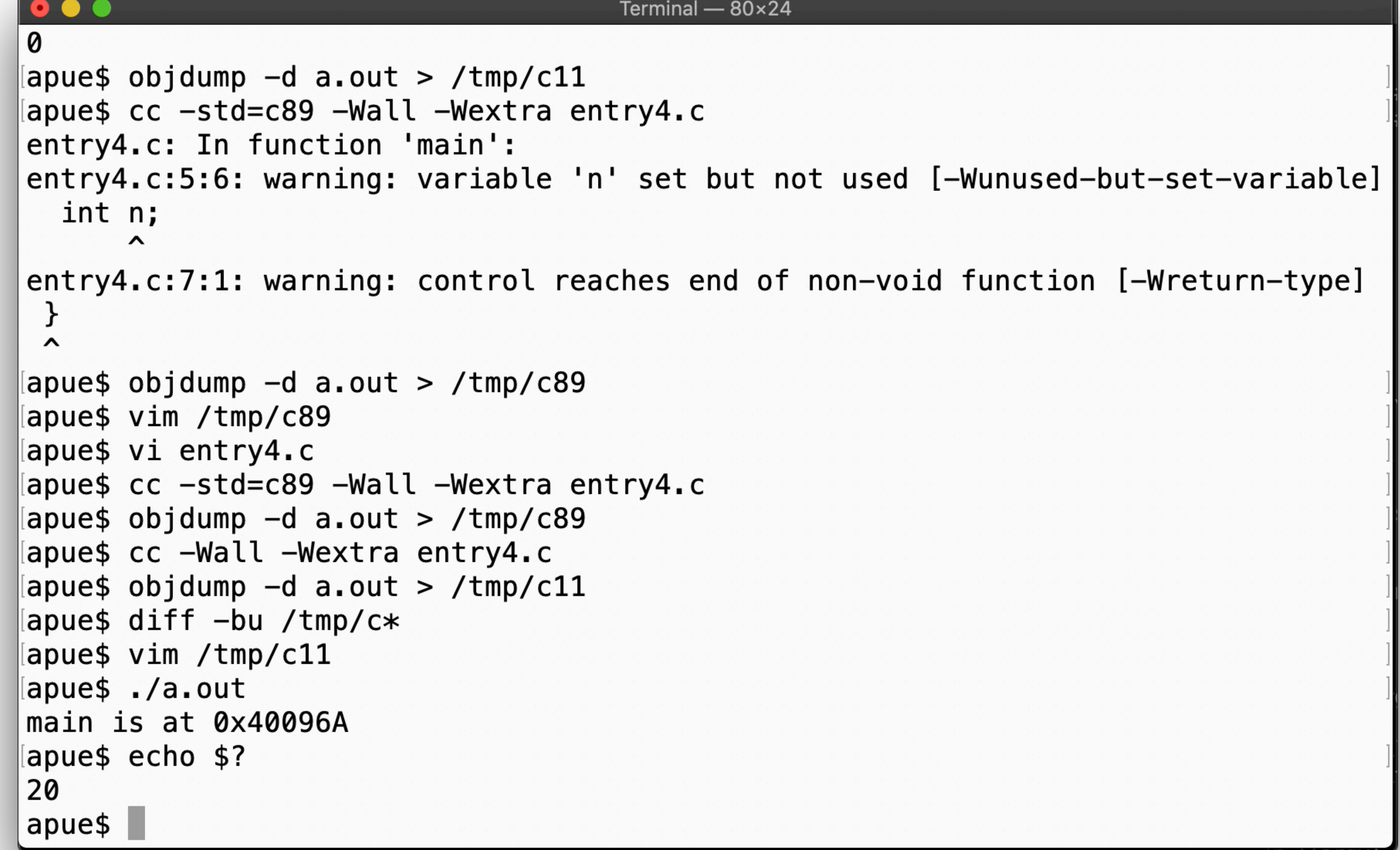
Advanced Programming in the UNIX Environment

