CSCI 330
The UNIX System

Process Management

Unit Overview

Process Management

- create new process
- change what a process is doing

Error Handling

- System calls and library functions share convention on how to report errors
  - return -1 in return status
  - set global variable **`errno`**
  - **`errno`** is index into table of error messages
- C library function **`perror`** translates this error code into human readable form

Process Management System Calls

- fork
  - create a new process
- wait
  - wait for a process to terminate
- exec
  - execute a program

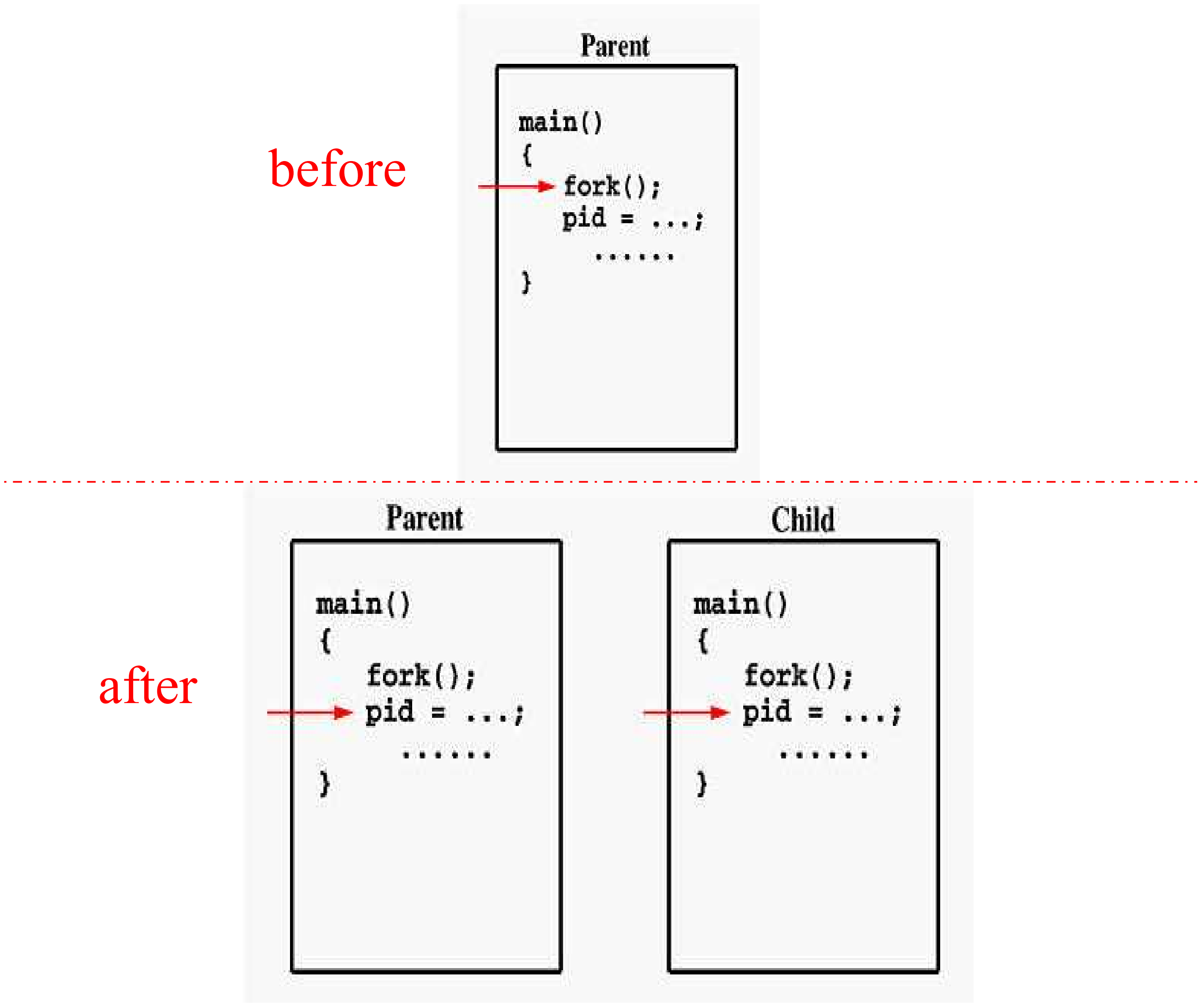
System Call: fork

- creates new process that is duplicate of current process
- new process is almost the same as current process
    - copy of memory space

