Signal Handler (why sigaction())

can restore original handling of signal

can block other signals during execution of handler

Signal Handler (sigaction)

```
int sigaction(int sig,
              const struct sigaction *act,
              struct sigaction *oact);

 struct sigaction {
         union {         /* signal handler */
                 void    (*__sa_handler)(int);
                 void    (*__sa_sigaction)(int, siginfo_t *, void *);
         } __sigaction_u;
         sigset_t sa_mask;          /* signal mask to apply */
         int      sa_flags;         /* see signal options */
 };
```


Signal Handler (use of sigaction)

```
...
struct sigaction action;
sigset_t signal_mask;

/* all signals to be blocked during handler() */
sigfillset(&signal_mask);
/* fill action structure */
action.sa_handler = handler;
action.sa_mask = signal_mask;
action.sa_flags = 0;
/* install handler */
sigaction (SIGTERM, &action, NULL);
...
```


Signal Handler (print signal names)

`void psignal(unsigned sig, const char *s);`

print message according to signal number `sig`

`char * strsignal(int sig);`

return pointer to message according to signal number `sig`

Examples (1)

ftpd.c – SIGCHLD $\rightsquigarrow$ wait for child processes

```
3273 void
3274 reapchild(int signo)
3275 {
3276         while (waitpid(-1, NULL, WNOHANG) > 0);
3277 }
```


Examples (2)

ftpd.c – SIGURG $\rightsquigarrow$ handle urgend TCP data

```
 223 static  volatile sig_atomic_t recvurg;
...
2754 static void
2755 sigurg(int signo)
2756 {
2757
2758         recvurg = 1;
2759 }
2760
```


Examples (3)

ftpd.c – SIGQUIT $\rightsquigarrow$ handle quit from keyboard

```
 666 static void
 667 sigquit(int signo)
 668 {
 669
 670         syslog(LOG_ERR, "got signal %d", signo);
 671         dologout(1);
 672 }
```


Signal Handler (summary)

handler = *exception handling*

the handler should be...

- short: do only one thing
- indicating its use in a global variable `volatile int`
- not time–consuming
- not implementing functional features
- not continue on program bugs (SIGBUS, SIGSEGV, SIGFPE)

`sigaction()` preferred to `signal()`

Signal Handler and Threads

- one process, many threads within same PID
- which thread gets the signal ?

unspecified

threads may block signal

$\rightsquigarrow$ one of the threads which do not block the signal

5. User Identities

Users and Groups

unique identifier for each user is a *numeric UID* (user id),

UID=0 is super user, usually called `root`

a user is member of one or more groups

one group is the principal group, the one found in `/etc/passwd`

this group is used as group owner for files the user created, unless he uses `newgrp`

other group memberships are located in `/etc/group`

Which UIDs do processes have?

- RUID: real User–ID
  who starts the process
  this is also inherited from parent processes
- EUID: effective User–ID
  decides about access to system ressources

these two are different only if setuid–bit set

```
-r-sr-xr-x  2 root  wheel  5828 Jan 12 08:41 /usr/bin/passwd
```

this is controlled by the system call `execve()`

Who am I?

Which UID do I have? Command Shell:

```
$ id
uid=2030(sysi30) gid=1000(stud) groups=1000(stud)
```

Which UID do I have? C–program:

```
uid_t u; /* this usually is a 16--bit--integer */
u=getuid();
```

There is a command

```
$ who am I
```

but it doesn't really show who I am ...

```
root             ttyp1    Jun 13 23:20 (localhost)
```

... but who that terminal belongs to.

Changing UIDs within a process (1)

Why should we want to do that?

- permission issues that are not solved by the filesystem
- security *principle of least privilege*

Changing UIDs within a process (2)

- from the command–shell: use command `su`
  - need username + password
  - with option ,,–" simulate login
- from a C–program: use functions `setuid()`, `seteuid()`
  - process has real UID and effective UID
  - only allowed to switch between original RUID and EUID
  - this is implemented storing the EUID on startup in a third (hidden) *saved UID*

set(e)uid() fails, if... `the user is not the super user and the ID specified is not the real, effective ID, or saved ID.`

su–command, simulating login

```
# su dweber
[t] u@h(#)#


# su - dweber
[13:38:44] dweber@isl-c-01(1)$
```

Managing UIDs: Programmer's View

functions that work with UIDs

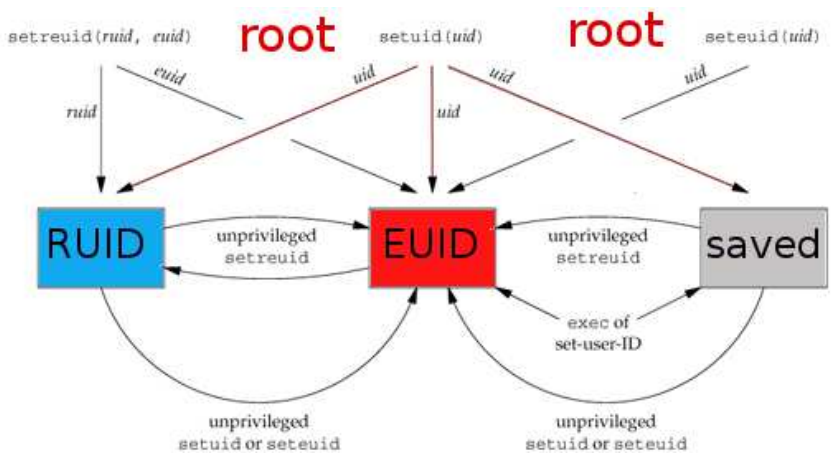
`setuid()` different historical implementations

