# Processi – Aspetti avanzati

Terminazione

Segnali

# Inizializzazione e terminazione di un programma

- **Dal punto di vista dell'utente, ogni programma "inizia" dalla funzione main()**
  - È la prima funzione utente ad essere eseguita
  - L'uscita da tale funzione coincide con la terminazione del programma
- **In realtà la situazione è più articolata**
  - Il kernel del sistema operativo chiama una funzione di "startup"
  - Tale funzione invoca la funzione main()
  - La funzione main() termina in modo "normale" o "abnormale"
- **Terminazione normale**
  - Istruzione return nel corpo della funzione main()
  - Chiamata della funzione exit()
  - Chiamata della funzione _exit()
- **Terminazione abnormale**
  - Chiamata della funzione abort()
  - Terminazione causata da un segnale

# Inizializzazione e terminazione di un programma

- **La funzione**

```
#include <unistd.h>
void _exit( int status );
```

  - Termina immediatamente l'esecuzione del programma
  - Ritorna il controllo al kernel

- **Mentre la funzione (già vista)**

```
#include <stdlib.h>
void exit( int status );
```

  - Esegue tutte le operazioni di housekeeping
    - Definite dall'utente
    - Definite dal sistema operativo
  - Invoca la funzione _exit() e quindi termina il processo

(c) 2013 - Carlo Brandolese, W.Fornaciari

# Inizializzazione e terminazione di un programma

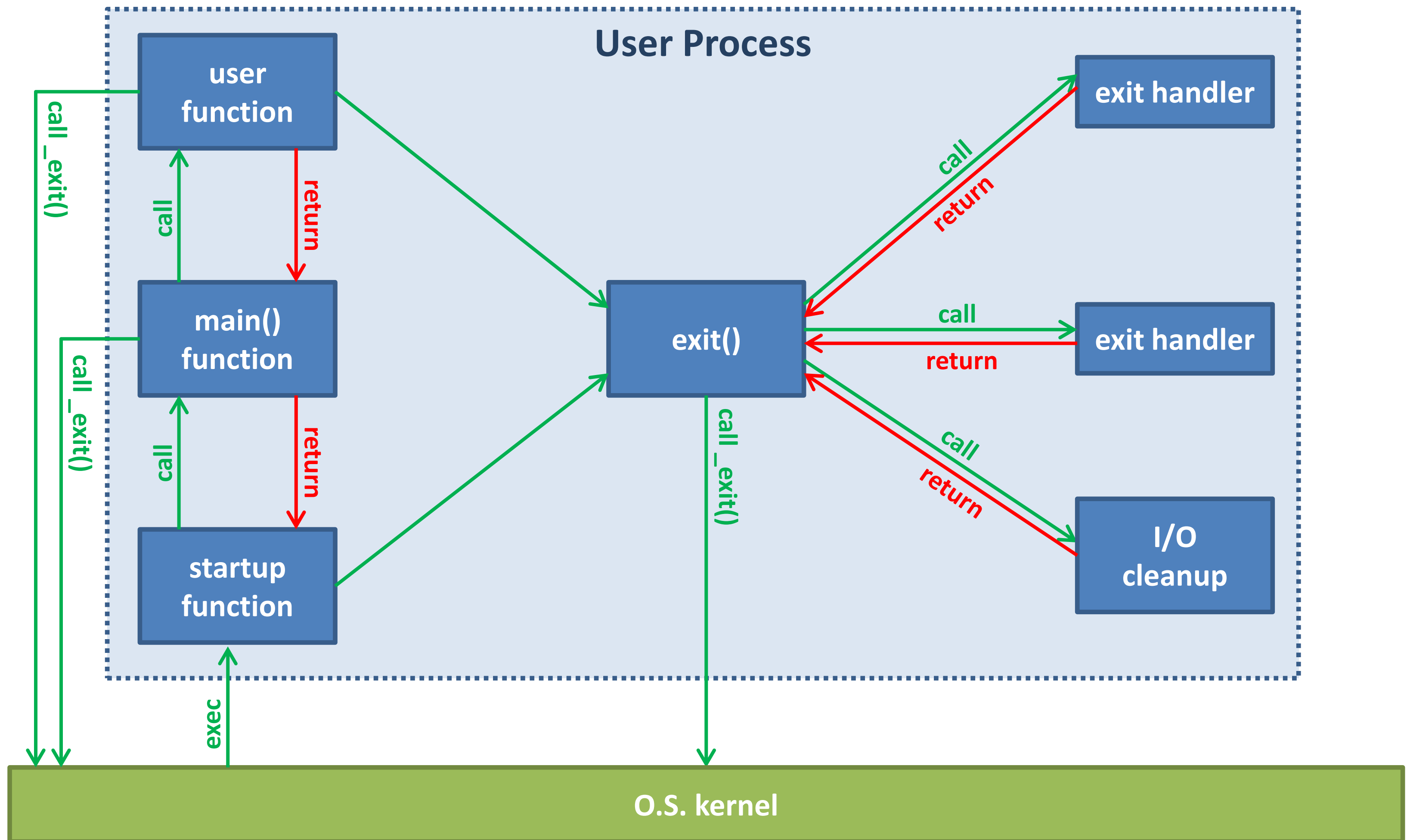
- **Le operazioni di housekeeping previste dal sistema operativo rigurdano sostanzialmente il rilascio delle risorse associate al processo**
  - Dispositivi di I/O, Memoria, ...
- **L'utente può specificare fino a 32 funzioni di housekeeping personalizzate**
  - Tali funzioni, dette "exit handler" devono essere "registrate"
  - Le funzioni saranno eseguite automaticamente dalla funzione exit()
    - Prima di invocare la funzione di terminazione _exit()
    - In ordine inverso rispetto a quello in cui sono state registrate
- **La funzione**

```
#include <stdlib.h>
int atexit( void (*f)(void) );
```