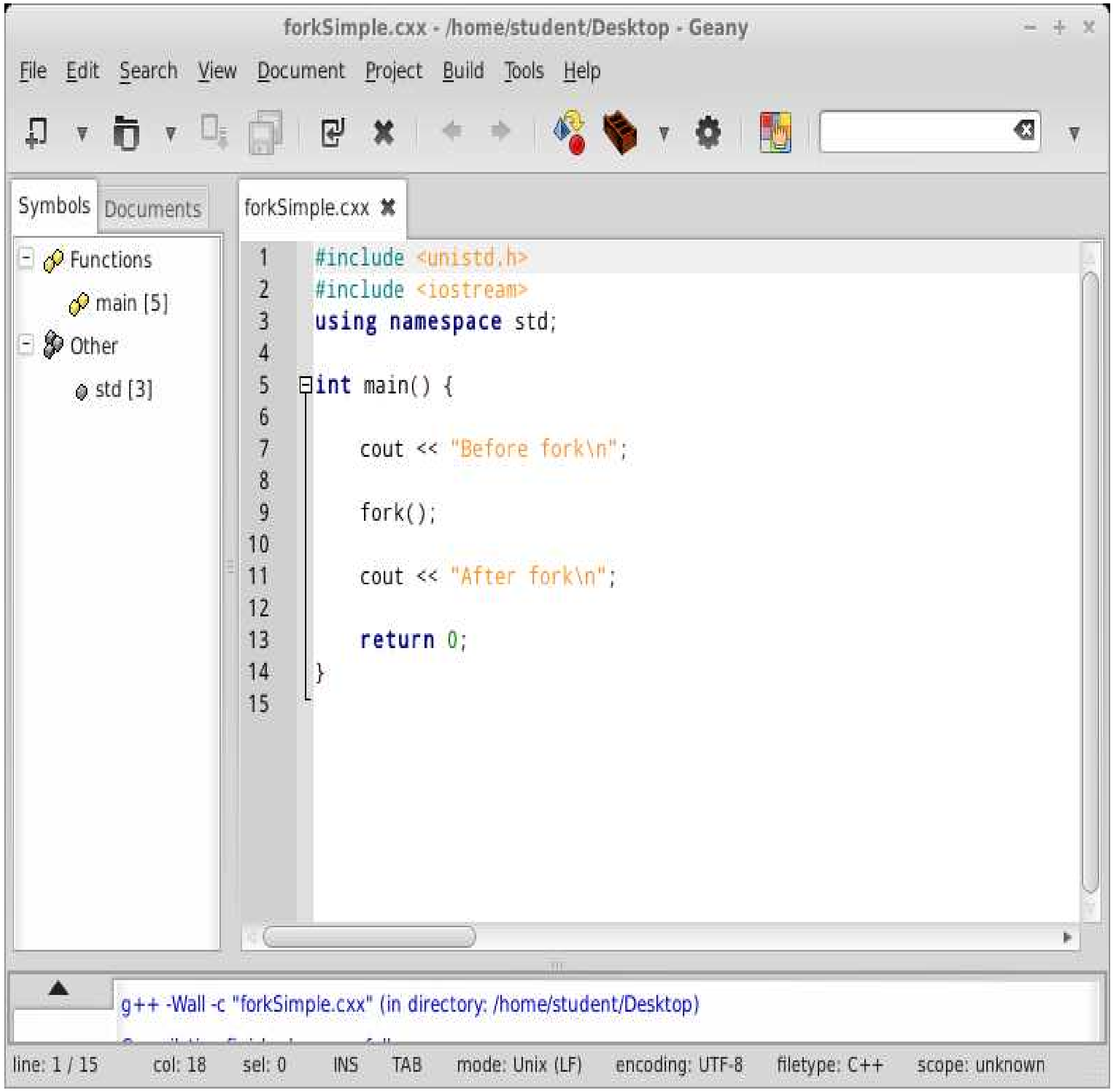
- new process is child of current process
- old process is parent of new process

- after call to fork, both processes run concurrently

System Call: fork timeline



System Call: fork example


forkSimple.cxx - /home/student/Desktop - Geany
File Edit Search View Document Project Build Tools Help
Symbols Documents
forkSimple.cxx
Functions
main [5]
Other
std [3]

```
#include <unistd.h>
#include <iostream>
using namespace std;

int main() {

    cout << "Before fork\n";

    fork();

    cout << "After fork\n";

    return 0;
}

```

```
g++ -Wall -c "forkSimple.cxx" (in directory: /home/student/Desktop)
```

```
line: 1 / 15    col: 18    sel: 0    INS    TAB    mode: Unix (LF)    encoding: UTF-8    filetype: C++    scope: unknown
```

System Call: fork

- new process is almost the same as current process

```
pid_t fork(void)
```

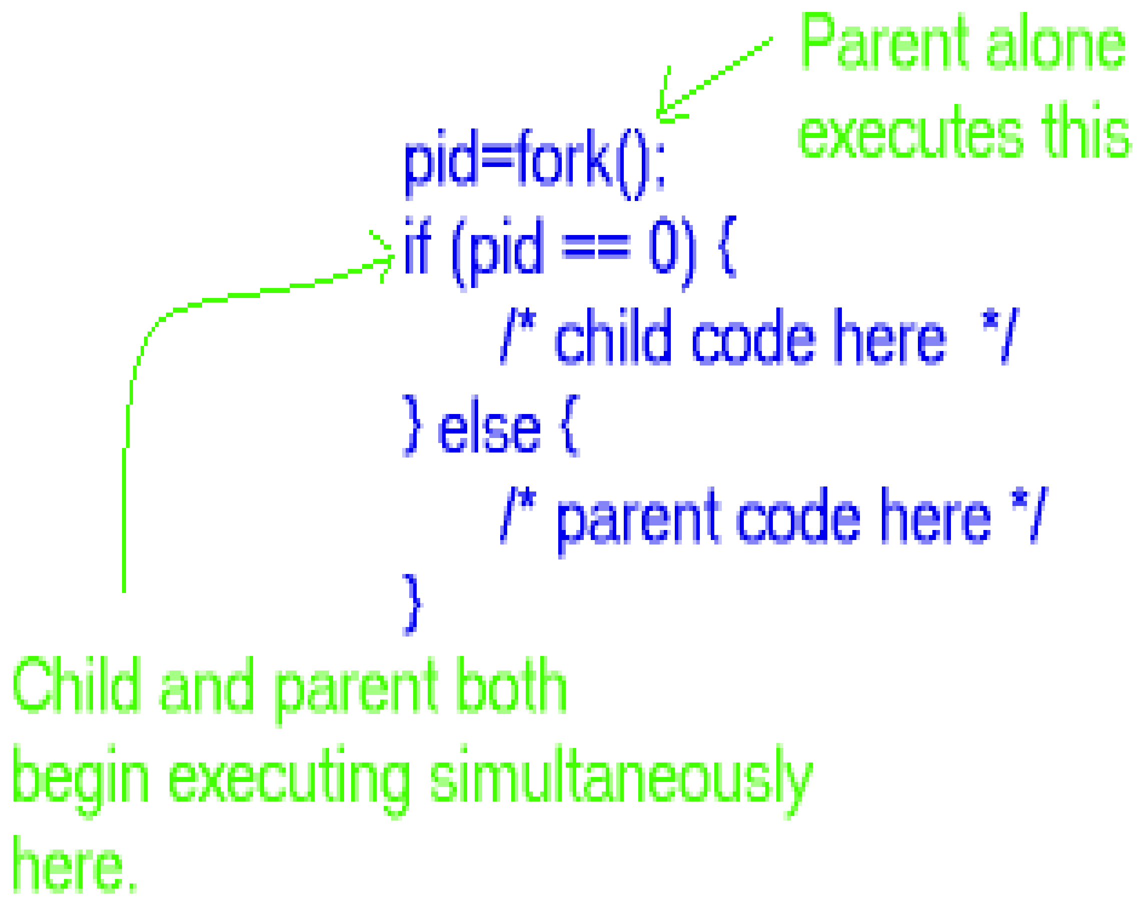
- the return value of fork is different:

