

Processi

Franco Maria Nardini

La funzione `main`

```
int main(int argc, char **argv);
```

- il kernel gestisce l'avvio del vostro codice C (attraverso una funzione `exec`)
 - routine speciale di start-up che prepara l'ambiente di esecuzione per il `main` (o qualunque altro entry-point definito)
- `argc` è il counter degli argomenti
- `argv` è un array di puntatori agli argomenti
- ANSI C e POSIX.1 garantiscono che `argv[argc] == NULL`

Un esempio (linux)

```
$ cc -Wall entry.c
$ readelf -h a.out | more
ELF Header:
[...]
  Entry point address:                0x400460
  Start of program headers:          64 (bytes into file)
  Start of section headers:         4432 (bytes into file)
$ objdump -d a.out
[...]
0000000000400460 <_start>:
   400460:        31 ed                xor     %ebp,%ebp
   400462:        49 89 d1             mov    %rdx,%r9
[...]
$
```

Un esempio (linux)

glibc/sysdeps/x86_64/start.S

0000000000401058 <_start>:

```
401058:      31 ed                xor    %ebp,%ebp
40105a:      49 89 d1            mov    %rdx,%r9
40105d:      5e                pop    %rsi
40105e:      48 89 e2            mov    %rsp,%rdx
401061:      48 83 e4 f0        and    $0xfffffffffffffff0,%rsp
401065:      50                push   %rax
401066:      54                push   %rsp
401067:      49 c7 c0 e0 1a 40 00 mov    $0x401ae0,%r8
40106e:      48 c7 c1 50 1a 40 00 mov    $0x401a50,%rcx
401075:      48 c7 c7 91 11 40 00 mov    $0x401191,%rdi
40107c:      e8 2f 01 00 00     callq 4011b0 <__libc_start_main>
401081:      f4                hlt
401082:      90                nop
401083:      90                nop
```

Un esempio (linux)

glibc/csu/ld_start.c

```
STATIC int
LIBC_START_MAIN (int (*main) (int, char **, char ** MAIN_AUXVEC_DECL),
                 int argc, char **argv,
                 __typeof (main) init,
                 void (*fini) (void),
                 void (*rtld_fini) (void), void *stack_end)
{
  [...]
  result = main (argc, argv, __environ MAIN_AUXVEC_PARAM);

  exit (result);
}
```

Terminazione

- Otto modalità di terminazione.
- Terminazione “normale”
 - return da `main`
 - chiamata a `exit()`
 - chiamata a `_exit()` oppure `_Exit()`
 - ritorno dell'ultimo thread
 - chiamata a `pthread_exit()` dall'ultimo thread

Terminazione

- Otto modalità di terminazione.
- Terminazione “anormale”
 - chiamata di abort
 - terminazione da segnale
 - risposta dell’ultimo thread ad una richiesta di cancellazione

`exit(3)` e `_exit(2)`

- `_exit()` e `_Exit()`:
 - ritornano al kernel immediatamente
 - `_exit()` introdotta da POSIX.1
 - `_Exit()` introdotta da ISO C99
 - sinonimi sotto UNIX

`exit(3)` e `_exit(2)`

- `exit()` esegue un cleanup e ritorna
- entrambe hanno un intero come argomento: exit status!

Exit Status

- Exit status di un processo non è definito:
 - `*exit()` sono invocate senza uno status
 - `main` ritorna senza valore
 - `main` non dichiarata per ritornare un intero
- ISO C 99:
 - se `main` ritorna intero e no `return` esplicito: exit status è 0
- `exit(0) == return(0)` dalla funzione `main`
- `exit(main(argc, argv));`

Un esempio (linux)

```
$ cc -e foo entry.c
```

```
$ ./a.out
```

```
Foo for the win!
```

```
Memory fault
```

```
$ cc -e bar entry.c
```

```
$ ./a.out
```

```
bar rules!
```

```
$ echo $?
```

```
1
```

```
$ cc entry.c
```

```
$ ./a.out
```

```
Hooray main!
```

```
$ echo $?
```

```
13
```

```
$
```

Un esempio (linux)

```
$ cc -Wall hw.c
hw.c: In function 'main':
hw.c:7: warning: control reaches end of non-void function
$ ./a.out
Hello World!
$ echo $?
10
$ cc -Wall --std=c99 hw.c
$ ./a.out
Hello World!
$ echo $?
0
$
```