  - Registra la funzione void f(void) come exit handler
  - La funzione f() deve essere definita dall'utente
  - Una stessa funzione può essere registrata più volte

# Inizializzazione e terminazione di un programma

- Questo scehma mostra il ciclo di vita di un progrmma, dalla sua inizializzazione alla sua terminazione naturale




(c) 2013 - Carlo Brandolese, W.Fornaciari

# Inizializzazione e terminazione di un programma

- **La terminazione abnormale di un processo avviene sempre mediante un segnale**
  - Generato da un'altro processo
  - Generato dal processo stesso mediante la funzione kill()
    - Tratteremo questa funzione più avanti nell'ambito dei segnali
  - Generato dal processo stesso mediante la funzione abort()

- **La funzione**

```
#include <stdlib.h>
void abort( void );
```

  - Termina immediatamente il processo corrente
    - Gli eventuali exit handler registrati non vengono chiamati
    - I file aperti vengono chiusi
  - La funzione abort() invia al processo stesso il segnale SIGABRT
    - Il segnale causa la terminazione del processo

# Segnali

- **Un segnale è un "software interrupt"**
  - Utilizzato per gestire eventi asincroni
  - Fornisce una forma minima di sincronizzazione tra processi
- **I segnali sono identificati da numeri interi o nomi**
  - Da 1 a 15, oppure da 1 a 31 a seconda del sistema
  - Da nomi del tipo SIG<name>
  - Il valore 0 non corrispone ad alcun segnale
- **I segnali possono essere generati da**
  - Terminale
    - Mediante opportune combinazioni di tasti (CTRL-C, CTRL-Z, ...)
    - Mediante il comando kill
  - Eccezioni hardware
    - Intercettate dal kernel e notificate ai processi
  - Eventi/eccezioni software
    - Scrittura su una pipe chiusa, scadenza di un allarme
  - La funzione kill()