  parent: fork returns process id of child process

  child: fork returns 0

- for returns -1 on failure

System Call: fork

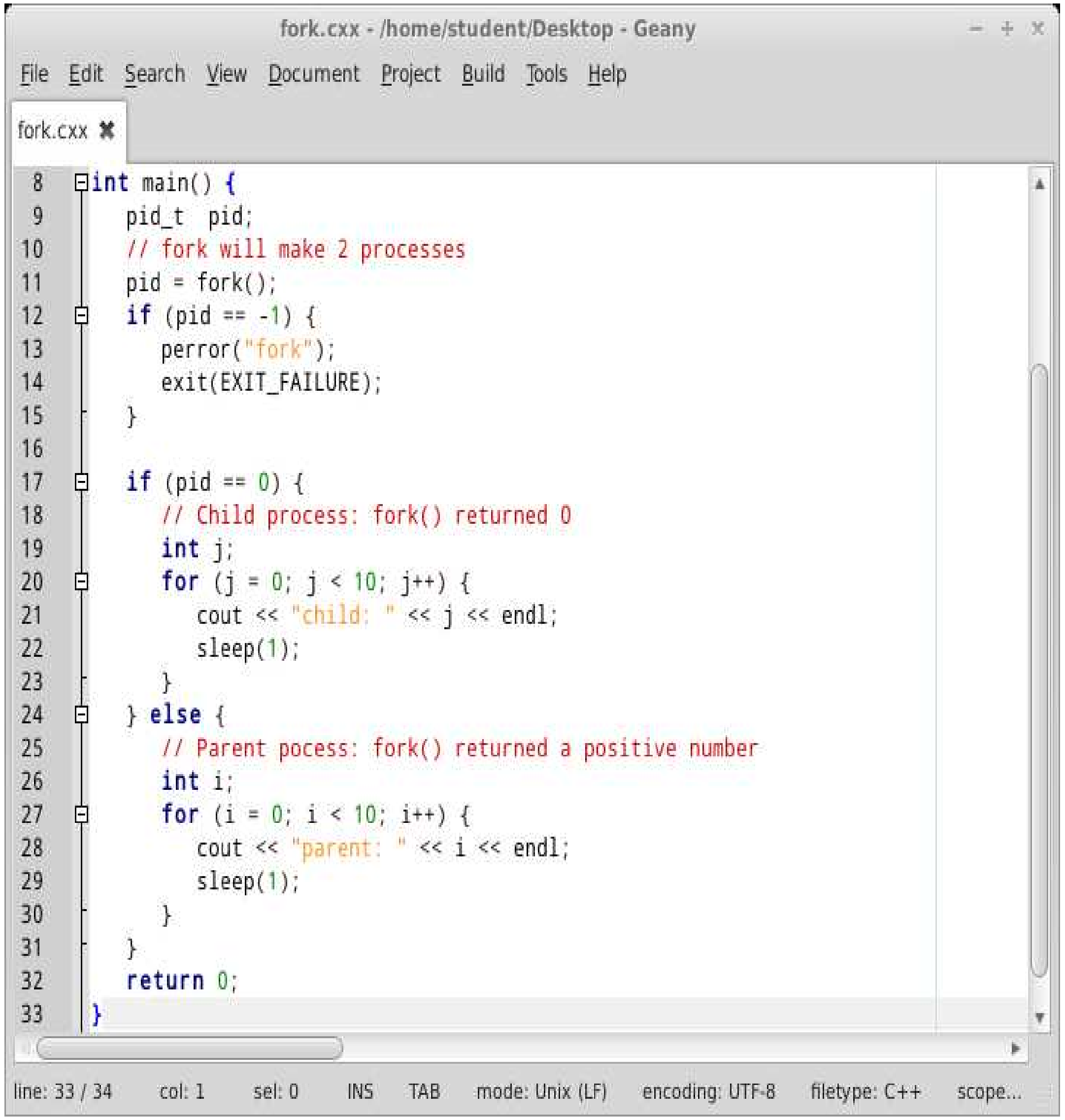


Parent alone executes this

```
pid=fork();
if (pid == 0) {
      /* child code here  */
} else {
      /* parent code here */
}
```

Child and parent both begin executing simultaneously here.

