

SUPSI

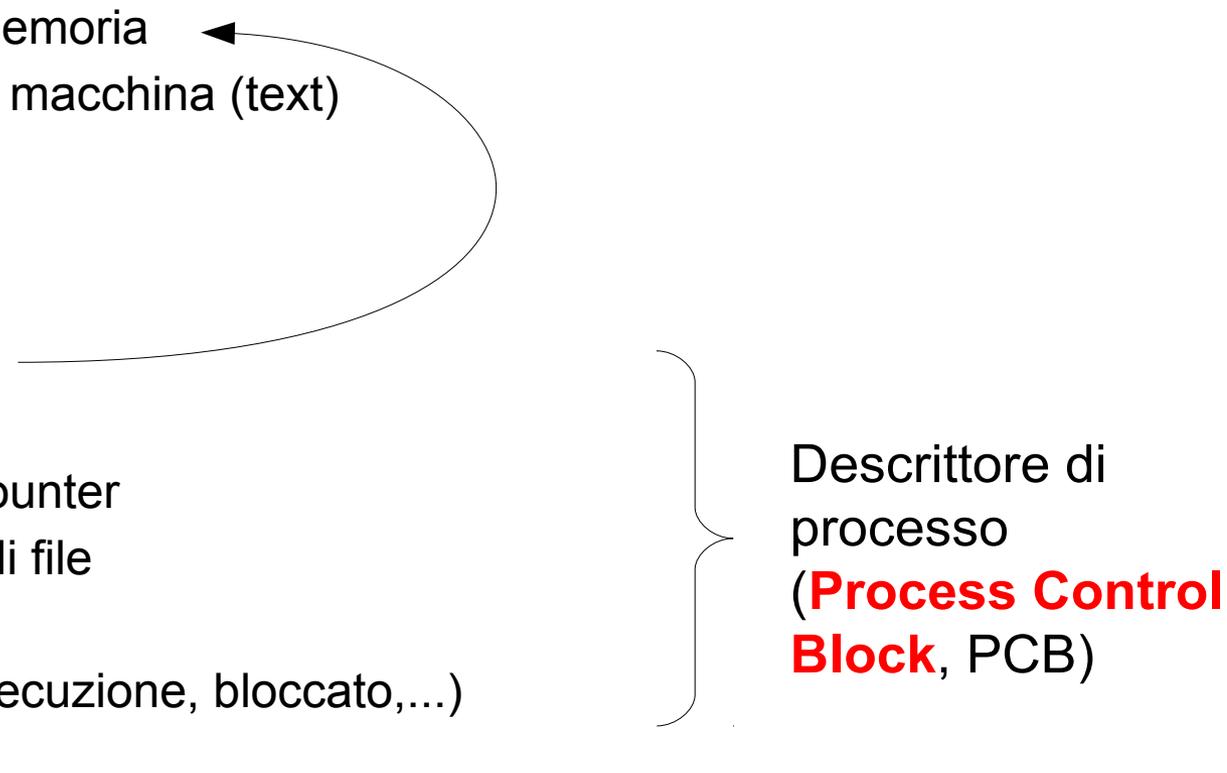
Programmazione multiprocesso

Amos Brocco, Ricercatore, DTI / ISIN

Processi

- Cos'è un **processo**?

- Un programma in esecuzione

- Spazio di memoria
 - Codice macchina (text)
 - Dati
 - Stack
 - Heap
 - Puntatori a
 - Registri
 - Program Counter
 - Descrittori di file
 - PID
 - Stato (in esecuzione, bloccato,...)
 - ...
- 

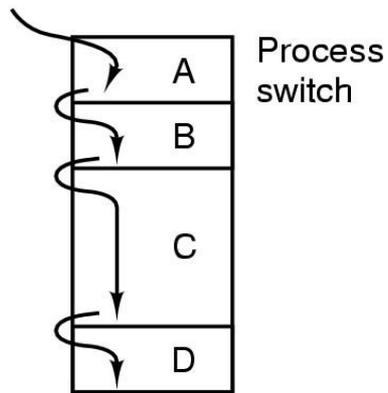
Descrittore di
processo
(**Process Control
Block**, PCB)

