APPLICAZIONE UTENTE

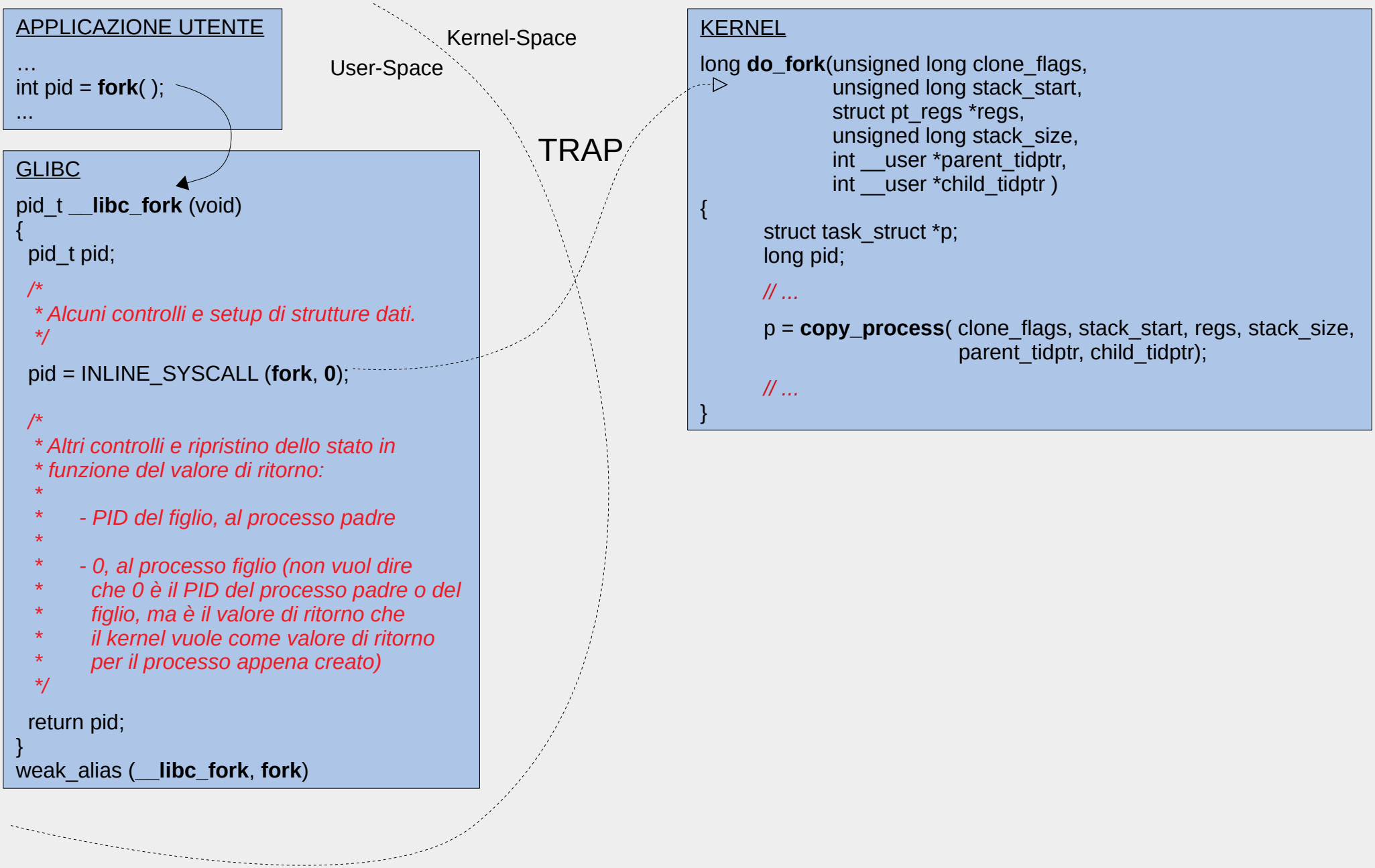
…
int pid = **fork**( );
...