⇝*setuid mess*, read article

`http://yarchive.net/comp/setuid_mess.html`

`getuid()`	return real UID
`geteuid()`	return effective UID
`setuid()`	set effective UID (root: EUID+RUID)
`seteuid()`	set effective UID
`setreuid()`	set real and effective UID

Managing UIDs: Programmer's View (2)



Managing UIDs: Programmer's View (3)

consequence/portability:

- use *seteuid()* whenever possible
- use *setuid()* to drop all privileges

`/etc/passwd` Entries: Programmer's View

Managing Users: Files (Linux/Solaris)

- user database is /etc/passwd

```
root:x:0:0:root,,,:/root:/bin/bash
ftpadmin:x:1004:1003:FTP Administrator:/home/ftp:/bin/tcsh
ftp:x:40:2:ftp account:/usr/local/ftp:/bin/true
user1:x:1001:1000:Some User:/home/user1:/bin/bash
^^^^^ ^ ^^^^ ^^^^ ^^^^^^^^^ ^^^^^^^^^^^ ^^^^^^^^^
name  x UID   GID description home dir   shell
```

note: a user may change his default shell by chsh.

- encrypted passwords are stored in /etc/shadow

```
user1:PkEZq9UQ/d9v.:11991:0:99999:7:::
user2:Ye4OQU1rqln8s:11992:0:99999:7:::
user3:aeFRGxtgxth.s:12064:0:99999:7:::
user4:KsiYtUF6aGDUQ:12123:0:99999:7:::
^^^^^ ^^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^^^
name, encrypted passwd, password aging info
```


Managing Users: Files (BSD)

- user database is /etc/master.passwd

```
-rw-------  1 root  wheel  /etc/master.passwd
root:$1$8Mj4lzRQ$...:0:0::0:0:Charlie &:/root:/bin/csh
toor:*:0:0::0:0:Bourne-again Superuser:/root:
daemon:*:1:1::0:0:system processes:/root:/usr/sbin/nologin
user1:*:1000:1000::0:0:System &:/:/bin/csh
^^^^^ ^ ^^^^ ^^^^ ^^ ^ ^^^^ ^^^ ^ ^^^^^^^^^^^^^^^^^
name  PWD UID   GID class pwd-change expire descr. home dir shell
```

- copy w/o passwords is stored in /etc/passwd

```
-rw-r--r--  1 root  wheel  1357 Mar 12 12:35 /etc/passwd
root:*:0:0:Charlie &:/root:/bin/csh
toor:*:0:0:Bourne-again Superuser:/root:
daemon:*:1:1:system processes:/root:/usr/sbin/nologin
user1:*:1000:1000:System &:/:/bin/csh
```


The Password Encryption/Hash

iterated versions of the following encryption– or hash–methods

- DES cipher:
  encrypt 000...0
  with key = password (25 iterations)
- Blowfish cipher
- MD5 hash (1000 iterations)
- SHA–256 hash
- SHA–512 hash

cryptographic hash functions work like this:

see manual page crypt(3) for more information

Examples

outdated DES is default method (and fallback) for passwords

```
openssl passwd -salt AbCdEfG secret_password
Warning: truncating password to 8 characters
AbKLsS6u5sAh6
```

several systems today use MD5

```
openssl passwd -1 -salt AbCdEfG secret_password
$1$AbCdEfG$PPiziSx3vbgV1HnIvpJAZ0
```

Attacks

invert Hashing / Encryption
⇝analyze algorithm, very hard (crypto research topic)

dictionary attack (variations of dictionary words)

brute force (= exhaustive search)

Dictionary Attack

```
Aachen          250d6e3dc34afb195271904349fcf790
Aachener        bb6fae8a70240eb9f26b0c8a53345d08
Aachenerin      107b911e2cec78856a4676ea3ce16f92
Aachenerinnen 657b25a7aff45f9434c36d4b1479cde3
Aachenern       6bc4b0cbdda46a3c30b19d3a1a6fbf5c
...
zytotoxischer 9b64262fe97427370242dbc4061722ba
zytotoxisches 1efec802b37771252068b36ee1ce0067
zzgl            71832d182a57a01f13b11014a1264cf7
```

135,000 words in German Duden (2^{17})

Brute Force Attack

```
00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000002
00000000000000000000000000000003
00000000000000000000000000000004
...
ffffffffffffffffffffffffffffffff
```

2^{128} bit strings of length 128