atexit (3)

```
#include <stdlib.h>

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

- Registra una funzione con una particolare dichiarazione (?) affinché sia chiamata all'uscita
- Le funzioni sono invocate in ordine inverso di registrazione
- La stessa funzione può essere registrata più volte
- Utile per gestire chiusura di file aperti, liberare risorse, ecc...

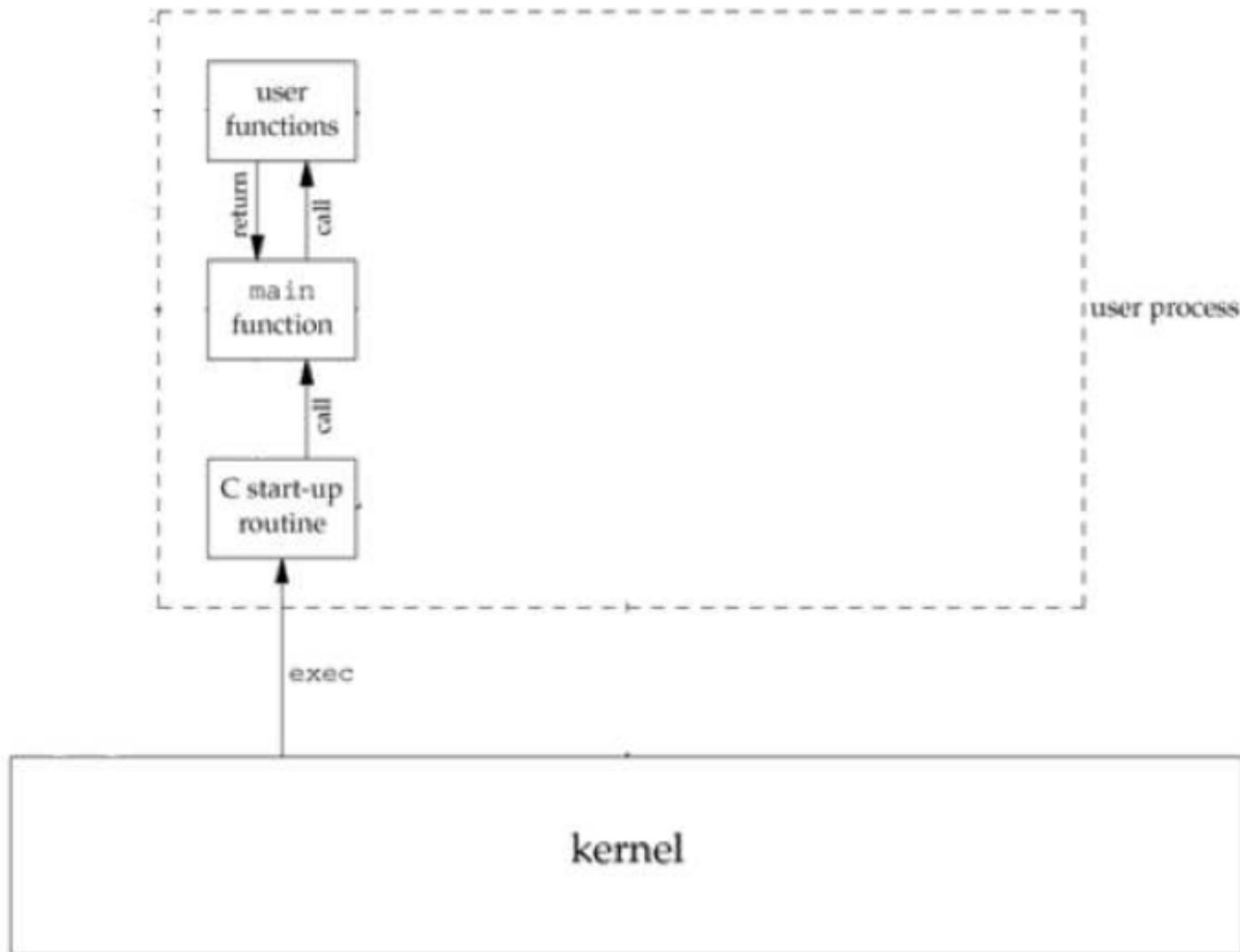
Esempio