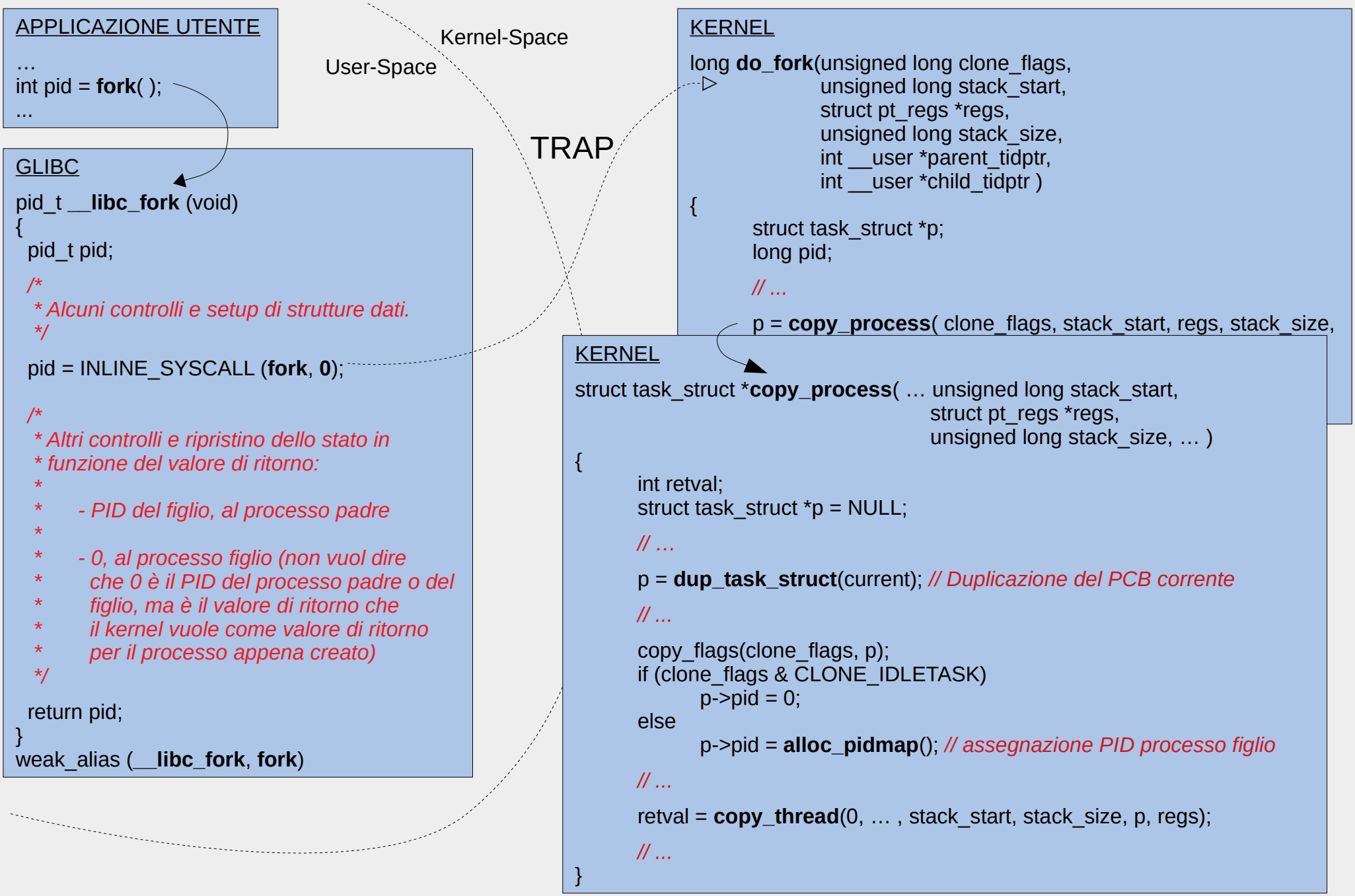
APPLICAZIONE UTENTE