(c) 2013 - Carlo Brandolese, W.Fornaciari

# Segnali

| Signal Name | Number [1] | POSIX.1 1990 | POSIX.1-2001 | Other[2] | Description | Default Action |
|---|---|---|---|---|---|---|
| SIGHUP | 1 | • | • | • | Hangup | Terminate |
| SIGINT | 2 | • | • | • | Terminal interrupt character | Terminate |
| SIGQUIT | 3 | • | • | • | Terminal quit character | Terminate with core dump |
| SIGILL | 4 | • | • | • | Illegal instruction | Terminate with core dump |
| SIGTRAP | 5 | | • | • | Hardware fault | Terminate with core dump |
| SIGABRT | 6 | • | • | • | Abnormal termination | Terminate with core dump |
| SIGIOT | 6 | | | • | Hardware fault, I/O | Terminate with core dump |
| SIGBUS | 7 | | • | • | Hardware fault, Bus | Terminate with core dump |
| SIGFPE | 8 | • | • | • | Floating-point exception | Terminate with core dump |
| SIGKILL | 9 | • | • | • | Termination | Terminate |
| SIGUSR1 | 10 (30,16) | • | • | • | User-defined | Terminate |
| SIGSEGV | 11 | • | • | • | Invalid memory reference | Terminate with core dump |
| SIGUSR2 | 12 (31,17) | • | • | • | User-defined | Terminate |
| SIGPIPE | 13 | • | • | • | Broken pipe | Terminate |
| SIGALRM | 14 | • | • | • | Alarm clock | Terminate |
| SIGTERM | 15 | • | • | • | Termination | Terminate |
| SIGSTKFLT | 16 | | | • | | Stack fault |
| SIGCHLD | 17 (20,18) | • | • | • | Chile process status change | Ignore |
| SIGCONT | 18 (19,25) | • | • | • | Continue | Continue / Ignore |
| SIGSTOP | 19 (17,23) | • | • | • | Stop | Stop |
| SIGTSTP | 20 (18,24) | • | • | • | Terminal stop character | Stop |
| SIGTTIN | 21 (21,16) | • | • | • | Background read from terminal | Stop |
| SIGTTOU | 22 (22,27) | • | • | • | Background write to terminal | Stop |
| SIGURG | 23 (16,21) | | • | • | Urgent condition | Ignore |
| SIGXCPU | 24 (24,30) | | • | • | CPU limit exceeded | Terminate with core dump |
| SIGXFSZ | 25 (25,31) | | • | • | File size limit exceeded | Terminate with core dump |
| SIGVTALRM | 26 (26,28) | | • | • | Virtual timer alarm | Terminate |
| SIGPROF | 27 (27,29) | | • | • | Profiling timer alarm | Terminate |
| SIGWINCH | 28 | | | • | Terminal window size change | Ignore |
| SIGIO | 29 | | • | • | Asynchronous I/O | Terminate / Ignore |
| SIGPOLL | 29 | | • | • | Pollable event | Terminate |
| SIGPWR | 30 | | | • | Power failure restart | Ignore |
| SIGSYS | 31 (12,12) | | • | • | Bad system call | Terminate with core dump |

**(1)** La forma x (y,z) indica un valore dipendente dall'architettura, in cui x = (x86, ia64, ppc, arm), y = (alpha, sparc) e z = (mips).
**(2)** Altri stadard, tra cui SVR4, BSD, Linux, eccetera.

(c) 2013 - Carlo Brandolese, W.Fornaciari

# Segnali

- **La funzione**

```
#include <signal.h>
#include <sys/types.h>
int kill( pid_t pid, int signal );
```

- **Invia il segnale siganl specificato al processo pid**
  - Se pid è maggiore di zero, il segnale è inviato al processo indicato dal pid
  - Se pid è minore o uguale a zero, il comportamento della funzione kill() è più complesso ed esula da questa trattazione
  - Se il segnale signal è uguale a 0, la funzione kill () non invia alcun segnale
- **Questa funzione è definita dallo standard POSIX**
- **La funzione**

```
#include <signal.h>
int raise( int signal );
```

- **Invia il segnale specificato al processo stesso**
- **Questa funzione è definita dallo standard ANSI C**

(c) 2013 - Carlo Brandolese, W.Fornaciari

# Segnali

- **Quando un processo riceve un segnale può reagire in tre modi diversi**

- **Ignorare il segnale**
  - L'esecuzione del programma continua normalmente
  - I due segnali SIGKILL e SIGSTOP non possono essere mai ignorati

- **Gestire il segnale**
  - L'utente specifica una funzione ad hoc per la gestione del segnale
  - Tale funzione prende il nome di "signal handler"
  - Tale funzione deve essere registrata medinate la funzione signal()

- **Compiere l'azione di default**
  - Ad ogni segnale è associata un'azione di default
  - Per la maggioranza dei segnali l'azione di defualt consiste nel terminare il processo

# Segnali

- **La funzione**

```
#include <signal.h>
typedef void (*sighandler_t)(int);
sighandler_t signal( int signum, sighandler_t handler );
```