```
$ wget http://hpc.isti.cnr.it/~nardini/siselab/07/exit-handlers.c
```

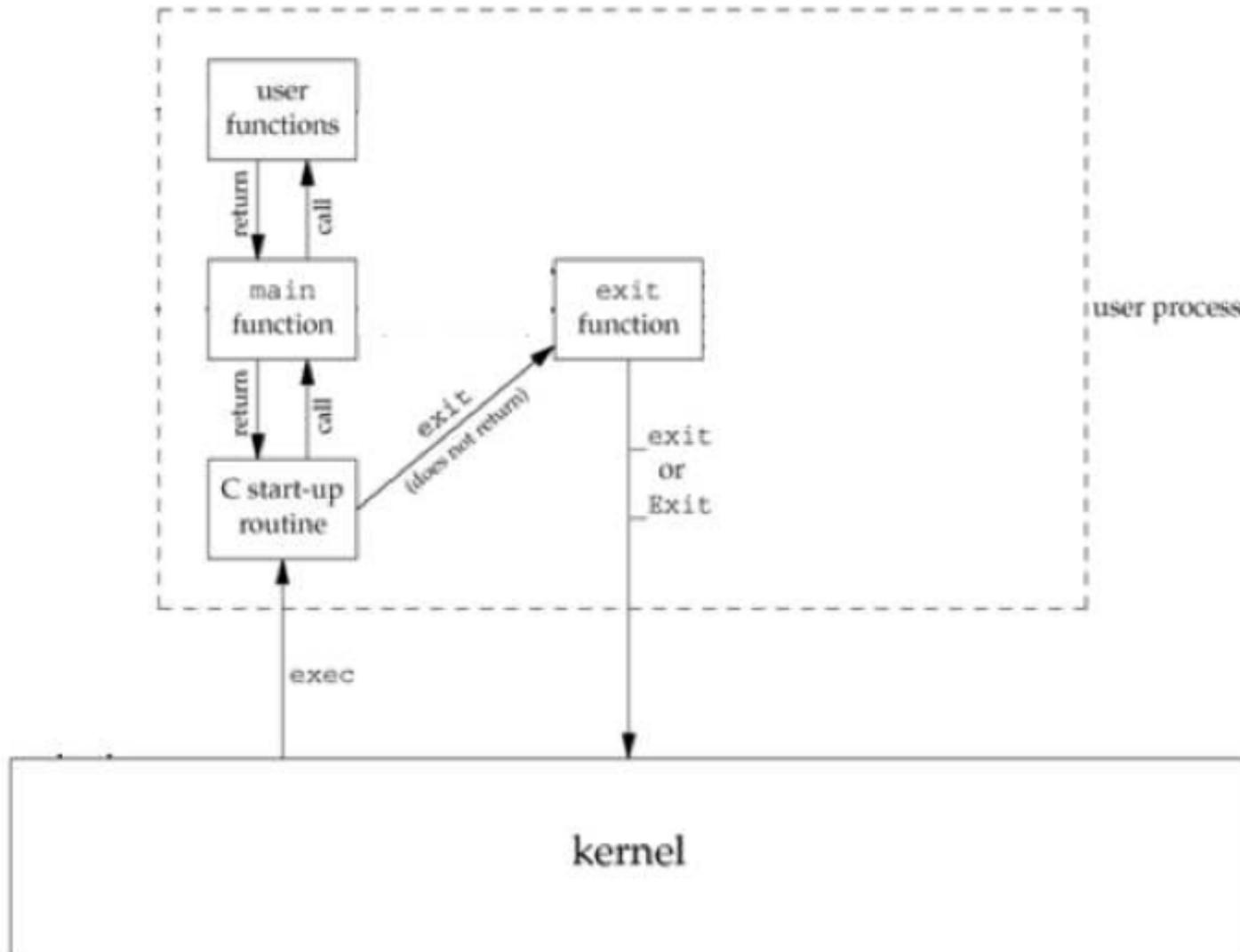
```
$ cc -Wall exit-handlers.c
```