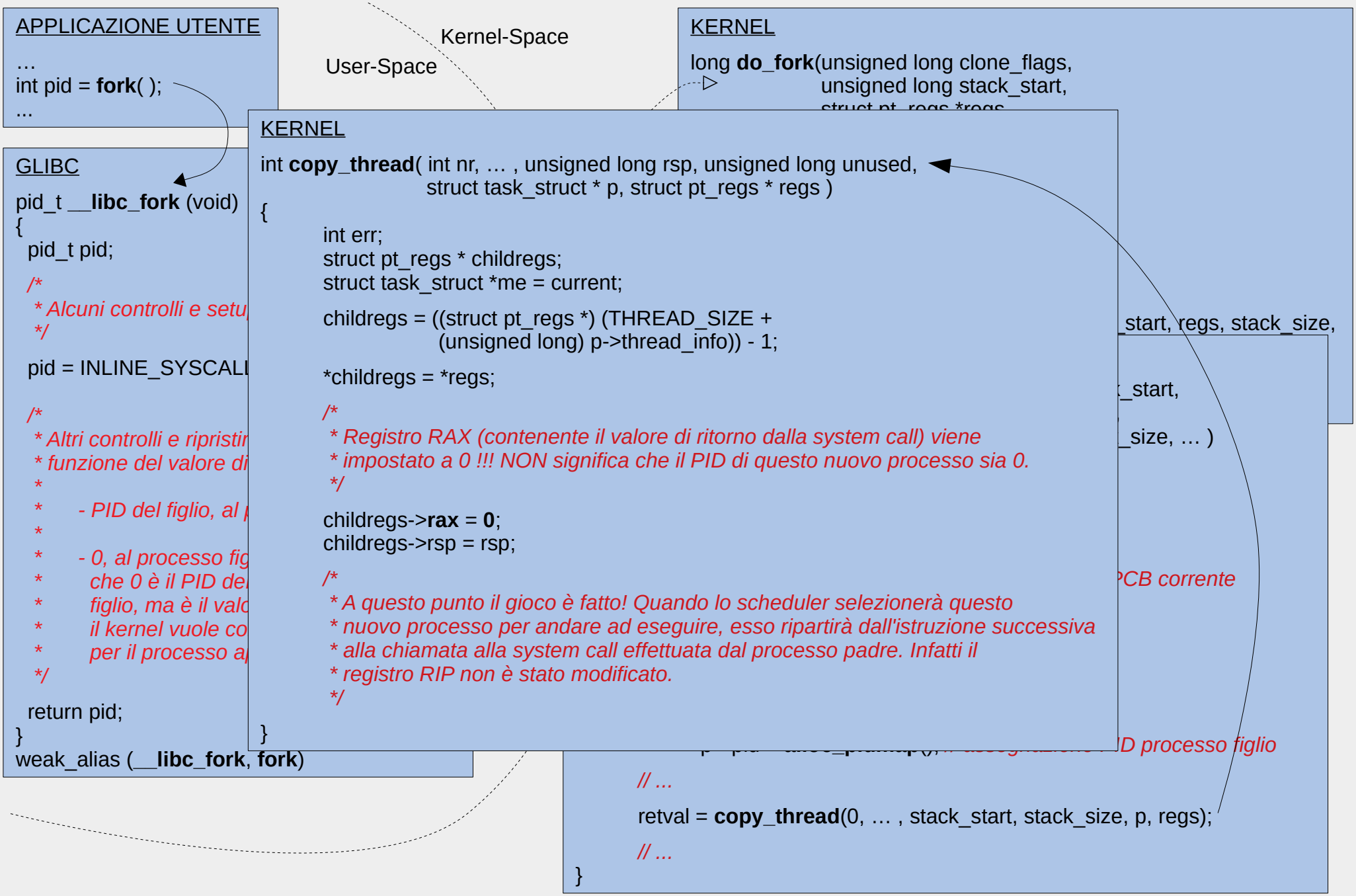
```
…
int pid = fork( );
...
