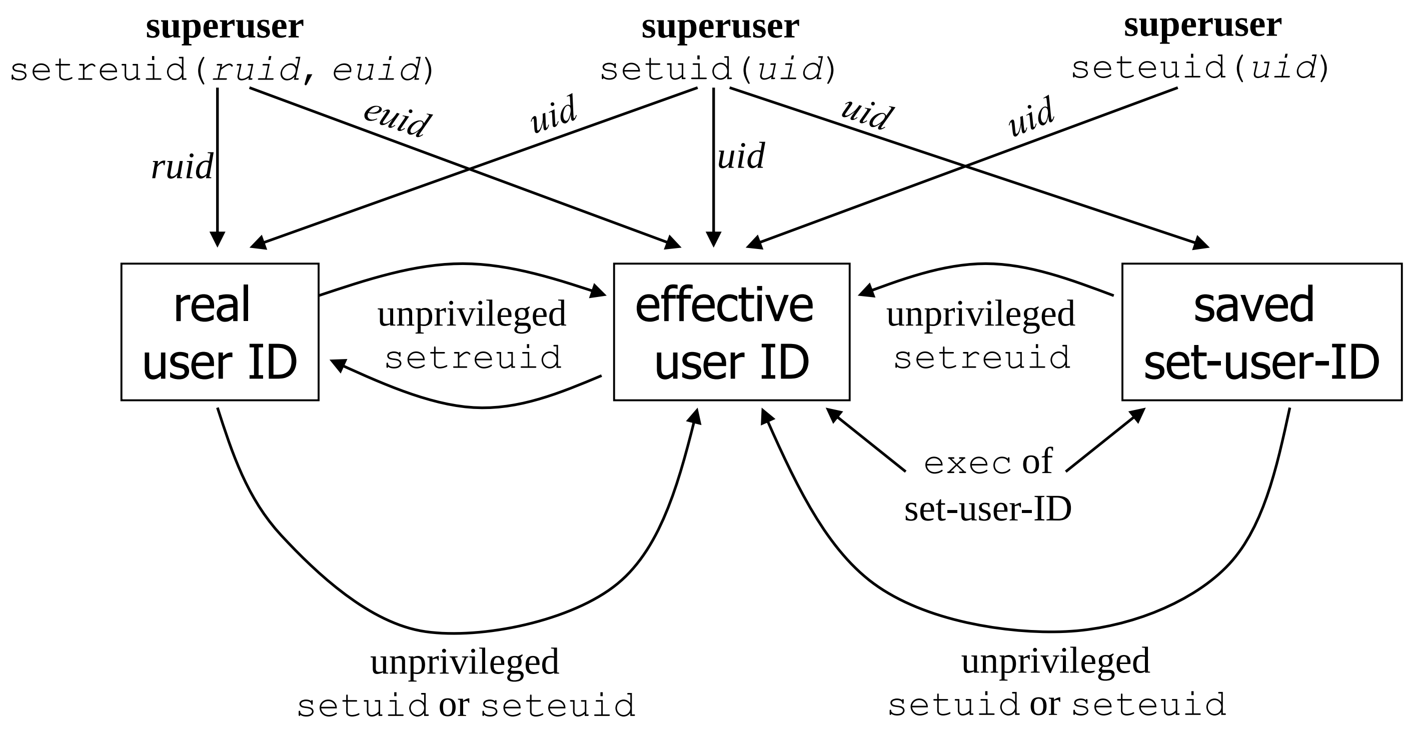
```
#include <unistd.h>
int setreuid(uid_t ruid, uid_t euid);
int setregid(gid_t rgid, gid_t egid);
```

- **Swapping of the real UID and the effective UID**
  - Either the real UID can be set to the effective UID, or the effective UID can either be set to the saved set-user ID or the real UID.

```
#include <unistd.h>
int seteuid(uid_t uid);
int setegid(gid_t gid);
```

- **The effective UID can be set to either the real UID or the saved set-user-ID.**
  - For a privileged user, only the effective UID is set to *uid*.

Changing User IDs and Group IDs



Interpreter Files

```
#include "apue.h"
#include <sys/wait.h>
int main(void)
{
   pid_t pid;
   if ((pid = fork()) < 0) {
      err_sys("fork error");
   } else if (pid == 0) { /* child */
   if (execl("/home/sar/bin/testinterp",
                 "testinterp", "myarg1", "MY ARG2", (char *)0)
       < 0)
       err_sys("execl error");
   }
   if (waitpid(pid, NULL, 0) < 0) /* parent */
       err_sys("waitpid error");
   exit(0);
}
```

Figure 8.20

❑ Interpreter files

- #! *pathname [optional-argument]*
- The actual file got `exec`ed by the kernel is the file specified by the pathname on the first line.
- Interpreter file vs. interpreter

❑ Figure 8.20

```
$ cat /home/sar/bin/testinterp
#!/home/sar/bin/echarg foo
$ ./a.out
argv[0]: /home/sar/bin/echoarg
argv[1]: foo
argv[2]: /home/sar/bin/testinterp
argv[3]: myarg1
argv[4]: MY ARGS2
```

`system` Function

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

- An interface to a shell (not to OS)
- Implemented by calling `fork`, `exec`, and `waitpid`
- Three different types of return values
  - If either the `fork` fails or `waitpid` returns an error other than `EINTR`, -1 with `errno` set to indicate the error.
  - If the `exec` fails, the return value is as if the shell had executed `exit`(127).
  - Otherwise, the return value from `system` is the termination status of the shell.
- Figure 8.22 & Figure 8.23

`system` Function

```
#include <sys/wait.h>
#include <errno.h>
#include <unistd.h>
int system(const char *cmdstring) /* version without signal handling */
{
   pid_t pid;
   int status;
   if (cmdstring == NULL)
      return(1); /* always a command processor with UNIX */
   if ((pid = fork()) < 0) {
      status = -1; /* probably out of processes */
   } else if (pid == 0) { /* child */
      execl("/bin/sh", "sh", "-c", cmdstring, (char *)0);
      _exit(127); /* execl error */
   } else { /* parent */
      while (waitpid(pid, &status, 0) < 0) {
         if (errno != EINTR) {
            status = -1; /* error other than EINTR from waitpid() */
            break;
         }
      }
   }
   return(status);
}
```

Figure 8.22

system Function

```
#include "apue.h"
#include <sys/wait.h>
int main(void)
{
   int status;
   if ((status = system("date")) < 0)
      err_sys("system() error");

   pr_exit(status);

   if ((status = system("nosuchcommand")) < 0)
      err_sys("system() error");

   pr_exit(status);

   if ((status = system("who; exit 44")) < 0)
      err_sys("system() error");

   pr_exit(status);
   exit(0);
}
```

Figure 8.23

Figure 8.22

`system` Function

- **An advantage over `fork` and `exec`, is that `system` does all the required error/signal handling.**
- **A security hole if we call `system` from a set-user-ID program**
  - A program with set-user-ID or set-group-ID should use `fork` and `exec` directly, being certain to change back to normal permission after the `fork`, before calling `exec`.
    (<u>no `system` with `setuid/setgid` programs</u>)

Thank you for your attention !!

Q and A