- **Regitra la funzione specificata come signal handler del segnale indicato**
  - La funzione restutisce il valore del precedente handler associato al segnale, oppure SIG_ERR in caso di errore.
- **Il parametro handler nella chiamata può essere**
  - La macro SIG_IGN
    - Indica di ignorare il segnale
  - La macro SIG_DFL
    - Indica di gestire il segnale mediante l'azione di default
  - Una funzione del tipo void handler(int)
    - Implementa una specifica procedura di gestione del segnale specificato
    - La funzione handler() viene invocata al momento della ricezione dello specifico segnale
    - Il parametro passato alla funzione handler() è il numero del segnale

# Segnali

- **La funzione**

```
#include <unistd.h>
unsigned int alarm( unsigned int secs );
```

- **Imposta un "allarme" o "software timer"**
  - L'allarme scadrà (almeno) dopo secs secondi
    - Se secs vale 0, nessun nuovo allarme è impostato e un eventuale allarme non ancora scaduto viene eliminato
  - Allo scaderere dell'allarme il sistema operativo invierà il segnale SIGALRM al processo che ha effettuato la chiamata
  - L'azione di default associata al seganle è la terminazione del processo
    - Normalmente viene installato uno specifico handler del segnale
- **La combinazione alarm/handler**
  - Permette di implementare un meccanismo di timeout
  - Permette di generare sequenze periodiche di eventi

# Segnali

- **La funzione**

```
#include <unistd.h>
int pause( void );
```

- **Sospende l'esecuzione fino a quando il processo non riceve un segnale, quindi**
  - Termina il processo se il segnale non è gestito, cioè se nessun handler è stato registrato
    - La funzione quindi non ritorna
  - Termina il processo se ciò è previsto dall'handler registrato
    - La funzione quindi non ritorna
  - Ritorna -1 se il segnale è gestito da un signal handler che prevede il ritorno
- **La combinazione alarm/handler/pause**
  - Permette di implementare un meccanismo attesa

# Examples - signals

- UNIX provides a mechanism for catching a signal (like an exception)

```
void (* signal( int sig, void (* handler)( int ) ) ( int );
```

- The function “handler()” will be invoked on a signal “sig”

handler( sig );

- Return the old signal handler on success and -1 on an error

- Examples

```
#include <signal.h>
...
signal( SIGINT, SIG_IGN ); /* ignore interrupts */
...
signal( SIGINT, SIG_DFL ); /* restore the default handler */
```

(c) 2013 - Carlo Brandolese, W.Fornaciari

# **Example -** Create a child process and kill it in 20 seconds

```
#include <signal.h>
int pid;
void OnAlarm( int sig )
{
kill( pid, SIGKILL );
}
...
if ( ( pid = fork() ) == 0 ) {
execlp( . . . );
}
signal( SIGALARM, OnAlarm ); /* setup the handler */
alarm( 20 ); /* fire off the alarm */
...
```

(c) 2013 - Carlo Brandolese, W.Fornaciari

# **Example -** Remove “tempfile” on an interrupt

```
#include <sys/signal.h>
char *tempfile = “temp.xxx”;
void cleanup( int sig )
{
unlink( tempfile );
exit( 1 );
};
void main( void )
{
int fd;
signal( SIGINT, cleanup ); /* set up cleanup() for
interrupt */
fd = open( tempfile, O_CREAT, 0666 );
... /* processing using
tempfile */
close( fd );
}
```

(c) 2013 - Carlo Brandolese, W.Fornaciari

# Example – signals and fork

- Signals are sent to all your processes
- You often want to disable signals and enable them later
- fork() may cause two processes to read your terminal (/dev/tty) at the same time

An example

```
#include <signal.h>
...
if ( fork() == 0 )
execlp( ... );
h = signal( SIGINT, SIG_IGN ); /*parent ignores interrupts*/
wait( &status ); /* until child is done */
signal( SIGINT, h ); /* restore interrupts */
```

(c) 2013 - Carlo Brandolese, W.Fornaciari

# Example - using a signal-handler function to count the number of times that the program receives SIGUSR1, one of the signals reserved for application use.

```
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <unistd.h>

sig_atomic_t sigusr1_count = 0;

void handler (int signal_number)
{
  ++sigusr1_count;
}

int main ()
{
  struct sigaction sa;
  memset (&sa, 0, sizeof (sa));
  sa.sa_handler = &handler;
  sigaction (SIGUSR1, &sa, NULL);

  /* Do some lengthy stuff here.  */
  /* ...  */

  printf ("SIGUSR1 was raised %d times\n", sigusr1_count);
  return 0;
}
```

(c) 2013 - Carlo Brandolese, W.Fornaciari