```

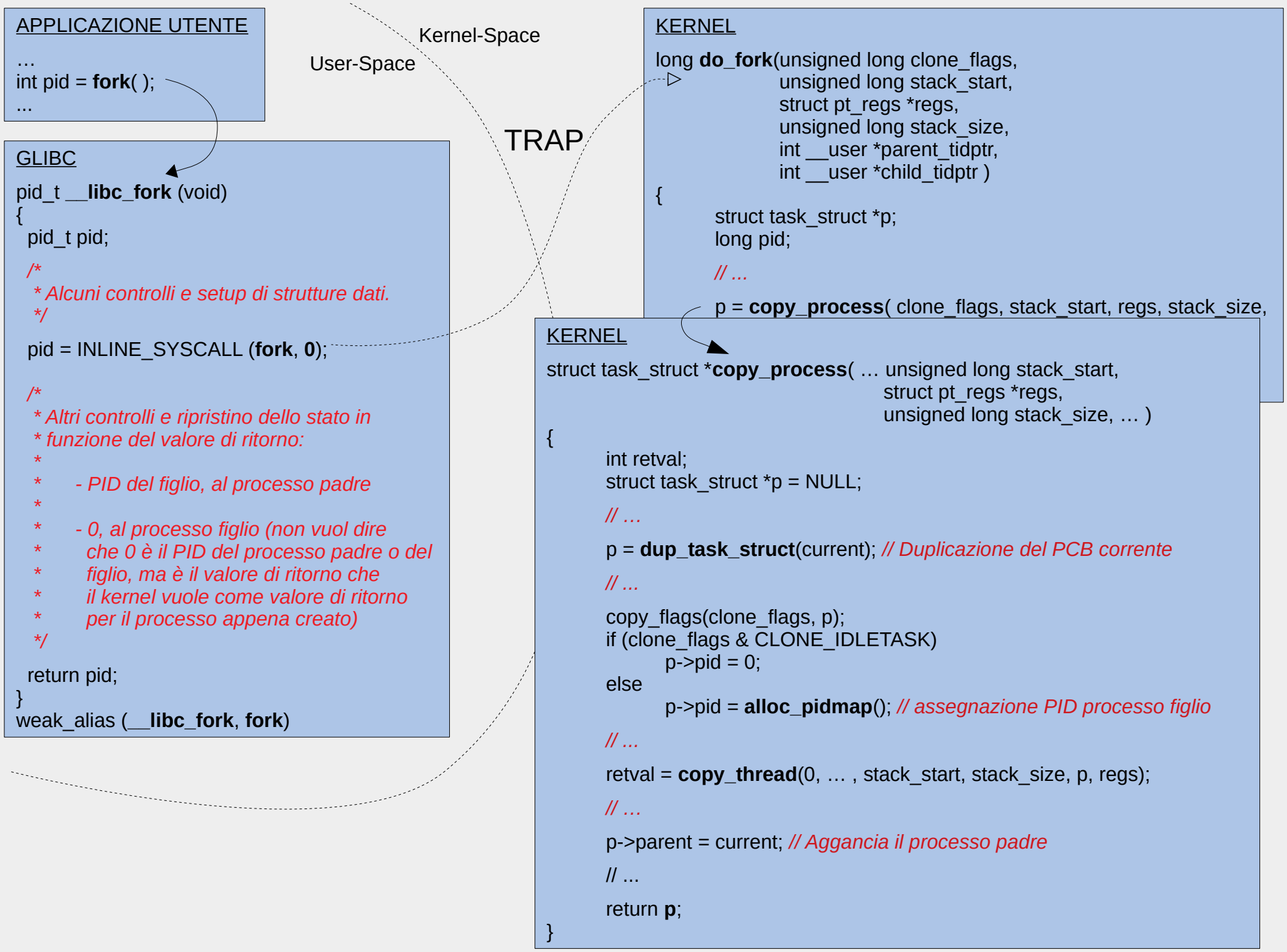
GLIBC

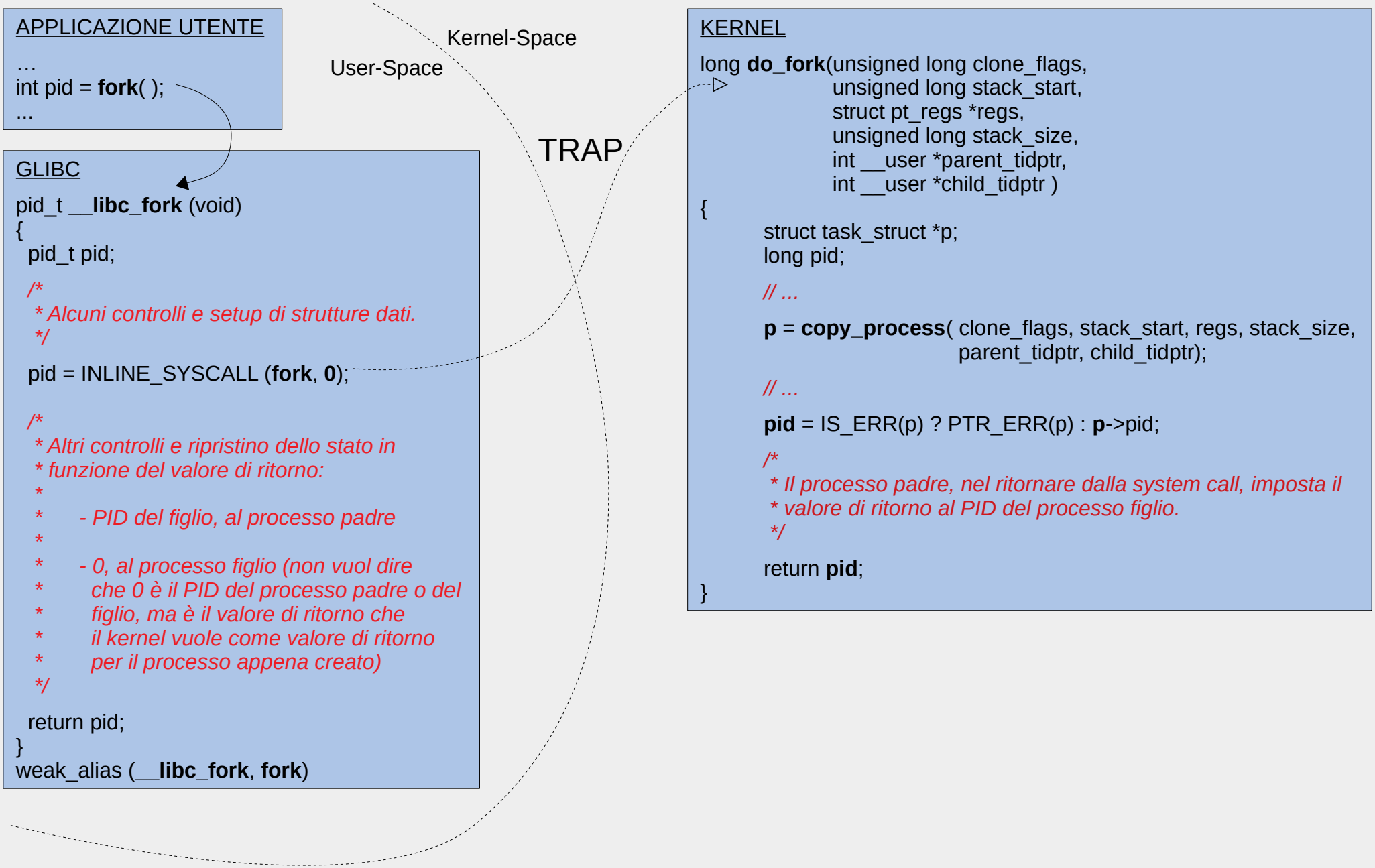
```
pid_t __libc_fork (void)
{
  pid_t pid;

  /*
   * Alcuni controlli e setup di strutture dati.