```
$ ./a.out
```

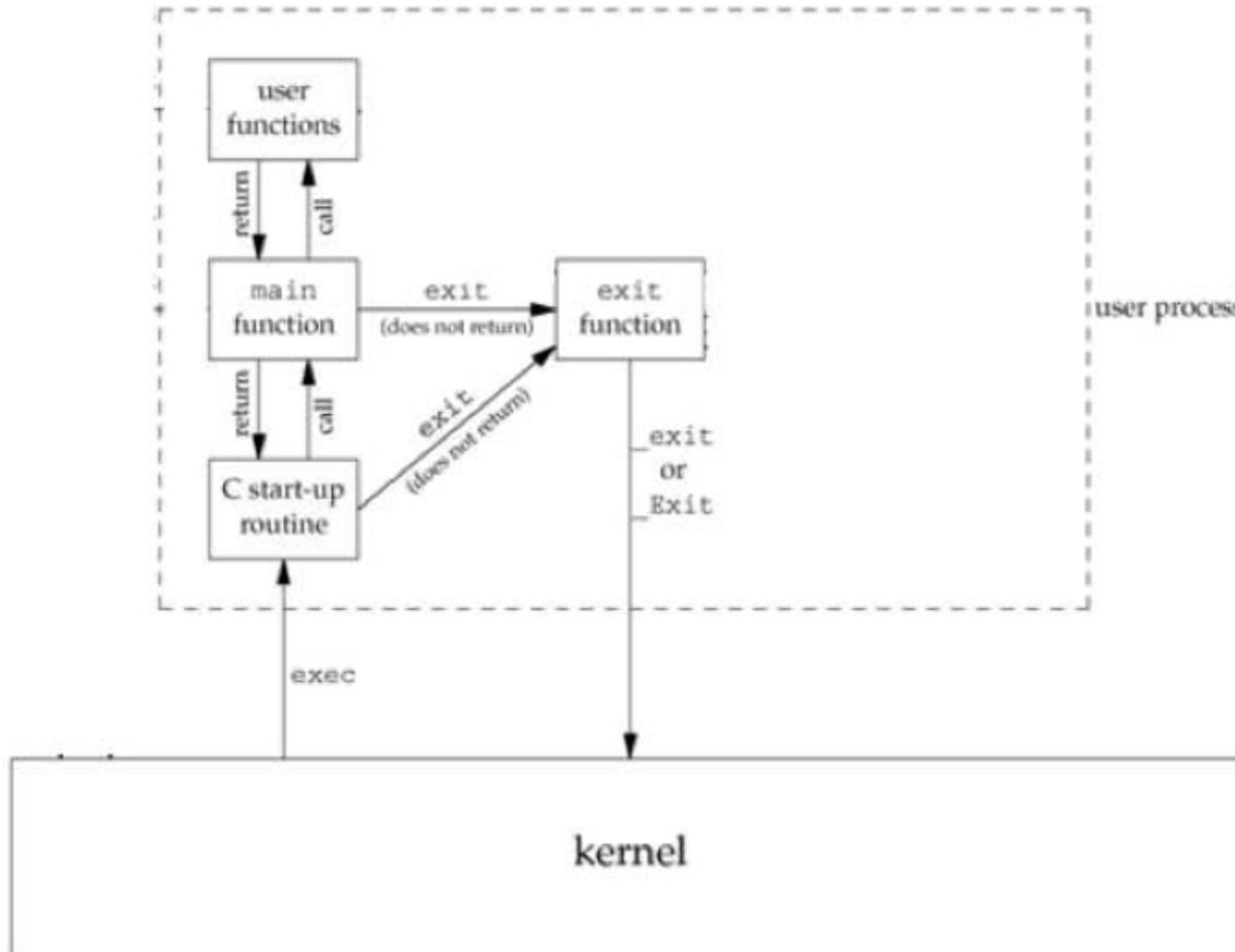
Lifetime di un processo



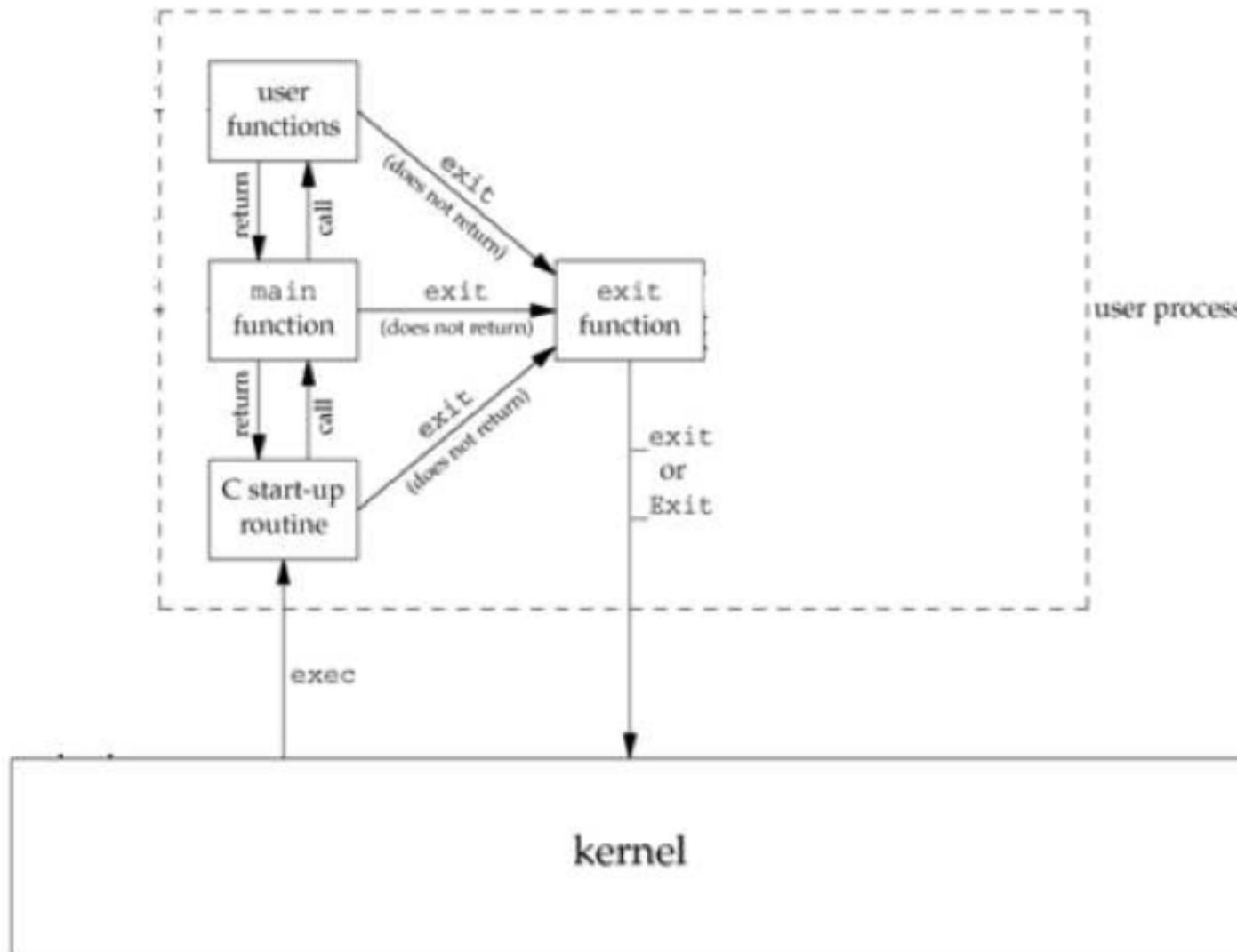
Lifetime di un processo



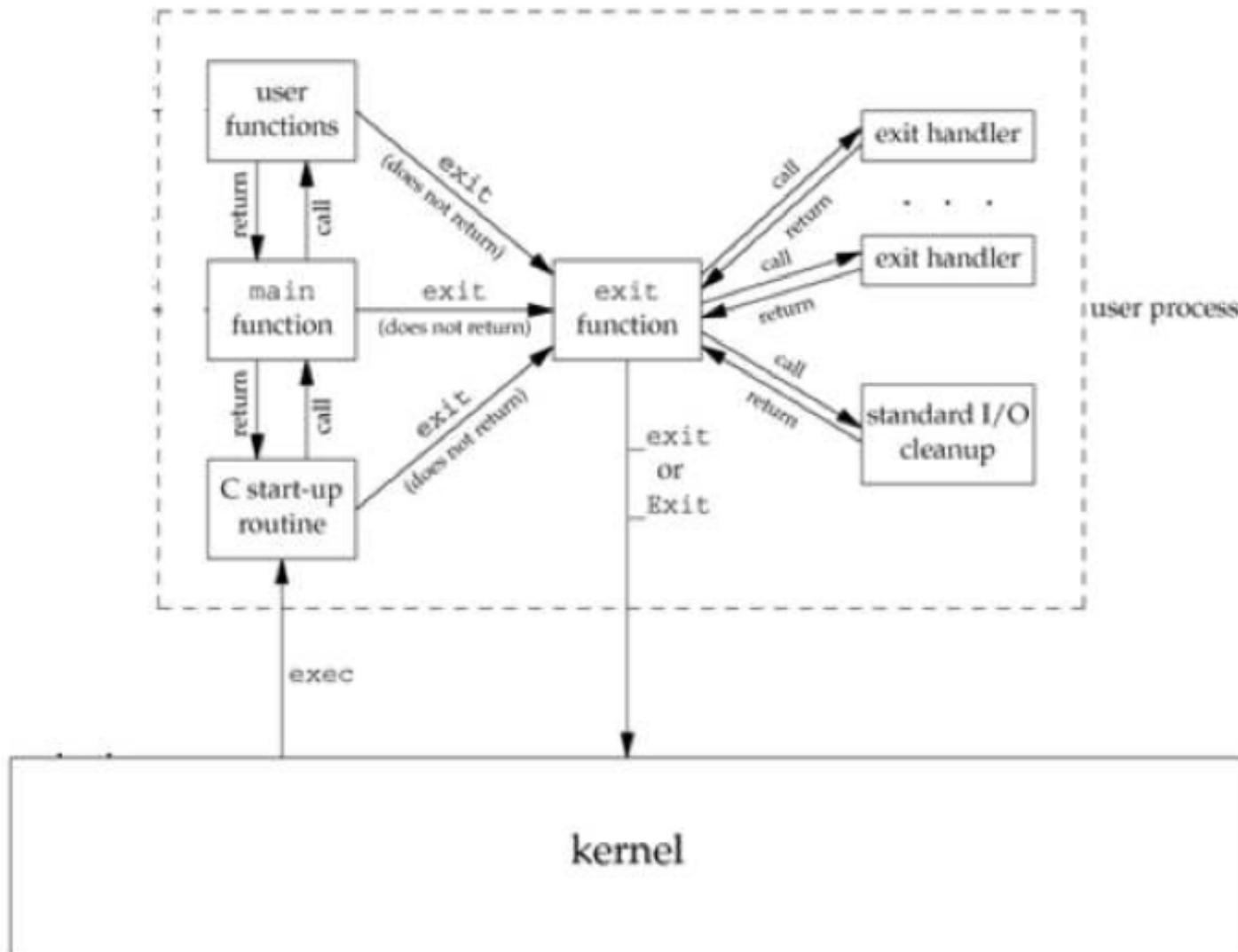