System Call: fork example


fork.cxx - /home/student/Desktop - Geany
File Edit Search View Document Project Build Tools Help
fork.cxx

```
int main() {
    pid_t  pid;
    // fork will make 2 processes
    pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(EXIT_FAILURE);
    }

    if (pid == 0) {
        // Child process: fork() returned 0
        int j;
        for (j = 0; j < 10; j++) {
            cout << "child: " << j << endl;
            sleep(1);
        }
    } else {
        // Parent pocess: fork() returned a positive number
        int i;
        for (i = 0; i < 10; i++) {
            cout << "parent: " << i << endl;
            sleep(1);
        }
    }
    return 0;
}
```

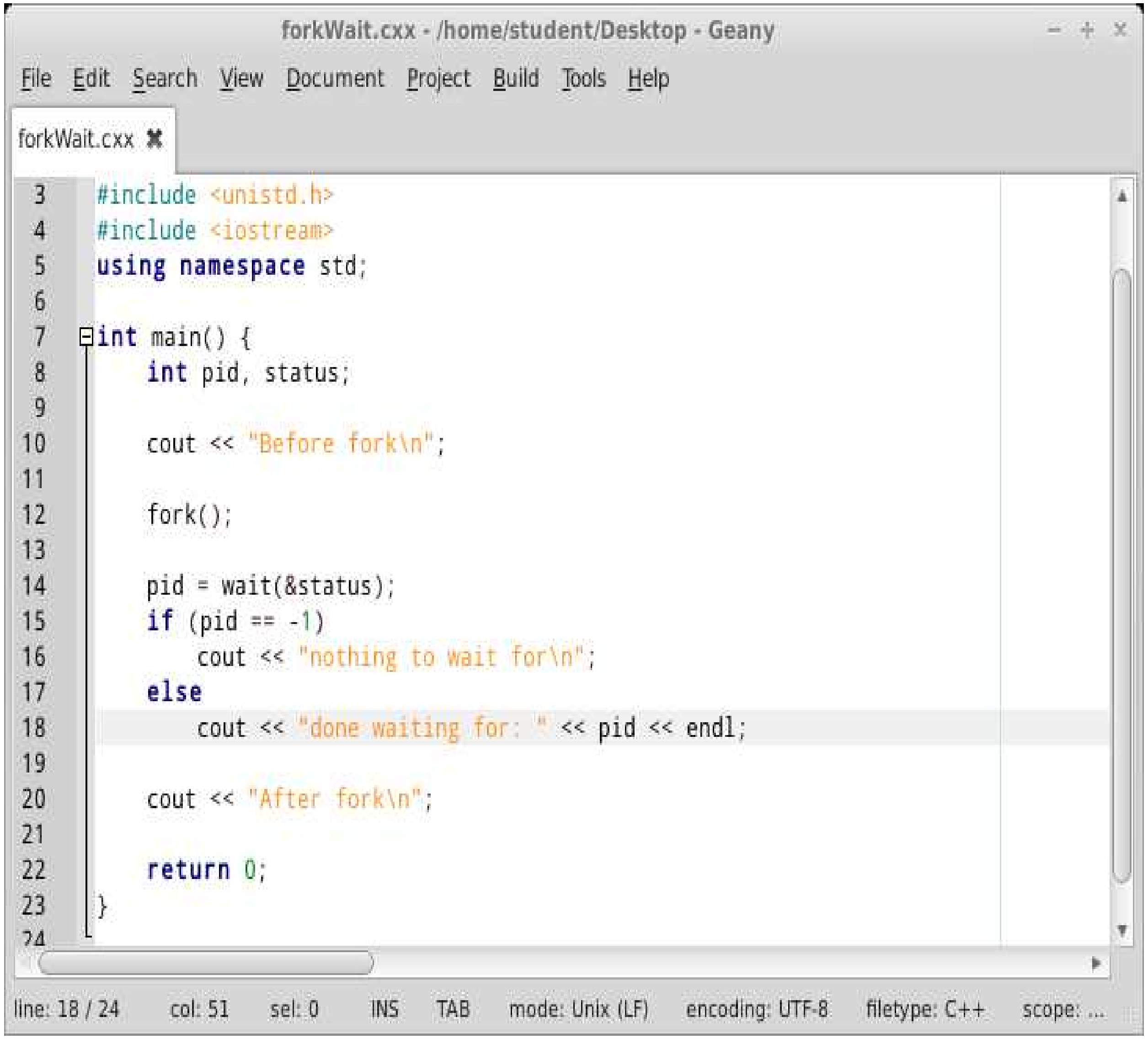
line: 33 / 34 col: 1 sel: 0 INS TAB mode: Unix (LF) encoding: UTF-8 filetype: C++ scope...

System Call: wait

```
pid_t wait(int *status)
```

- lets parent process wait until a child process terminates
  - parent is restarted once a child process terminates
- returns process id of terminated child
  - return -1 if there is no child to wait for
- `status` holds exit status of child

System Call: wait example


forkWait.cxx - /home/student/Desktop - Geany
File Edit Search View Document Project Build Tools Help
forkWait.cxx