   */

  pid = INLINE_SYSCALL (fork, 0);

  /*
   * Altri controlli e ripristino dello stato in
   * funzione del valore di ritorno:
   *
   *      - PID del figlio, al processo padre
   *
   *      - 0, al processo figlio (non vuol dire
   *        che 0 è il PID del processo padre o del
   *        figlio, ma è il valore di ritorno che
   *        il kernel vuole come valore di ritorno
   *        per il processo appena creato)
   */

  return pid;
}
weak_alias (__libc_fork, fork)
```


APPLICAZIONE UTENTE
…
int pid = fork( );
...
GLIBC
pid_t __libc_fork (void)
{
pid_t pid;
/*
* Alcuni controlli e setup di strutture dati.
*/
pid = INLINE_SYSCALL (fork, 0);
/*
* Altri controlli e ripristino dello stato in
* funzione del valore di ritorno:
*
* - PID del figlio, al processo padre
*
* - 0, al processo figlio (non vuol dire
* che 0 è il PID del processo padre o del
* figlio, ma è il valore di ritorno che
* il kernel vuole come valore di ritorno
* per il processo appena creato)
*/
return pid;
}
weak_alias (__libc_fork, fork)
User-Space
Kernel-Space
TRAP
KERNEL
long do_fork(unsigned long clone_flags,
unsigned long stack_start,
struct pt_regs *regs,
unsigned long stack_size,
int __user *parent_tidptr,
int __user *child_tidptr)
{
struct task_struct *p;
long pid;
// ...
p = copy_process(clone_flags, stack_start, regs, stack_size,
parent_tidptr, child_tidptr);
// ...
}


APPLICAZIONE UTENTE
…
int pid = fork( );
...
User-Space
Kernel-Space
TRAP
GLIBC
pid_t __libc_fork (void)
{
pid_t pid;
/*
* Alcuni controlli e setup di strutture dati.
*/
pid = INLINE_SYSCALL (fork, 0);
/*
* Altri controlli e ripristino dello stato in
* funzione del valore di ritorno:
*
* - PID del figlio, al processo padre
*
* - 0, al processo figlio (non vuol dire
* che 0 è il PID del processo padre o del
* figlio, ma è il valore di ritorno che
* il kernel vuole come valore di ritorno
* per il processo appena creato)
*/
return pid;
}
weak_alias (__libc_fork, fork)
KERNEL
long do_fork(unsigned long clone_flags,
unsigned long stack_start,
struct pt_regs *regs,
unsigned long stack_size,
int __user *parent_tidptr,
int __user *child_tidptr)
{
struct task_struct *p;
long pid;
// ...
p = copy_process(clone_flags, stack_start, regs, stack_size,
KERNEL
struct task_struct *copy_process(… unsigned long stack_start,
struct pt_regs *regs,
unsigned long stack_size, …)
{
int retval;
struct task_struct *p = NULL;
// …
p = dup_task_struct(current); // Duplicazione del PCB corrente
// ...
copy_flags(clone_flags, p);
if (clone_flags & CLONE_IDLETASK)
p->pid = 0;
else
p->pid = alloc_pidmap(); // assegnazione PID processo figlio
// ...
retval = copy_thread(0, … , stack_start, stack_size, p, regs);
// ...
}


APPLICAZIONE UTENTE
…
int pid = fork( );
...
User-Space
Kernel-Space
GLIBC
pid_t __libc_fork (void)
{
pid_t pid;
/*
* Alcuni controlli e setu
*/
pid = INLINE_SYSCALL
/*
* Altri controlli e ripristin
* funzione del valore di
*
* - PID del figlio, al
*
* - 0, al processo fig
* che 0 è il PID del
* figlio, ma è il valo
* il kernel vuole co
* per il processo a
*/
return pid;
}
weak_alias (__libc_fork, fork)
KERNEL
int copy_thread(int nr, … , unsigned long rsp, unsigned long unused,
struct task_struct * p, struct pt_regs * regs)
{
int err;
struct pt_regs * childregs;
struct task_struct *me = current;
childregs = ((struct pt_regs *) (THREAD_SIZE +
(unsigned long) p->thread_info)) - 1;
*childregs = *regs;
/*
* Registro RAX (contenente il valore di ritorno dalla system call) viene
* impostato a 0 !!! NON significa che il PID di questo nuovo processo sia 0.