Lifetime di un processo



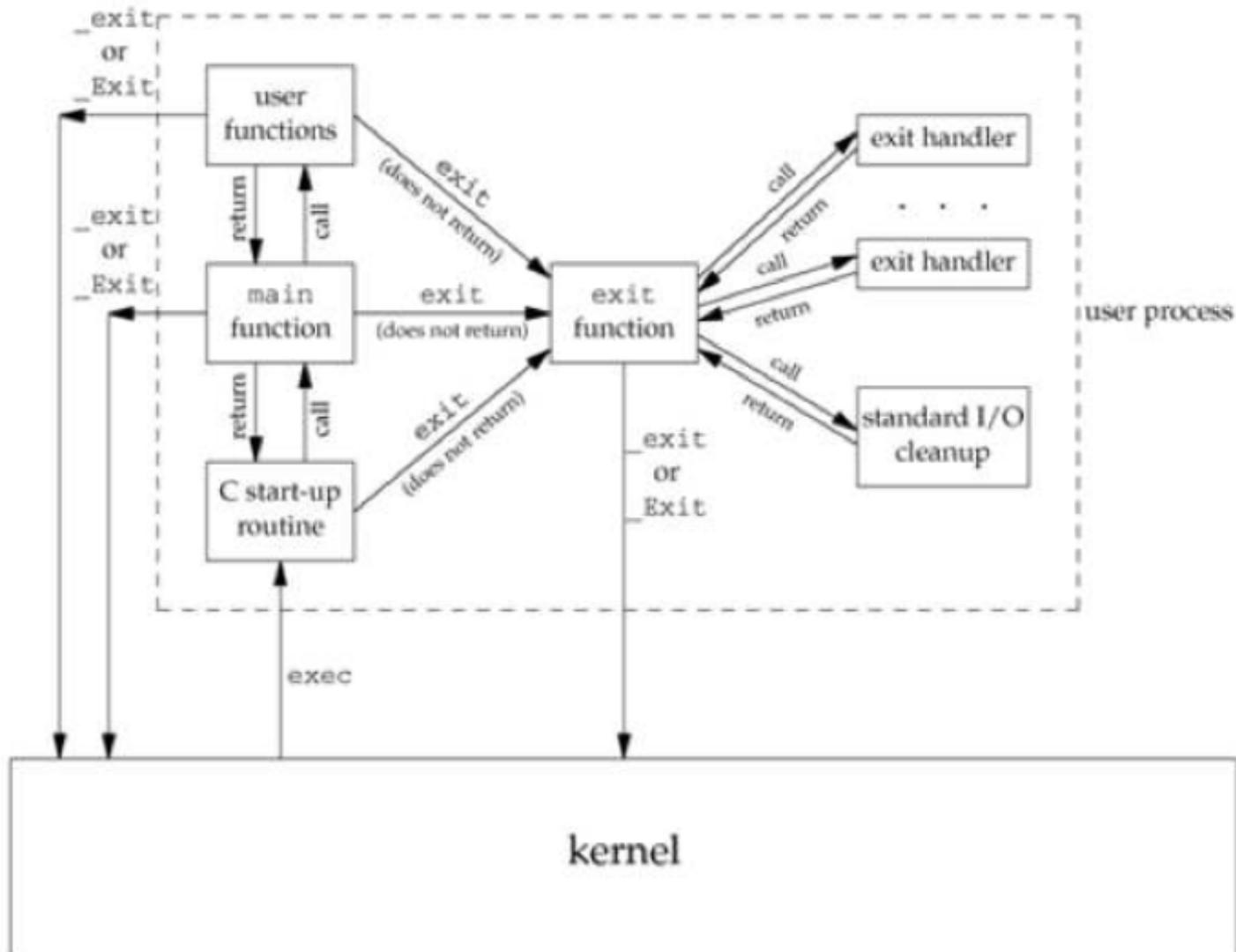
Lifetime di un processo



Lifetime di un processo



Lifetime di un processo



Variabili d'Ambiente

- della forma NOME=VALORE
- definiscono l'ambiente per le varie applicazioni
 - il kernel UNIX non le utilizza
- pesantemente usate dalla shell
 - più comuni: HOME e USER (settate a login)