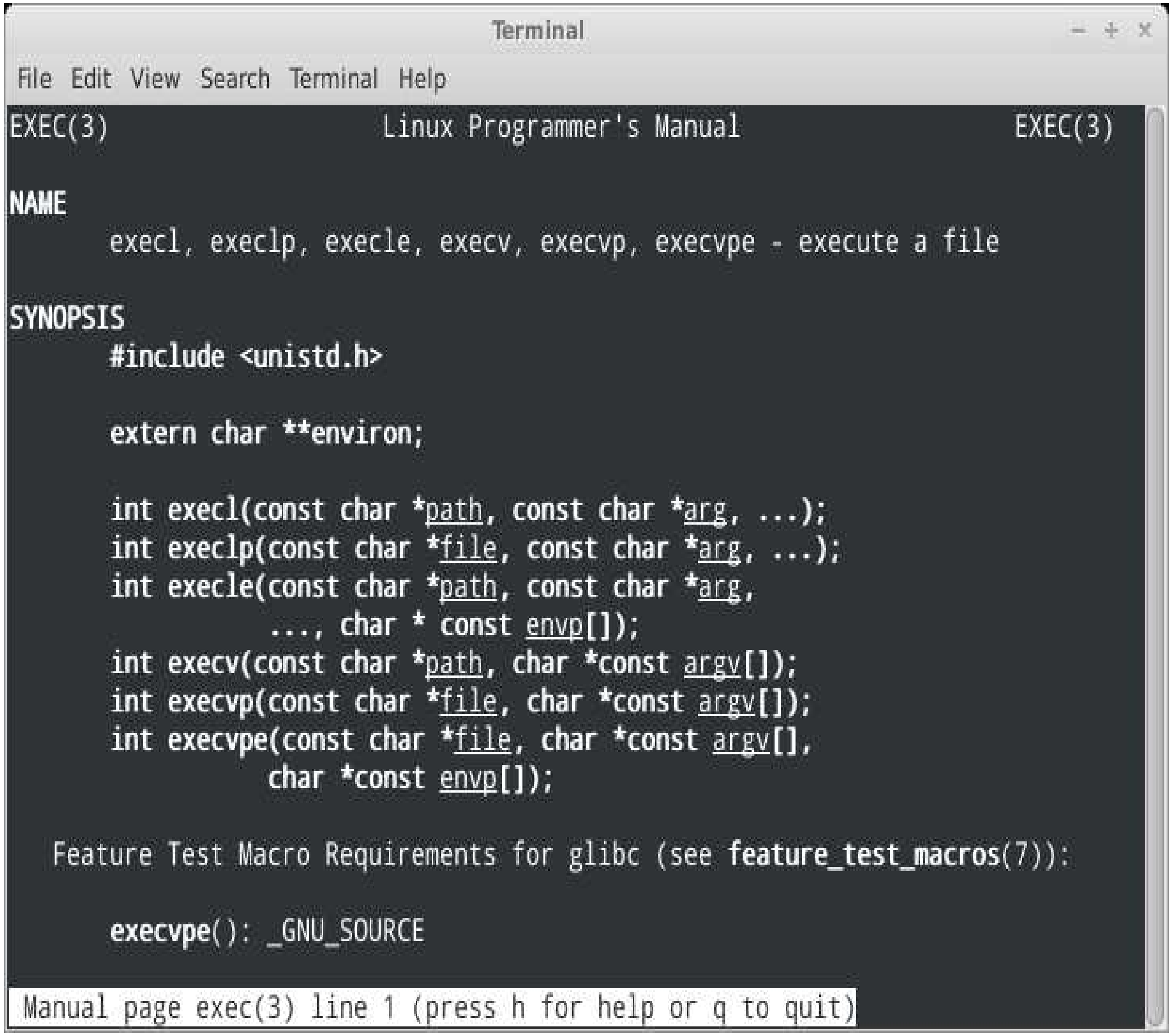
```
#include <unistd.h>
#include <iostream>
using namespace std;

int main() {
    int pid, status;

    cout << "Before fork\n";

    fork();

    pid = wait(&status);
    if (pid == -1)
        cout << "nothing to wait for\n";
    else
        cout << "done waiting for: " << pid << endl;

    cout << "After fork\n";

    return 0;
}
```

line: 18 / 24 col: 51 sel: 0 INS TAB mode: Unix (LF) encoding: UTF-8 filetype: C++ scope: ...

System Call: exec

- family of functions that replace current process image with a new process image
- actual system call: execve
- library functions
  - execl, exexlp, execle
  - execv, execvp
- arguments specify new executable to run and its arguments and environment

C Library Functions: exec



```
EXEC(3)                    Linux Programmer's Manual                   EXEC(3)

NAME
       execl, execlp, execle, execv, execvp, execvpe - execute a file

SYNOPSIS
       #include <unistd.h>

       extern char **environ;

       int execl(const char *path, const char *arg, ...);
       int execlp(const char *file, const char *arg, ...);
       int execle(const char *path, const char *arg,
                  ..., char * const envp[]);
       int execv(const char *path, char *const argv[]);
       int execvp(const char *file, char *const argv[]);
       int execvpe(const char *file, char *const argv[],
                  char *const envp[]);

   Feature Test Macro Requirements for glibc (see feature_test_macros(7)):

       execvpe(): _GNU_SOURCE

 Manual page exec(3) line 1 (press h for help or q to quit)
```

