7-UNIX

Environment

User and Group ID

Material from Chapter 8, Users and Groups

Material from Chapter 9, Process Credentials

Material from Chapter 15, File Attributes

User and Group ID

GID – Real Group ID.

Users can belong to one or more groups.

UID – Real User ID.

Identifies the user who is responsible for the running process.

EGID – Effective Group ID. The ID that matters.

EUID - Effective User ID. The ID that matters.

A user with effective user ID of zero has all the privileges of the superuser.

It is called a *privileged process*.

Effective User ID - *euid*

Used to assign ownership of newly created files, to check file access permissions, and to check permission to send signals to processes.

To change ***euid***:

- executes a setuid-program that has the set-uid bit set
- or invokes the setuid( ) system call.

The **setuid(uid)** system call:

if euid is not superuser,

- uid must be the real uid
- or the saved uid (the kernel also resets euid to uid).

Real and effective uid: inherit (fork), maintain (exec).

Real ID Functions

pid_t getuid(void);

Returns the real user ID of the current process

pid_t geteuid(void);

Returns the effective user ID of the current process

gid_t getgid(void);

Returns the real group ID of the current process

gid_t getegid(void);

Returns the effective group ID of the current process

Change UID and GID (1)

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

Sets the effective user ID of the current process.

Superuser process resets the real effective user IDs to ***uid***.

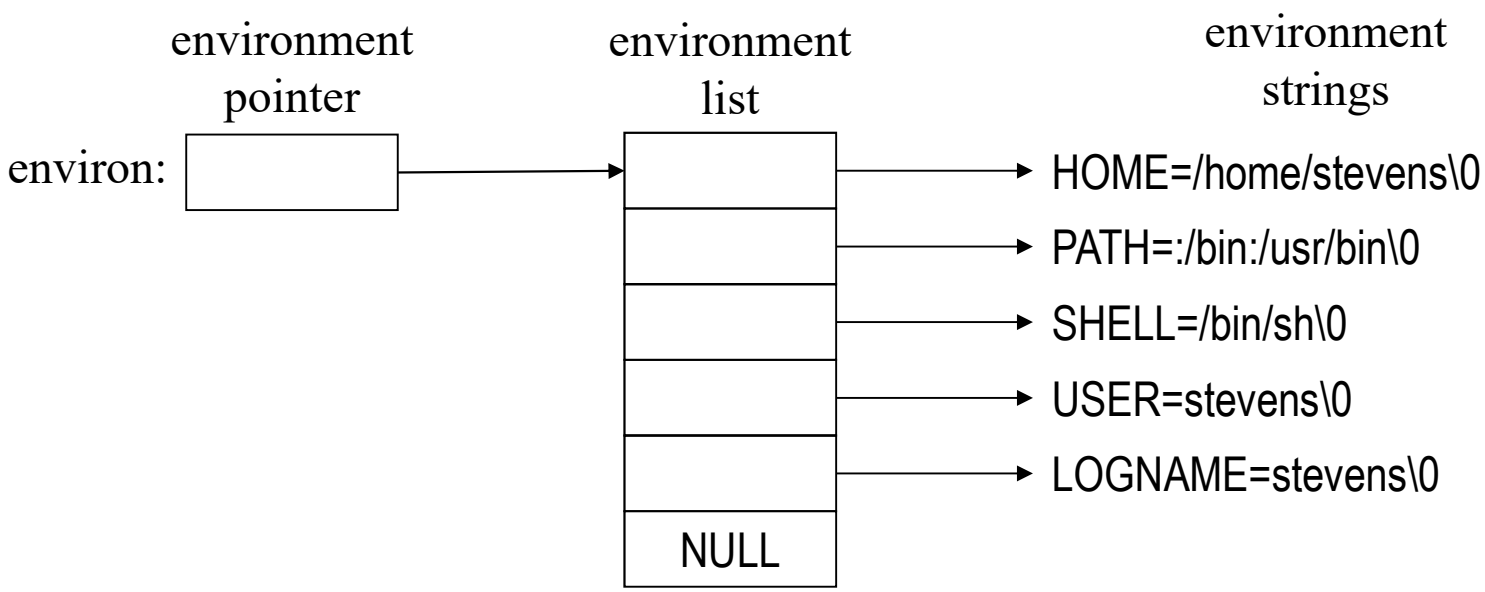
Non-superuser process can set effective user ID to ***uid***, only when ***uid*** equals real user ID or the saved set-user ID (set by executing a setuid-program in exec).

In any other cases, ***setuid*** returns error.

Environment

extern char **environ;

int main(int *argc*, char **argv[ ]*, char **envp[ ]*)



Example: environ

```
#include <stdio.h>

void main( int argc, char *argv[], char *envp[] )
{
  int i;
  extern char **environ;

  printf( "from argument envp\n" );

  for( i = 0; envp[i]; i++ )
     puts( envp[i] );

  printf("\nFrom global variable environ\n");

  for( i = 0; environ[i]; i++ )
     puts(environ[i]);
}
```

getenv (Page 127)

```
#include <stdlib.h>
 char *getenv(const char *name);
```

Retrieves individual values from the process environment.