```
nardini@tokaji:~/sisElab$ env
XDG_SESSION_ID=10
TAB=
TERM=xterm-256color
SHELL=/bin/bash
XDG_SESSION_COOKIE=a9ffb7f0c64da6287319876100000016-1428999348.549833-1245717052
SSH_CLIENT=146.48.82.2 51743 22
SSH_TTY=/dev/pts/2
LC_ALL=C
SVN_EDITOR=/usr/bin/vim
USER=nardini
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40
*.lz=01;31:*.xz=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.
1;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mr
1;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01
.axa=00;36:*.oga=00;36:*.spx=00;36:*.xspf=00;36:
MAIL=/var/mail/nardini
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
QT_QPA_PLATFORMTHEME=appmenu-qt5
PWD=/home/nardini/sisElab
JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64/
LANG=en_US.UTF-8
SHLVL=1
HOME=/home/nardini
LANGUAGE=en_US:en
LOGNAME=nardini
SSH_CONNECTION=146.48.82.2 51743 146.48.82.220 22
LESSOPEN=| /usr/bin/lesspipe %s
XDG_RUNTIME_DIR=/run/user/1000
LESSCLOSE=/usr/bin/lesspipe %s %s
_=/usr/bin/env
OLDPWD=/home/nardini
```

Variabili d'Ambiente

- le variabili d'ambiente sono memorizzate in un array globale di puntatori:

- `extern char **environ;`

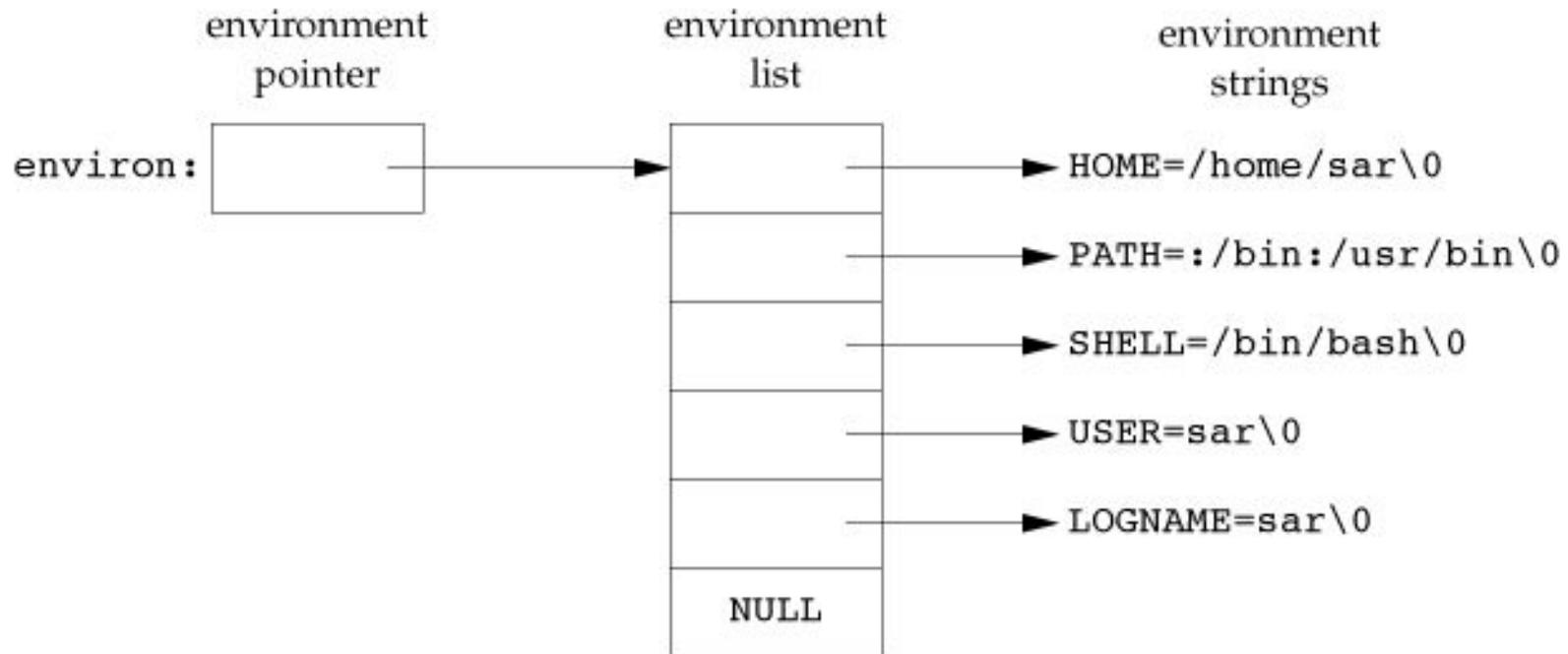
- La lista è `null` terminata.

#	Function	ISO C	POSIX.1	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
	<code>getenv</code>	•	•	•	•	•	•
c	<code>putenv</code>		XSI	•	•	•	•
i	<code>setenv</code>		•	•	•	•	
i	<code>unsetenv</code>		•	•	•	•	
v	<code>clearenv</code>				•		

`ite);`