Week 06, Segment 3:
Program Termination

Department of Computer Science
Stevens Institute of Technology

Jan Schaumann
jschauma@stevens.edu
https://stevens.netmeister.org/631/



Terminal — 80×24

```
0
apue$ objdump -d a.out > /tmp/c11
apue$ cc -std=c89 -Wall -Wextra entry4.c
entry4.c: In function 'main':
entry4.c:5:6: warning: variable 'n' set but not used [-Wunused-but-set-variable]
  int n;
      ^
entry4.c:7:1: warning: control reaches end of non-void function [-Wreturn-type]
 }
 ^
apue$ objdump -d a.out > /tmp/c89
apue$ vim /tmp/c89
apue$ vi entry4.c
apue$ cc -std=c89 -Wall -Wextra entry4.c
apue$ objdump -d a.out > /tmp/c89
apue$ cc -Wall -Wextra entry4.c
apue$ objdump -d a.out > /tmp/c11
apue$ diff -bu /tmp/c*
apue$ vim /tmp/c11
apue$ ./a.out
main is at 0x40096A
apue$ echo $?
20
apue$ 
```

Process Termination

There are multiple ways for a process to terminate:

Normal termination:

- implicit return from `main`
- explicit return from `main`
- calling `exit(3)`
- calling `_exit(2)` (or `_Exit(2)`)
- return of last thread from its start routine
- calling `pthread_exit(3)` from last thread

Abnormal termination:

- calling `abort(3)`
- termination by a signal
- response of the last thread to a cancellation request

exit(3) and _exit(2)

```
#include <stdlib.h>

void exit(int status);

                                                        Returns: doesn't
```

exit(3) terminates a process. Before termination it performs the following functions in the order listed:

- Call the functions registered with the atexit(3) function, in the reverse order of their registration.
- Flush all open output streams.
- Close all open streams.
- Unlink all files created with the tmpfile(3) function.

Following this, exit(3) calls _exit(2).

exit(3) and _exit(2)

```
#include <unistd.h>

void _exit(int status);

                                                    Returns: doesn't
```

_exit(2) terminates the process immediately.

There are a number of consequences relating to process relationships that we will see in future segments.

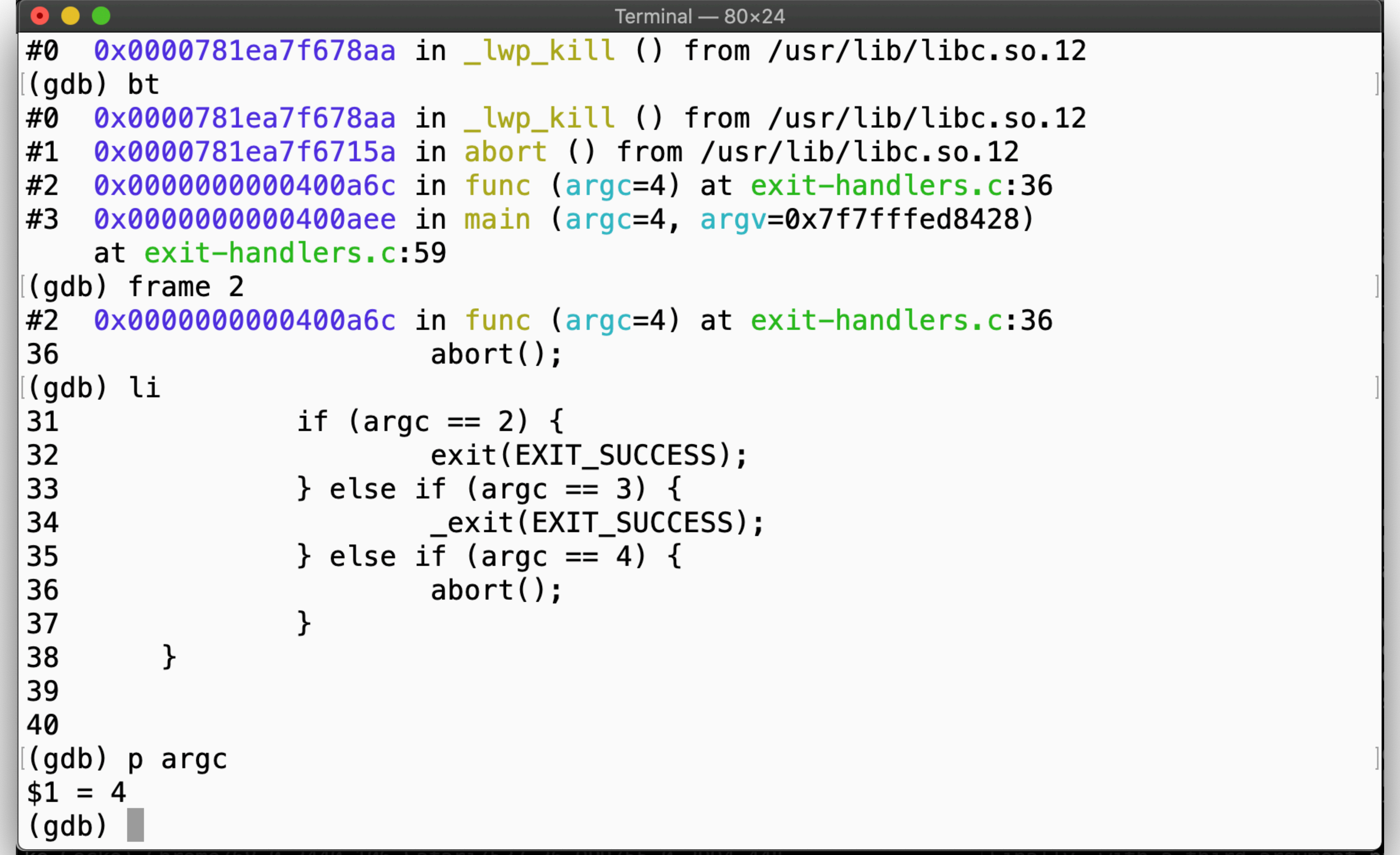
atexit(3)

```
#include <stdlib.h>

int atexit(void (*function)(void));
```