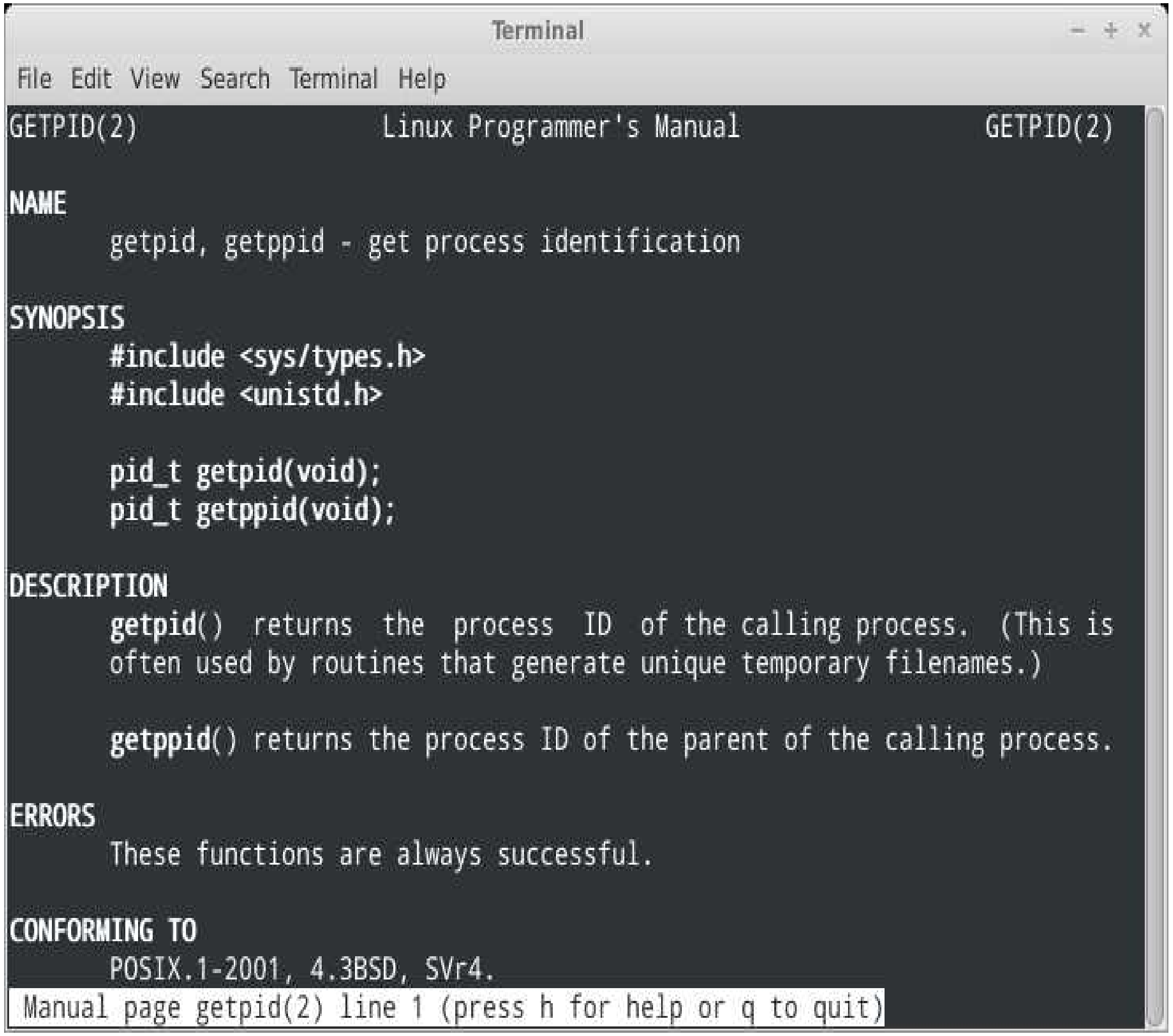
C Library Function: execl

`int execl(const char *path, const char *arg, ...)`

- starts executable for command specified in **`path`**
- new executable runs in current process
- **`path`** is specified as absolute path
- arguments are specified as list, starting at argv[0],

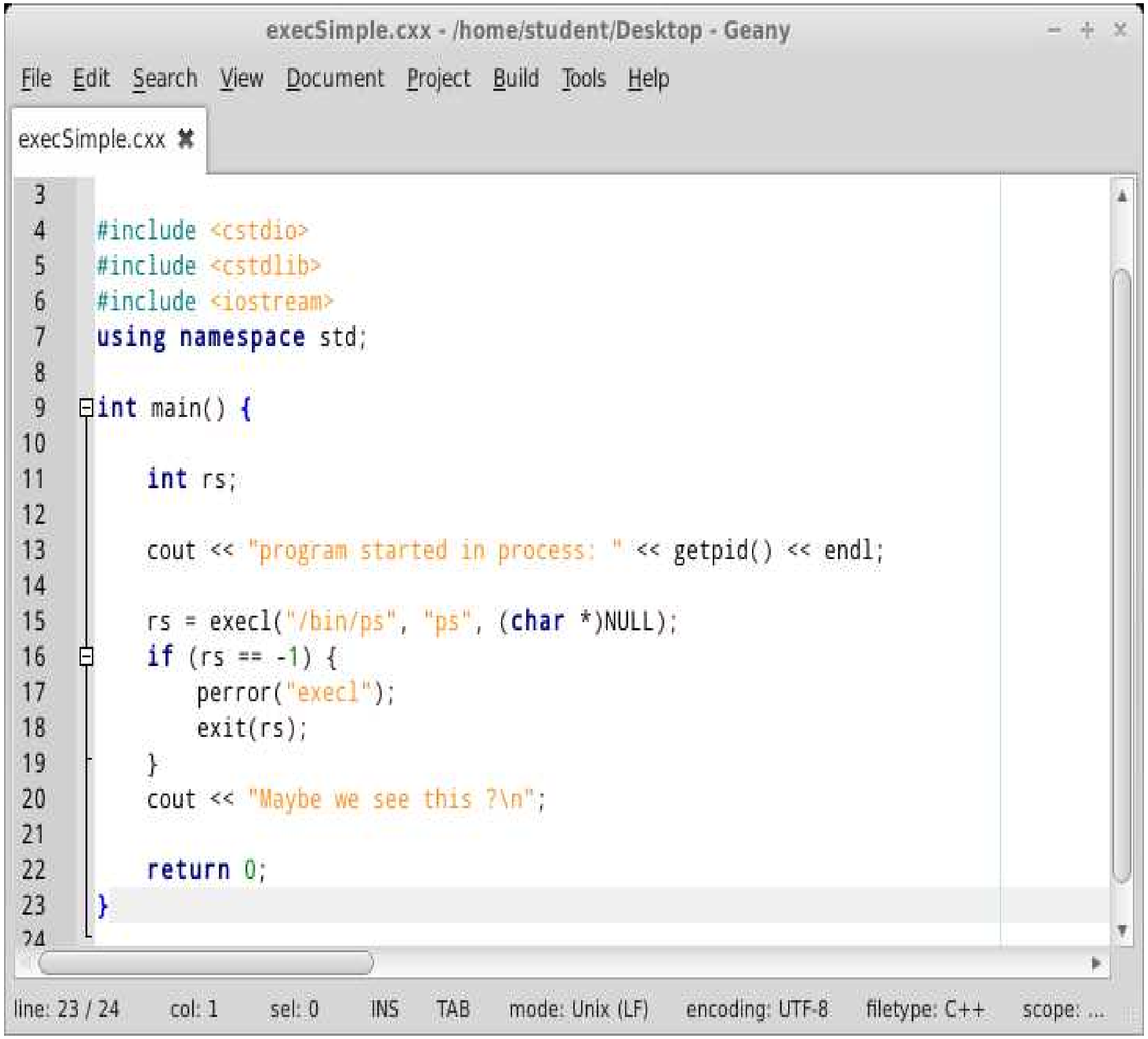
  terminated with **`(char *NULL)`**
- new executable keeps same environment
- return -1 on error