Searches the environment list for a string that matches the string pointed to by *name*.

Returns a pointer to the value in the environment, or NULL if there is no match.

putenv (page 128)

```
#include <stdlib.h>
    int putenv(const char *string);
```

- Adds or changes the value of the calling process's environment variables.
- The argument *string* is of the form name=value.
- If name does not already exist in the environment, then string is added to the environment.
- If name does exist, then the value of name in the environment is changed to value.
- Returns zero on success, or -1 if an error occurs.

Example : getenv, putenv

```
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
    printf("Home directory is %s\n", getenv("HOME"));

    putenv("HOME=/");

    printf("New home directory is %s\n", getenv("HOME"));
}
```

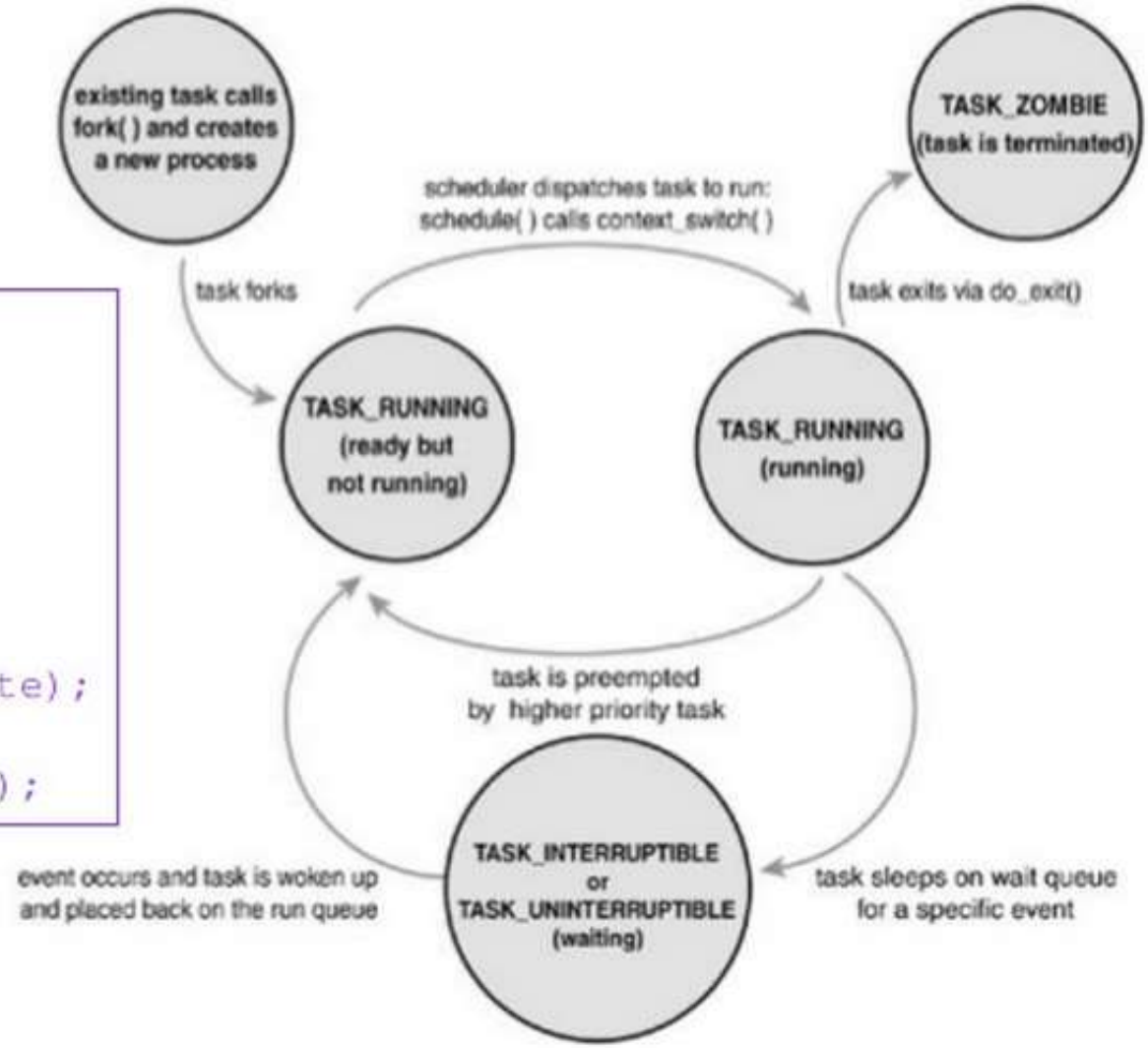
Linux Scheduling

Re-typed on next two slides.