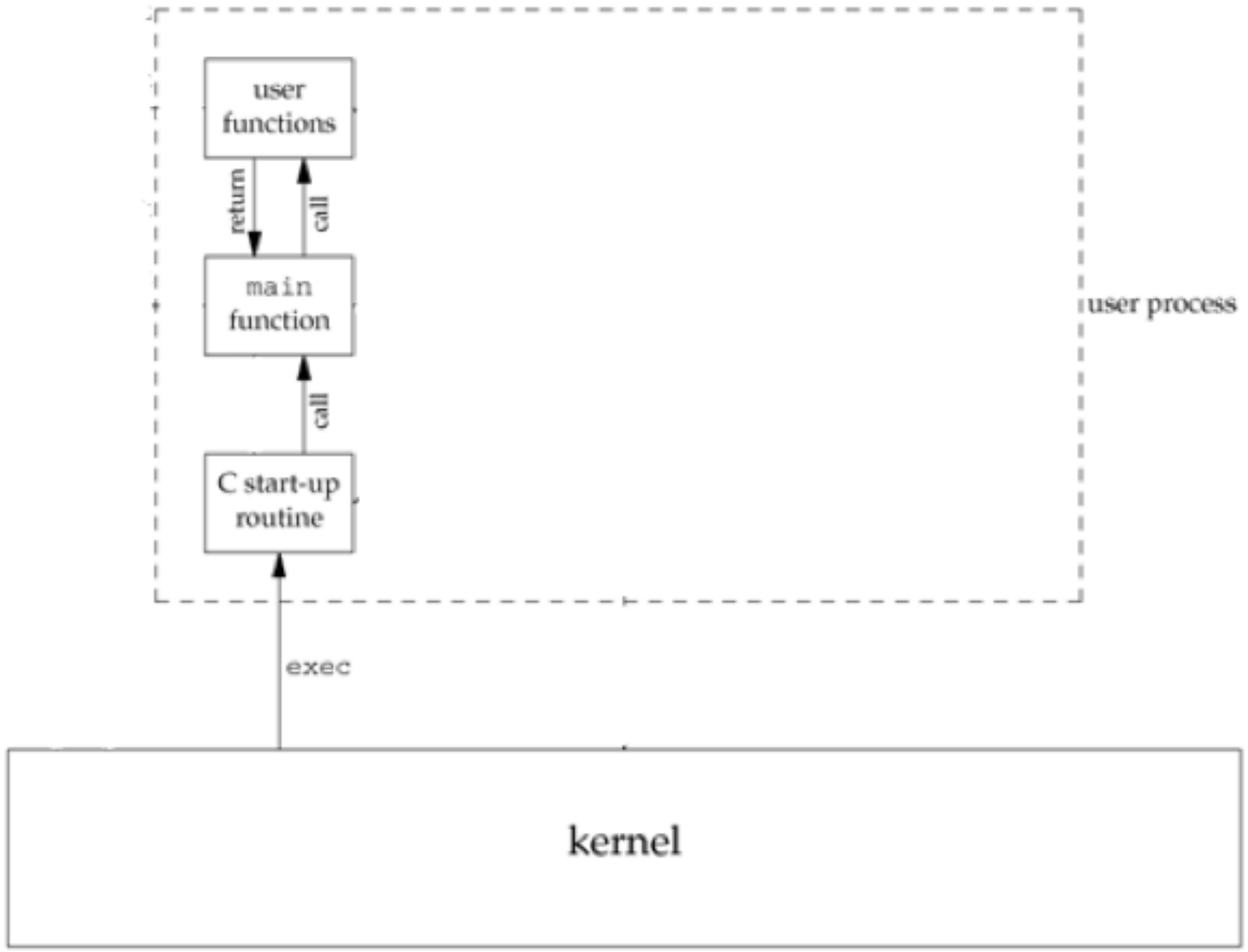
Returns: 0 on success; -1 on error

- registers a function with a signature of `void function(void)` to be called *at exit*
- functions are invoked *at exit* in reverse order of registration
- the same function can be registered more than once
- extremely useful for cleaning up open files, freeing certain resources, etc.

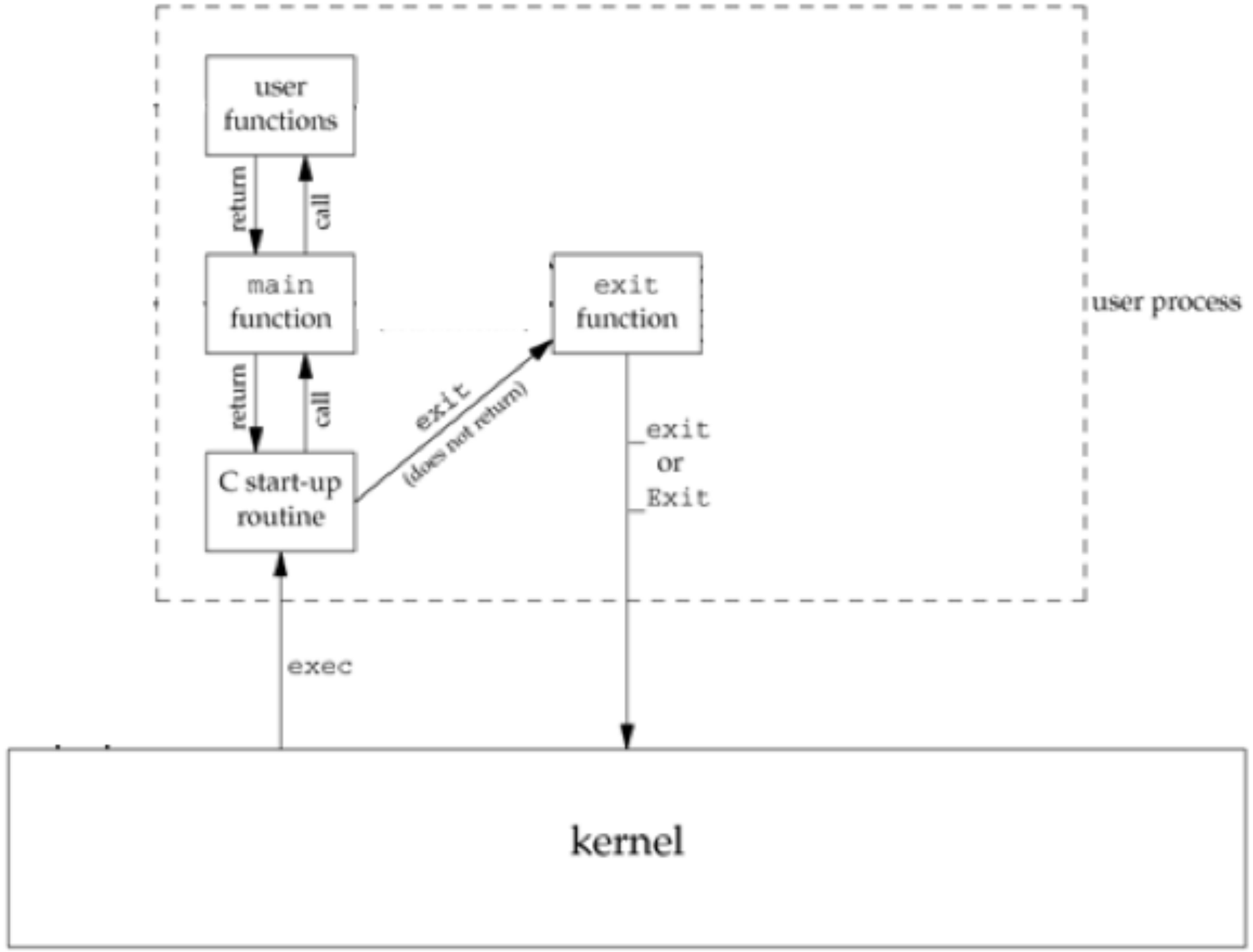


```
#0  0x0000781ea7f678aa in _lwp_kill () from /usr/lib/libc.so.12
(gdb) bt
#0  0x0000781ea7f678aa in _lwp_kill () from /usr/lib/libc.so.12
#1  0x0000781ea7f6715a in abort () from /usr/lib/libc.so.12
#2  0x0000000000400a6c in func (argc=4) at exit-handlers.c:36
#3  0x0000000000400aee in main (argc=4, argv=0x7f7fffed8428)
    at exit-handlers.c:59
(gdb) frame 2
#2  0x0000000000400a6c in func (argc=4) at exit-handlers.c:36
36                      abort();
(gdb) li
31              if (argc == 2) {
32                      exit(EXIT_SUCCESS);
33              } else if (argc == 3) {
34                      _exit(EXIT_SUCCESS);
35              } else if (argc == 4) {
36                      abort();
37              }
38      }
39
40
(gdb) p argc
$1 = 4
(gdb) 
```