Sistemi multiprogrammati

- Sistemi **multiprogrammati**
 - **più processi contemporaneamente in memoria**
 - Il kernel mantiene una tabella dei processi (*process table*) con le informazioni necessarie all'esecuzione di ogni processo (*process control block*)
 - Esecuzione:
 - Pseudo parallelismo (time sharing, CPU singola)
 - Vero parallelismo (multi-core, multi-processore)

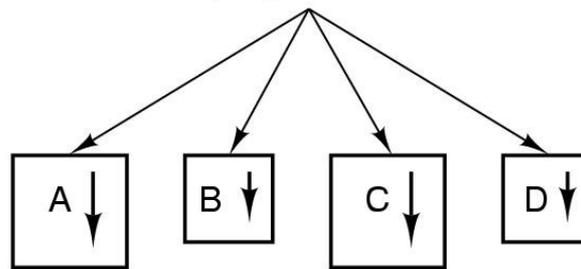
Sistemi multiprogrammati

One program counter

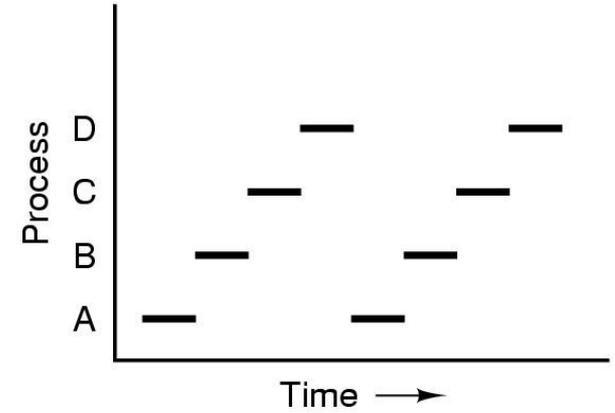


(a)

Four program counters

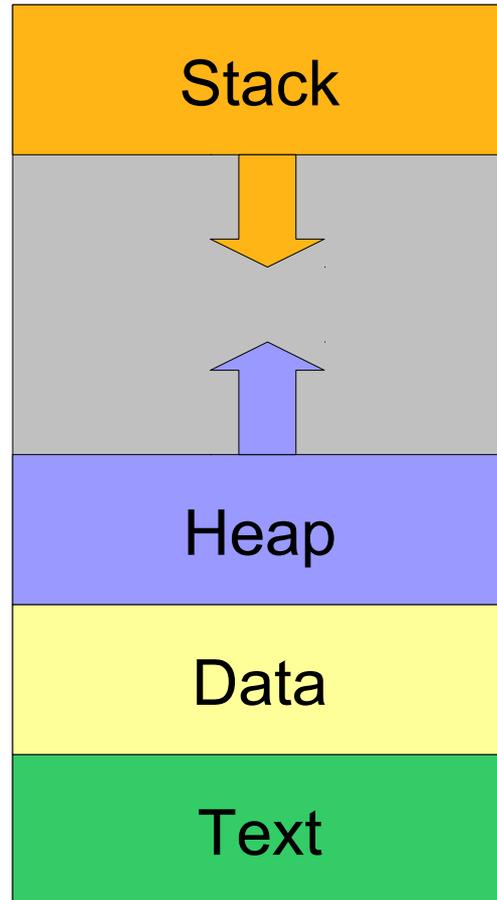


(b)



(c)