System Call: getpid



```
GETPID(2)                 Linux Programmer's Manual                GETPID(2)

NAME
       getpid, getppid - get process identification

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

       pid_t getpid(void);
       pid_t getppid(void);

DESCRIPTION
       getpid()  returns  the  process  ID  of the calling process.  (This is
       often used by routines that generate unique temporary filenames.)

       getppid() returns the process ID of the parent of the calling process.

ERRORS
       These functions are always successful.

CONFORMING TO
       POSIX.1-2001, 4.3BSD, SVr4.
 Manual page getpid(2) line 1 (press h for help or q to quit)
```

C Library Function: execl


execSimple.cxx - /home/student/Desktop - Geany
File Edit Search View Document Project Build Tools Help
execSimple.cxx

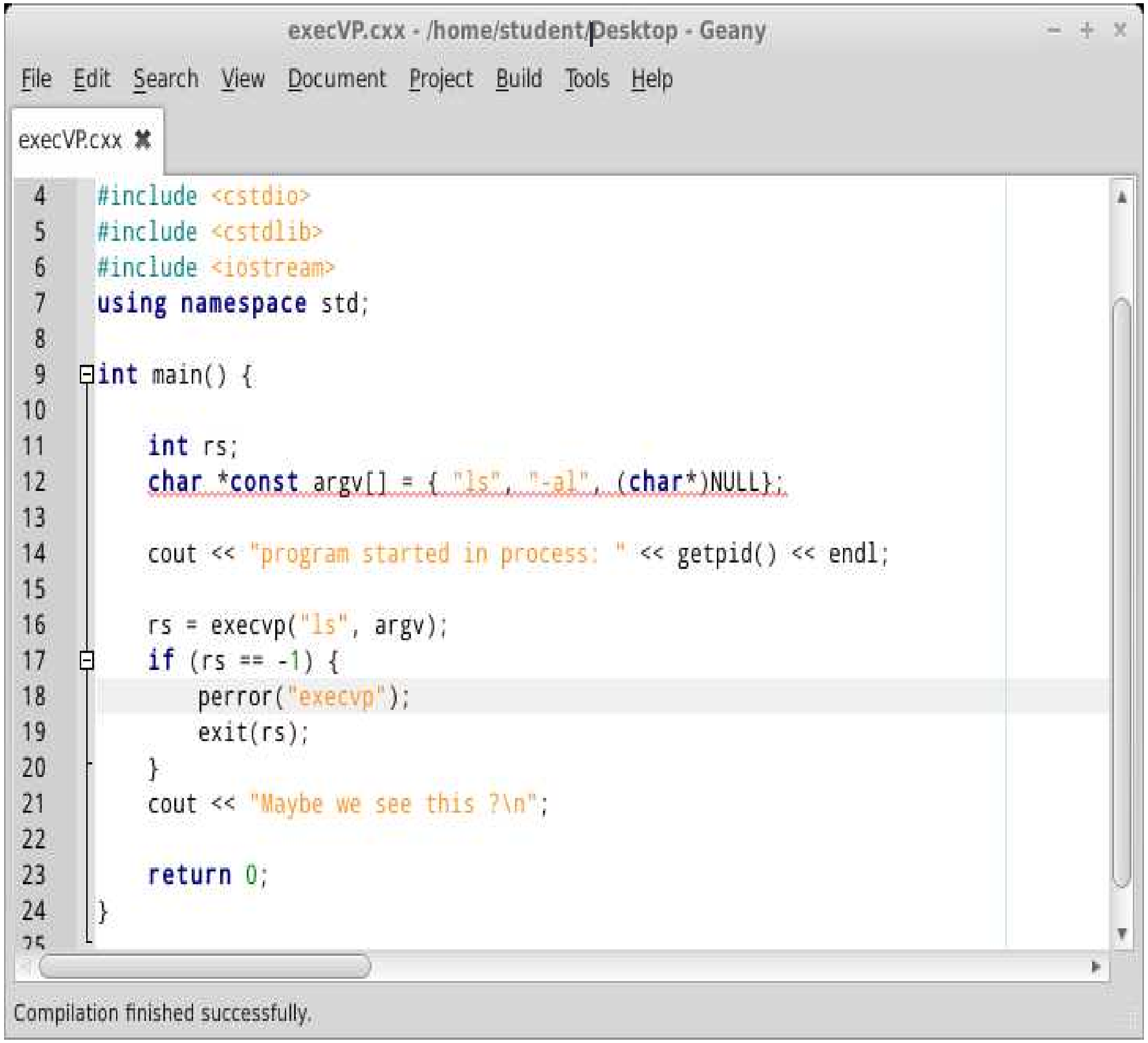
```

#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

    int rs;

    cout << "program started in process: " << getpid() << endl;

    rs = execl("/bin/ps", "ps", (char *)NULL);
    if (rs == -1) {
        perror("execl");
        exit(rs);
    }
    cout << "Maybe we see this ?\n";

    return 0;
}

```

line: 23 / 24   col: 1   sel: 0   INS   TAB   mode: Unix (LF)   encoding: UTF-8   filetype: C++   scope: ...

