Operating Systems Lab (C+Unix)

Enrico Bini

University of Turin

Outline

Replacement of a process with execve()

- A process can replace its memory segments by another executable

```
#include <unistd.h>

int execve(const char *executable_name,
           char *const argv[],
           char *const envp[]);
```

- executable_name, executable to be launched;
- argv, arguments passed to the launched program (NULL-terminated list)
- envp, variables of the environment of the launched program (NULL-terminated list)
- if successful, execve() does not return
  - the execution continues in executable_name with same PID, same open file descriptors of invoking process
- if not successful, it returns -1 with errno set accordingly
- *test-execve.c*

Effect/usage of `execve()`

- the PID of the invoking process is preserved (no new process)
- the open file descriptors are preserved
- user-defined signal handlers are **cleared** after `execve`
- all memory segments (stack, data, heap, etc.) of the calling process are replaced by the ones of the program called by `execve()`
- can be used to create child processes that execute a given program

```
if(!fork()) {
    execve("to_be_launched", child_args, child_env);
    exit(EXIT_FAILURE);
}
```

- The shell `bash` does exactly this

Launching a command with system()

```
#include <stdlib.h>

int system(const char *command);
```

- system("my_command") launches the executable my_command
- More precisely it does something like

```
/* Roughly analogous to system("my_command"); */
if(!fork()) {
    /* Launch the command */
    execve("my_command", child_args, child_env);
    exit(EXIT_FAILURE);
}
/* Wait for the launched command to terminate */
wait(&returned_status);
/* Return the exit status */
return WEXITSTATUS(returned_status);
```