

System call Unix nel linguaggio C

Antonio Rocchia

16 aprile 2022

Indice

1	Le system call di Unix	1
2	Lista delle system call Unix	1
3	Wrapper delle system call nel linguaggio C	2
3.1	fork()	2
3.2	getpid(),getppid()	3

1 Le system call di Unix

Il kernel di Linux espone un API di funzioni che un utente o un processo può chiamare per richiedere che un servizio venga svolto dal sistema operativo a livello kernel.

Ogni linguaggio di programmazione fa uso di queste system call in tantissimi modi. Ad esempio per aprire e creare file, leggere da un file, inizializzare un processo e così via.

Nei linguaggi di programmazione moderni le system call sono esposte al programmatore tramite una serie di funzioni(wrapper) che ne semplificano l'uso ed impediscono che vengano commesse violazioni dell'API.

2 Lista delle system call Unix

-
- Controllo dei processi/thread:
 - Creazione processi
 - Terminazione processi
 - Sospensione processi
 - Attesa terminazione
 - Sostituzione di un processo

- Gestione del file system
- Gestione dei dispositivi
- Comunicazione

3 Wrapper delle system call nel linguaggio C

3.1 fork()

fork - Crea un processo figlio

Includere fork

```
1 #include <unistd.h>
2
3 int fork();
```

Descrizione: `fork()` genera un processo figlio duplicando il processo che chiama questa funzione. Il nuovo processo è chiamato *figlio*, il processo chiamante è chiamato *padre*

Il processo figlio ed il padre *continuano l'esecuzione in spazi di memoria diversi*. Al momento della `fork()` *il figlio ottiene una copia dello spazio di memoria del padre*. Il padre ed il figlio sono identici eccetto per queste differenze:

- Il figlio ha il suo PID unico e diverso dal padre
- il figlio non eredita allarmi (*alarm()*) dal padre
- Il segnale di terminazione del figlio è sempre *SIGCHLD*
- Il figlio eredita le copie dei descrittori (*file descriptor*) aperti dal padre
- Il figlio eredita una copia dello stream di direttori aperti dal padre (*open-dir()*)

Valore di ritorno: In caso di successo, il processo padre riceve il PID del processo figlio appena generato, e il figlio riceve 0. In caso di fallimento, il padre riceve -1, non viene generato nessun processo figlio e *errno* viene impostata correttamente.

Esempi: Di seguito vengono riportati alcuni esempi di codice che mostrano gli utilizzi tipici di `fork()`. Si cerca dove possibile di spiegare al meglio ciò che succede durante l'esecuzione

Esempio minimale di chiamata a `fork()`

```
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main(void) {
5     // Inizializzo e controllo il valore di ritorno di fork()
6     int pid = 0;
7     pid = fork();
```

Il processo figlio ottiene una copia dello spazio di memoria del padre, in questo caso la variabile `pid` inizializzata a 0. Il processo padre ed il processo figlio proseguono l'esecuzione allo stesso momento (in parallelo), ma con una differenza, *fork restituisce il PID del processo figlio generato al padre e restituisce 0 al figlio*. Questo valore viene assegnato alla variabile `pid`. Ora possiamo scrivere del codice che permetta al processo di capire se è il padre o il figlio.

```
1  if(pid==0) { // Sono il figlio
2      // Codice del figlio
3      printf("Sono il padre");
4      return 0;
5  } else if(pid > 0) { // Sono il padre
6      // Codice del padre
7      printf("Sono il padre");
8  }
9  return 0;
10 }
```

Il processo figlio esegue lo stesso codice del processo padre. Pertanto nella maggior parte dei casi è necessario terminare l'esecuzione del figlio prima di uscire dal blocco *if(pid==0)*

3.2 getpid(),getppid()

`getpid`, `getppid` - restituisce l'identificatore del processo(PID)

Includere `getpid()`

```
1  #include <unistd.h>
2
3  int getpid();
4  int getppid();
```

Descrizione: *getpid()* restituisce al processo che chiama la funzione il suo identificatore di processo (*PID*).

getppid() restituisce al processo che chiama la funzione l'identificativo di processo (*PID*) del suo processo padre.

Controllo degli errori Questa system call non fallisce mai. Non è necessario alcun controllo.

Esempi: Di seguito alcuni esempi di codice che mostrano l'utilizzo di `getpid()` e `getppid()`.

`getpid()` in azione

```
1  #include <unistd.h>
2
3  int main(void) {
4      int pid = 0;
5  }
```

```
6   pid = fork();
7   if(pid==0) {
8       // processo figlio
9       printf("La variabile pid = %d\n", pid);
10      printf("Il mio pid: %d\n", getpid());
11      printf("Il pid del mio processo padre: %d\n",
12             getppid());
13      return 0;
14  } else if(pid < 0) {
15      perror("Errore durante la fork");
16      return -1;
17  } else { // if(pid>0)
18      // processo padre
19      printf("La variabile pid = %d\n", pid);
20      printf("Il mio pid: %d\n", getpid());
21      printf("Il pid del mio processo padre: %d\n",
22             getppid());
23      return 0;
24  }
25 }
```

Scrivo altro