*/
childregs->rax = 0;
childregs->rsp = rsp;
/*
* A questo punto il gioco è fatto! Quando lo scheduler selezionerà questo
* nuovo processo per andare ad eseguire, esso ripartirà dall'istruzione successiva
* alla chiamata alla system call effettuata dal processo padre. Infatti il
* registro RIP non è stato modificato.
*/
}
KERNEL
long do_fork(unsigned long clone_flags,
unsigned long stack_start,
_start, regs, stack_size,
_start,
_size, …)
PCB corrente
D processo figlio
// ...
retval = copy_thread(0, … , stack_start, stack_size, p, regs);
// ...
}


APPLICAZIONE UTENTE
…
int pid = fork( );
...
User-Space
Kernel-Space
TRAP
GLIBC
pid_t __libc_fork (void)
{
pid_t pid;
/*
* Alcuni controlli e setup di strutture dati.
*/
pid = INLINE_SYSCALL (fork, 0);
/*
* Altri controlli e ripristino dello stato in
* funzione del valore di ritorno:
*
* - PID del figlio, al processo padre
*
* - 0, al processo figlio (non vuol dire
* che 0 è il PID del processo padre o del
* figlio, ma è il valore di ritorno che
* il kernel vuole come valore di ritorno
* per il processo appena creato)
*/
return pid;
}
weak_alias (__libc_fork, fork)
KERNEL
long do_fork(unsigned long clone_flags,
unsigned long stack_start,
struct pt_regs *regs,
unsigned long stack_size,
int __user *parent_tidptr,
int __user *child_tidptr)
{
struct task_struct *p;
long pid;
// ...
p = copy_process(clone_flags, stack_start, regs, stack_size,
KERNEL
struct task_struct *copy_process(… unsigned long stack_start,
struct pt_regs *regs,
unsigned long stack_size, …)
{
int retval;
struct task_struct *p = NULL;
// …
p = dup_task_struct(current); // Duplicazione del PCB corrente
// ...
copy_flags(clone_flags, p);
if (clone_flags & CLONE_IDLETASK)
p->pid = 0;
else
p->pid = alloc_pidmap(); // assegnazione PID processo figlio
// ...
retval = copy_thread(0, … , stack_start, stack_size, p, regs);
// …
p->parent = current; // Aggancia il processo padre
// ...
return p;
}


APPLICAZIONE UTENTE
…
int pid = fork( );
...
GLIBC
pid_t __libc_fork (void)
{
pid_t pid;
/*
* Alcuni controlli e setup di strutture dati.
*/
pid = INLINE_SYSCALL (fork, 0);
/*
* Altri controlli e ripristino dello stato in
* funzione del valore di ritorno:
*
* - PID del figlio, al processo padre
*
* - 0, al processo figlio (non vuol dire
* che 0 è il PID del processo padre o del
* figlio, ma è il valore di ritorno che
* il kernel vuole come valore di ritorno
* per il processo appena creato)
*/
return pid;
}
weak_alias (__libc_fork, fork)
User-Space
Kernel-Space
TRAP
KERNEL
long do_fork(unsigned long clone_flags,
unsigned long stack_start,
struct pt_regs *regs,
unsigned long stack_size,
int __user *parent_tidptr,
int __user *child_tidptr)
{
struct task_struct *p;
long pid;
// ...
p = copy_process(clone_flags, stack_start, regs, stack_size,
parent_tidptr, child_tidptr);
// ...
pid = IS_ERR(p) ? PTR_ERR(p) : p->pid;
/*
* Il processo padre, nel ritornare dalla system call, imposta il
* valore di ritorno al PID del processo figlio.
*/
return pid;
}