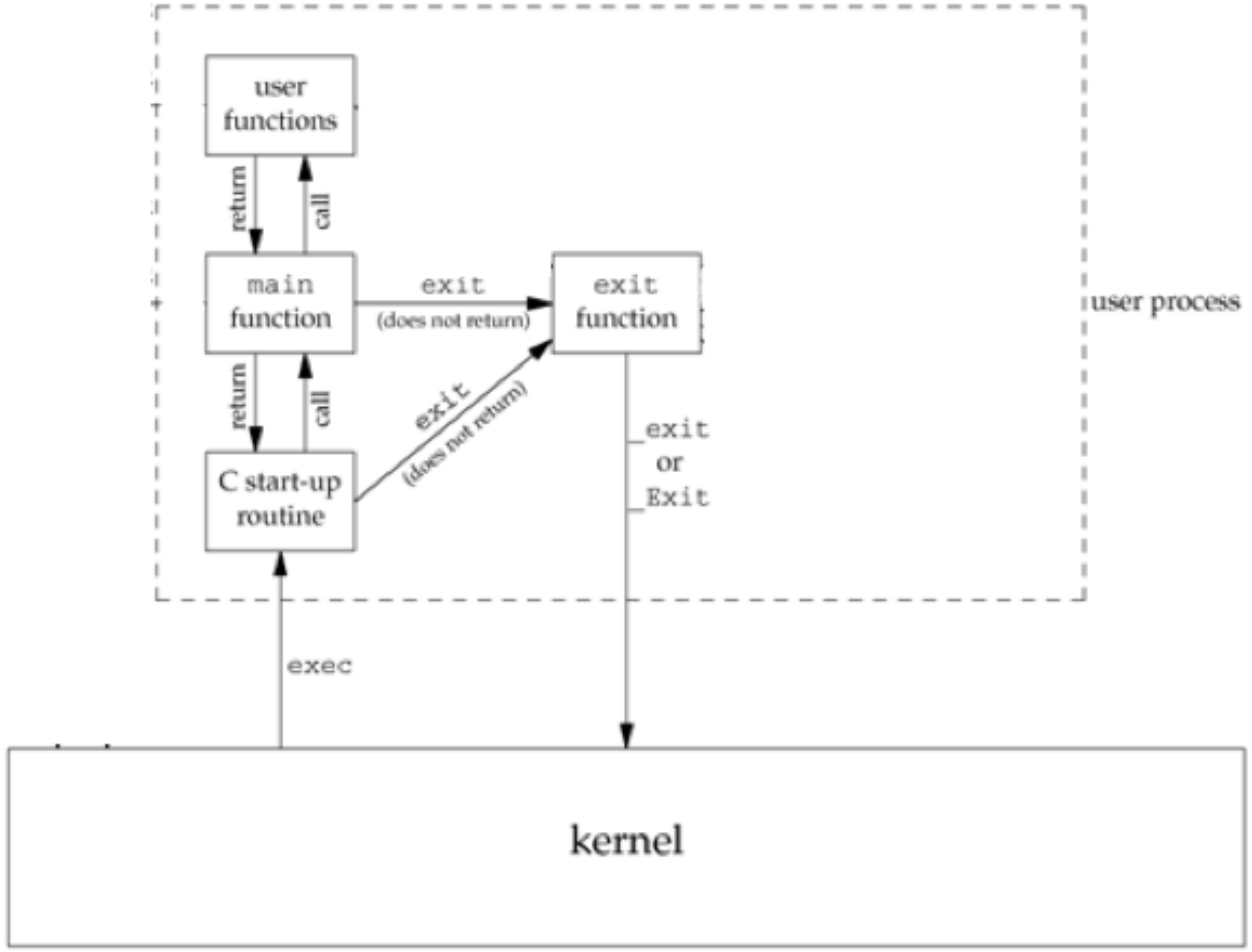
Lifetime of a Unix Process



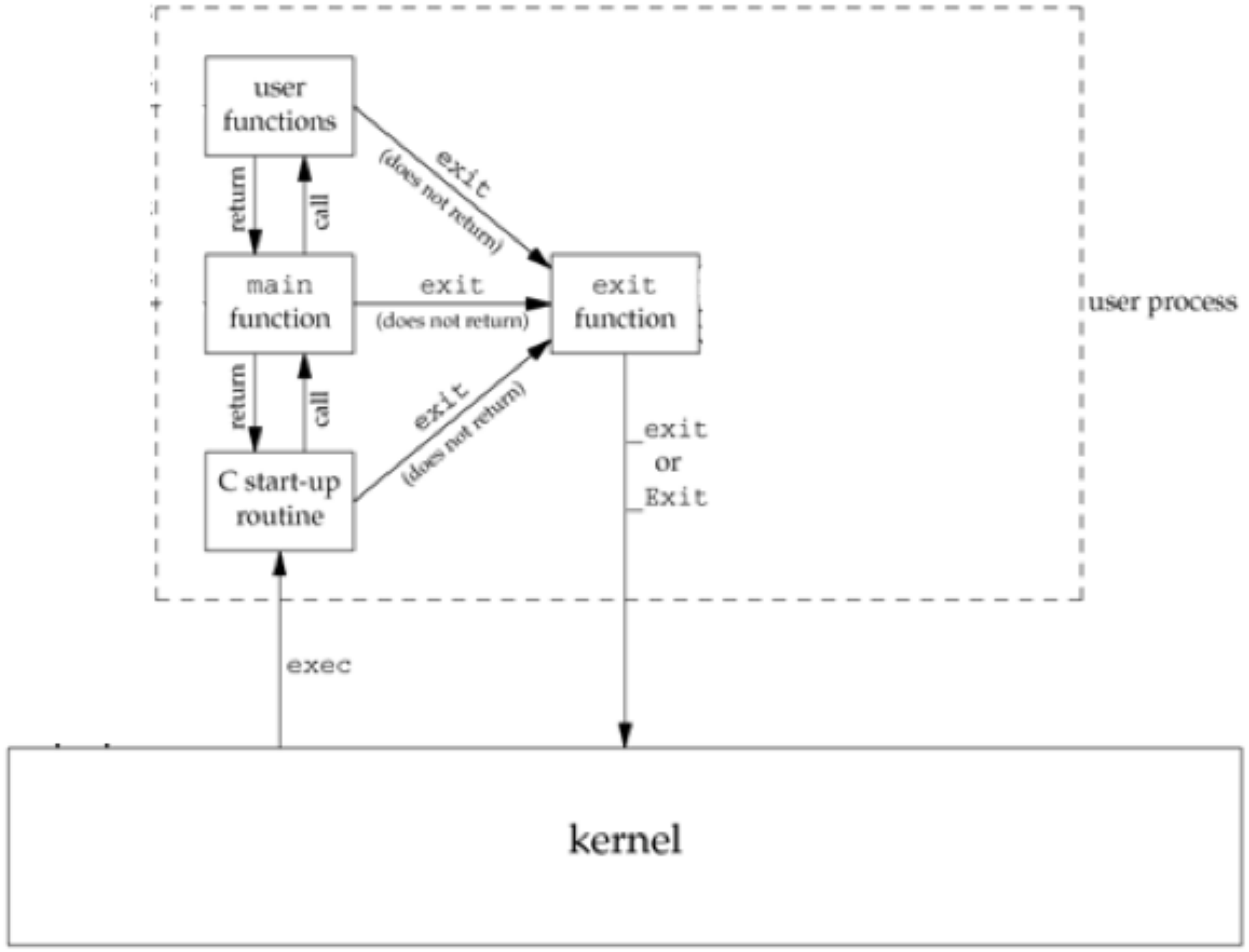
Lifetime of a Unix Process



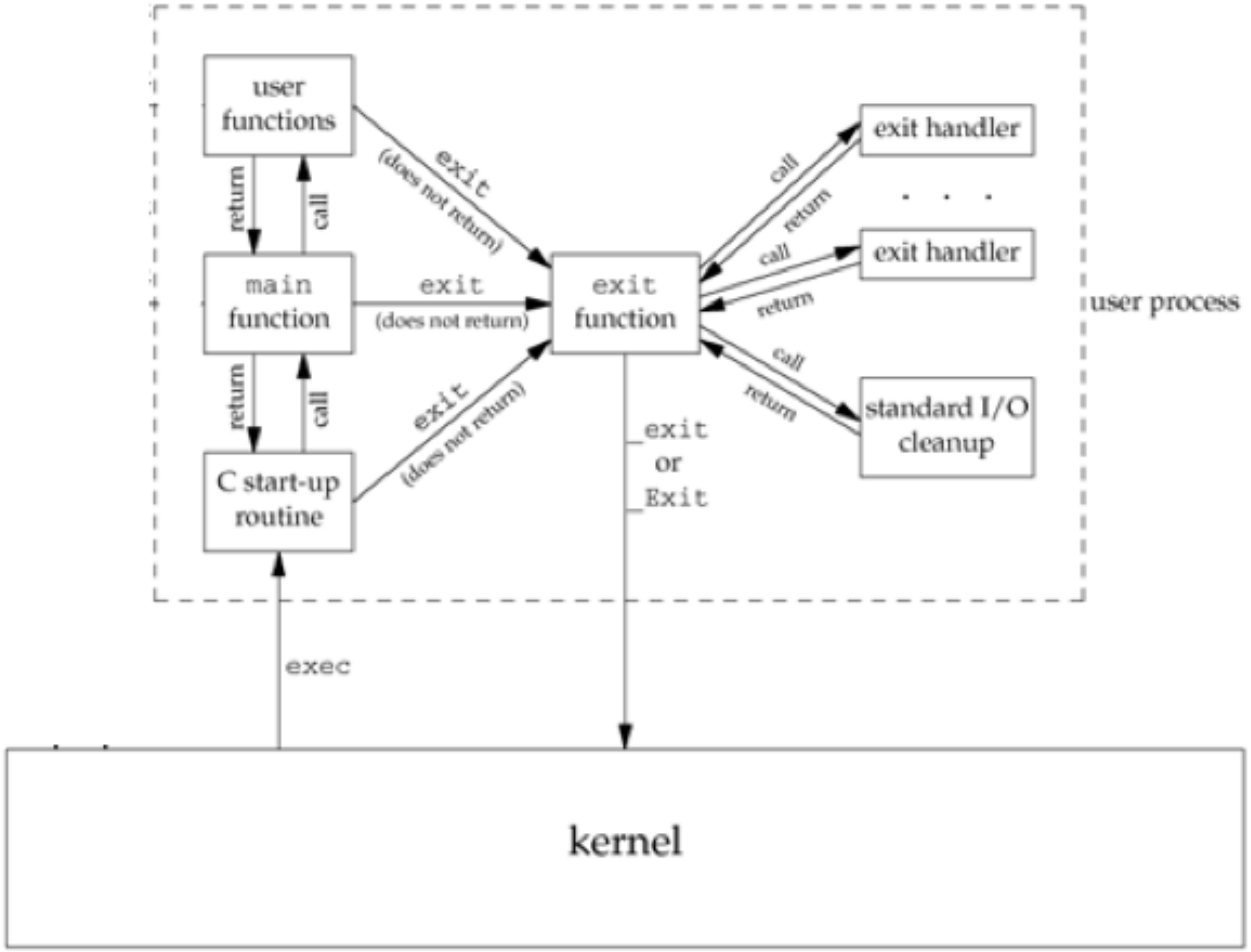