C Library Functions: exec

- execl, execlp, execle
  - specify arguments and environment as list
- execv, execvp
  - specifiy arguments and environment as array of string values
- execlp, execvp
  - look for new executable via PATH

C Library Function: execv


execVP.cxx - /home/student/Desktop - Geany
File Edit Search View Document Project Build Tools Help
execVP.cxx

```
#include <cstdio>
#include <cstdlib>
#include <iostream>
using namespace std;

int main() {

    int rs;
    char *const argv[] = { "ls", "-al", (char*)NULL};

    cout << "program started in process: " << getpid() << endl;

    rs = execvp("ls", argv);
    if (rs == -1) {
        perror("execvp");
        exit(rs);
    }
    cout << "Maybe we see this ?\n";

    return 0;
}
```

Compilation finished successfully.

Together: fork and exec

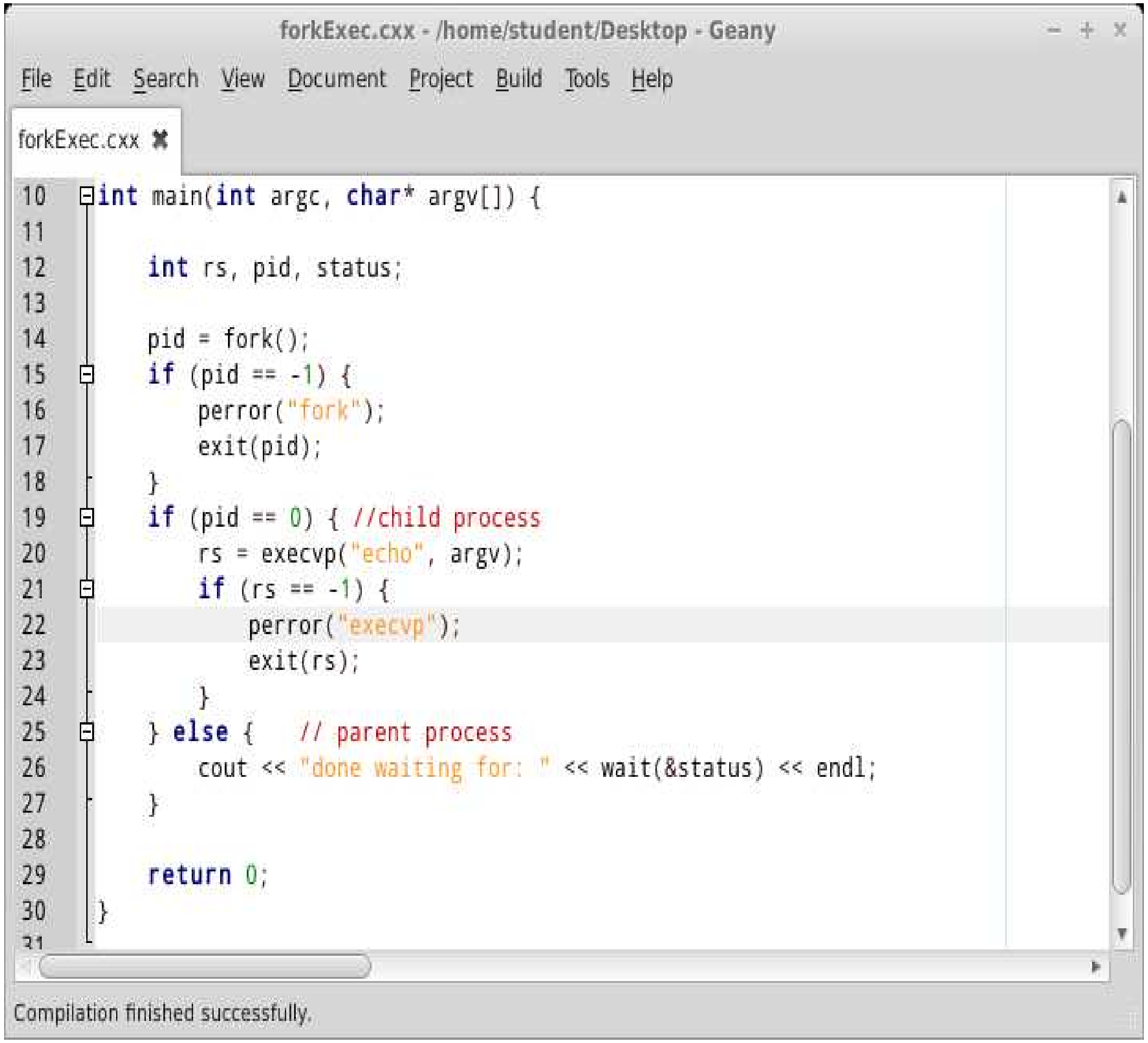
- UNIX does not have a system call to spawn a new

  additional

  process with a new executable

- instead:
  - fork to duplicate current process
  - exec to morph child process into new executable

Together: fork and exec


forkExec.cxx - /home/student/Desktop - Geany
File Edit Search View Document Project Build Tools Help
forkExec.cxx

```
int main(int argc, char* argv[]) {

    int rs, pid, status;

    pid = fork();
    if (pid == -1) {
        perror("fork");
        exit(pid);
    }
    if (pid == 0) { //child process
        rs = execvp("echo", argv);
        if (rs == -1) {
            perror("execvp");
            exit(rs);
        }
    } else {    // parent process
        cout << "done waiting for: " << wait(&status) << endl;
    }

    return 0;
}
```

Compilation finished successfully.

Summary

Process management

`fork()`

`wait()`

`exec()`