[include/linux/sched.h]

```
TASK_RUNNING
TASK_INTERRUPTIBLE
TASK_UNINTERRUPTIBLE
TASK_STOPPED
EXIT_ZOMBIE
EXIT_DEAD

set_task_state(task, state);
task_state = state;
set_current_state( state);
```



Robert Love, "Linux Kernel Development," 2nd Edition

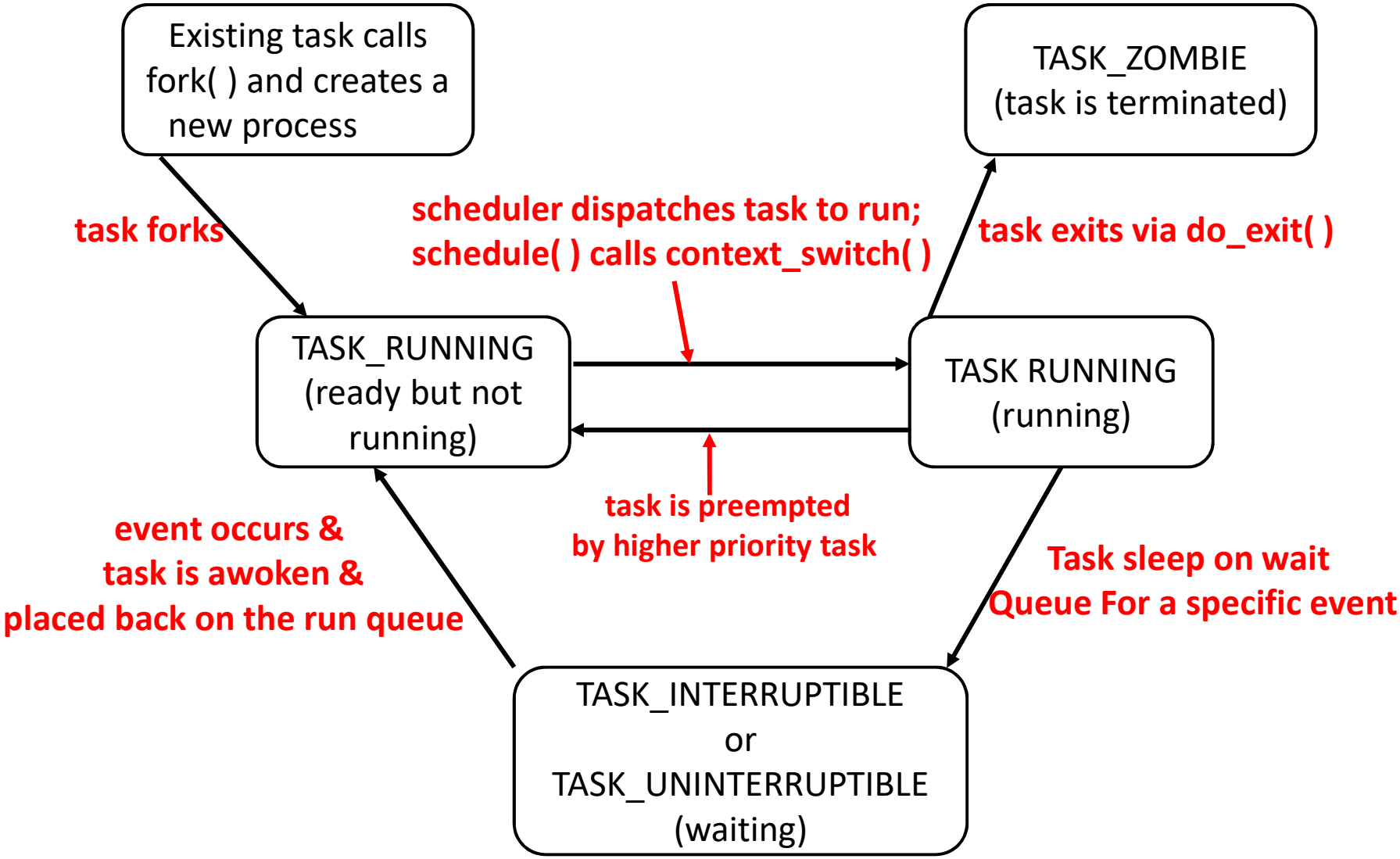
This slide shows the contents of the box on the previous slide.

(include/linux/sched.h)

```
TASK_RUNNING
TASK_INTERRUPTIBLE
TASK_UNINTERRUPTIBLE
TASK_STOPPED
EXIT_ZOMBIE
EXIT_DEAD
```

```
set_task_state(task, state);
task_state = state;
set_current_state ( state );
```

Linux Scheduling



Process State Codes (from *ps* command)

PROCESS STATE CODES

Here are the different values that the s, stat and state output specifiers (header "STAT" or "S") will display to describe the state of a process.

D Uninterruptible sleep (usually IO)

R Running or runnable (on run queue)

S Interruptible sleep (waiting for an event to complete)

T Stopped, either by a job control signal or because it is being traced.

W paging (not valid since the 2.6.xx kernel)

X dead (should never be seen)

Z Defunct ("zombie") process, terminated but not reaped by its parent.

Process State Codes (from *ps* command)

For BSD formats and when the stat keyword is used, additional characters may be displayed:

< high-priority (not nice to other users)

N low-priority (nice to other users)

L has pages locked into memory (for real-time and custom IO)

s is a session leader

l is multi-threaded (using CLONE_THREAD, like NPTL pthreads do)

+ is in the foreground process group

7-UNIX

Environment

The End