Lifetime of a Unix Process



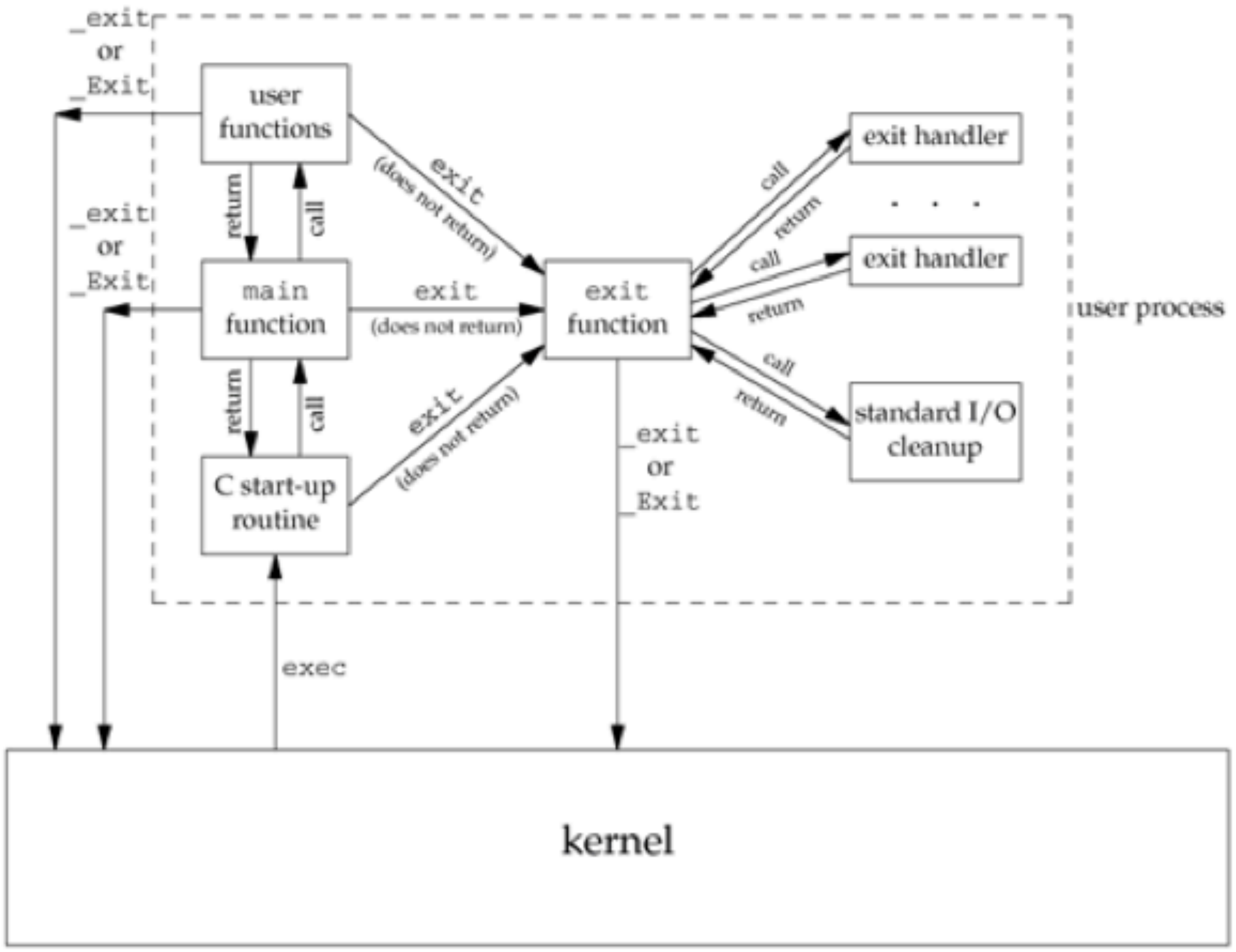
Lifetime of a Unix Process



Lifetime of a Unix Process



Lifetime of a Unix Process



Program Termination

- To implicitly exit(3), (implicitly or explicitly) return from main. Exit status depends on C standard and last function call.
- Explicitly exit(3) at any time.
- Register exit handlers via `atexit(3)`.
- Exit without calling exit handlers etc. via `_exit(2)` or `abort(3)`.

Impact of process termination on related processes will be covered in future classes.