```
int main(int argc, char **argv, char **envp);
```

Variabili d'Ambiente



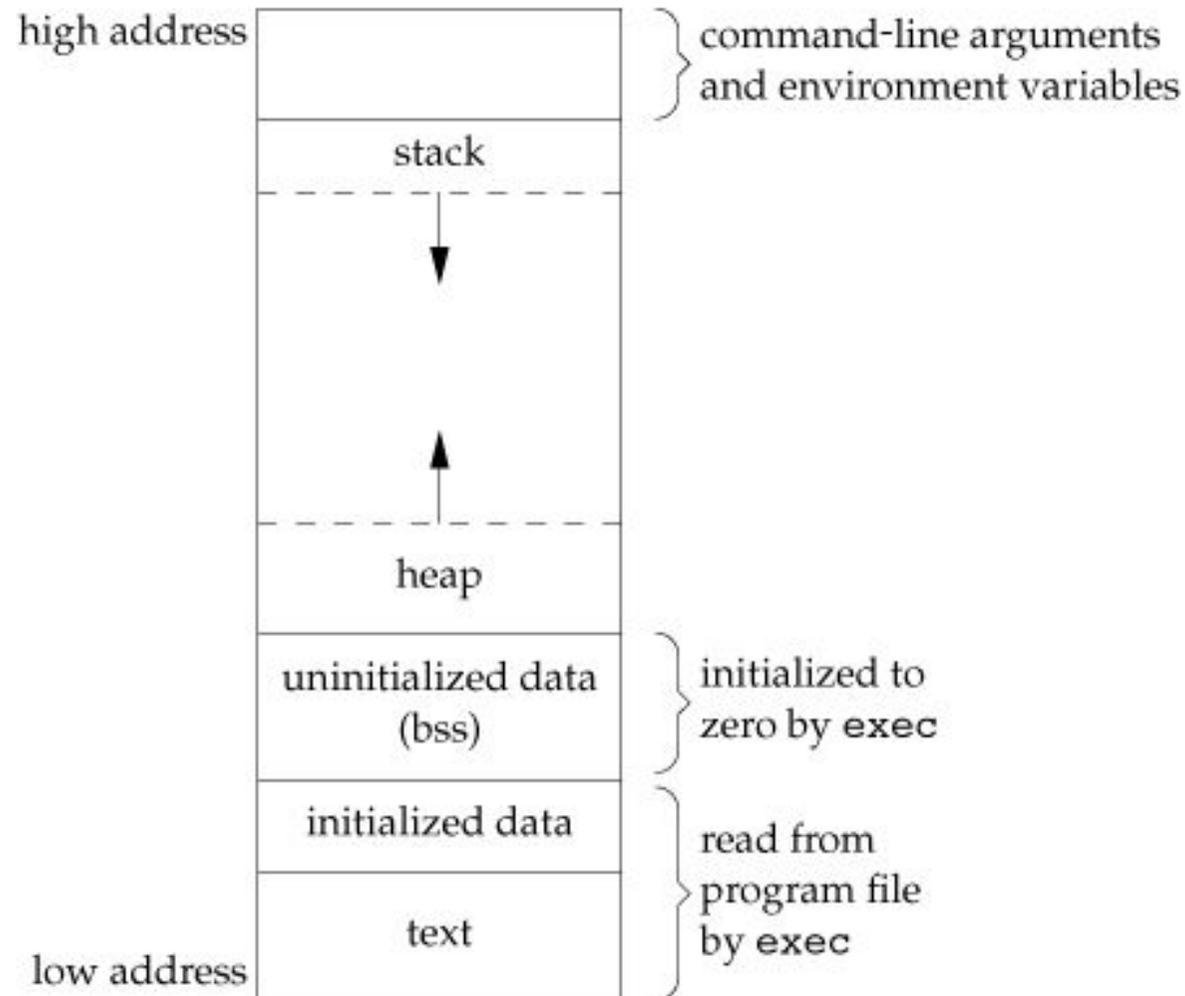
Memory Layout

- un programma C è composto dalle seguenti parti:
 - text segments: sono le istruzioni che la CPU esegue. è condiviso in memoria (una sola copia). è read-only.
 - (initialized) data segments: contiene variabili che sono specificamente inizializzate nel programma. La dichiarazione `int count = 0;` fuori da funzioni, causa la memorizzazione in questa sezione.

Memory Layout

- un programma C è composto dalle seguenti parti:
 - uninitialized data segments (bss): tutte le variabili inizializzate dal kernel a zero o NULL prima dell'esecuzione. `long sum[1000]` fuori da funzioni è memorizzato qui.
 - stack: dove la parte automatica è salvata insieme a tutte informazioni salvate ogni volta che una funzione è chiamata. ogni chiamata corrisponde ad un nuovo stack frame
 - heap: contiene l'allocazione dinamica.

Memory Layout



Memory Layout

```
$ cc hw.c
$ file a.out
a.out: ELF 32-bit LSB executable, Intel 80386, version 1 (SYSV),
dynamically linked (uses shared libs), for GNU/Linux 2.6.15, not stripped
$ ldd a.out
linux-gate.so.1 => (0x00c66000)
libc.so.6 => /lib/tls/i686/cmov/libc.so.6 (0x006b4000)
/lib/ld-linux.so.2 (0x005fe000)
$ size a.out
   text    data     bss     dec     hex filename
   918     264       8    1190    4a6 a.out
$ objdump -d a.out >obj
$ wc -l obj
225 obj
$
```

Homework

- esercizi assegnati: situazione?
- pagine man dei contenuti visti
- studiare Stevens, cap. 7