Spazio di indirizzamento di un processo



Organizzazione dei processi

- Il sistema operativo (kernel) identifica univocamente ogni processo tramite un **PID** (process identifier)
- Il **PID** è assegnato alla creazione del processo

```

attila@dusty: ~
File Modifica Visualizza Cerca Terminale Aiuto
attila@dusty:~$ ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0 24320 1788 ?        Ss   Feb03  0:01 /sbin/init
root         2  0.0  0.0     0     0 ?        S    Feb03  0:00 [kthreadd]
root         3  0.0  0.0     0     0 ?        S    Feb03  0:17 [ksoftirqd/0]
root         6  0.0  0.0     0     0 ?        S    Feb03  0:00 [migration/0]
root         7  0.0  0.0     0     0 ?        S    Feb03  0:00 [migration/1]
root         9  0.0  0.0     0     0 ?        S    Feb03  0:06 [ksoftirqd/1]
root        11  0.0  0.0     0     0 ?        S    Feb03  0:00 [migration/2]
root        13  0.0  0.0     0     0 ?        S    Feb03  0:05 [ksoftirqd/2]
root        14  0.0  0.0     0     0 ?        S    Feb03  0:00 [migration/3]
root        16  0.0  0.0     0     0 ?        S    Feb03  0:06 [ksoftirqd/3]
root        17  0.0  0.0     0     0 ?        S<   Feb03  0:00 [cpuset]
root        18  0.0  0.0     0     0 ?        S<   Feb03  0:00 [khelper]
root        19  0.0  0.0     0     0 ?        S<   Feb03  0:00 [netns]
root        21  0.0  0.0     0     0 ?        S    Feb03  0:00 [sync_supers]
root        22  0.0  0.0     0     0 ?        S    Feb03  0:00 [bdl-default]
root        23  0.0  0.0     0     0 ?        S<   Feb03  0:00 [kintegrityd]
root        24  0.0  0.0     0     0 ?        S<   Feb03  0:00 [kblockd]
root        25  0.0  0.0     0     0 ?        S<   Feb03  0:00 [ata_sff]
root        26  0.0  0.0     0     0 ?        S    Feb03  0:00 [khubd]
root        27  0.0  0.0     0     0 ?        S<   Feb03  0:00 [nd]
root        30  0.0  0.0     0     0 ?        S    Feb03  0:00 [khungtaskd]
root        31  0.0  0.0     0     0 ?        S    Feb03  0:13 [kswapd0]
root        32  0.0  0.0     0     0 ?        SN   Feb03  0:00 [ksmd]
root        33  0.0  0.0     0     0 ?        SN   Feb03  0:00 [khugepaged]
root        34  0.0  0.0     0     0 ?        S    Feb03  0:00 [fsnotify_mark]
root        35  0.0  0.0     0     0 ?        S    Feb03  0:00 [cryptfs-kthrea]

```

Name	ID	Priority	CPU	Mem Usage	User Name	Full Path
AdobeARM.exe	2256	Normal	0	7,844 K	utente1	C:\Programmi\File comuni\Adobe\ARM\1.0\Ado...
alg.exe	2260	Normal	0	912 K	SERVIZIO LOCALE	C:\WINDOWS\System32\alg.exe
aspnet_regis...	2916	Normal	0	7,960 K	SYSTEM	c:\WINDOWS\Microsoft.NET\Framework\v2.0.0...
csrss.exe	596	Normal	0	2,612 K	SYSTEM	C:\WINDOWS\system32\csrss.exe
ctfmon.exe	1888	Normal	0	856 K	utente1	C:\WINDOWS\system32\ctfmon.exe
Explorer.EXE	1532	Normal	0	2,084 K	utente1	C:\WINDOWS\Explorer.EXE
HotFixInstall...	3140	Normal	1	5,432 K	SYSTEM	c:\9d8a426282d4c389dd7136ebf2b3\HotFixIn...
ieexplore.exe	3528	Normal	0	1,524 K	utente1	C:\Programmi\Internet Explorer\ieexplore.exe
ieexplore.exe	3720	Normal	0	53,672 K	utente1	C:\Programmi\Internet Explorer\ieexplore.exe
lsass.exe	676	Normal	0	3,252 K	SYSTEM	C:\WINDOWS\system32\lsass.exe
msiexec.exe	2200	Normal	3	17,736 K	SYSTEM	C:\WINDOWS\system32\msiexec.exe
MsiExec.exe	2776	Normal	1	5,444 K	SYSTEM	C:\WINDOWS\system32\MsiExec.exe
MsMpEng.exe	1060	Normal	16	145,284 K	SYSTEM	C:\Programmi\Microsoft Security Client\Antimal...
msseces.exe	1880	Normal	0	3,556 K	utente1	C:\Programmi\Microsoft Security Client\mssece...
NDP20SP2-K...	1032	Normal	0	4,956 K	SYSTEM	C:\WINDOWS\SoftwareDistribution\Download\...
powershell.exe	3552	Normal	0	316 K	utente1	C:\WINDOWS\system32\windowspowershell\y...
PrcView.exe	2920	Normal	0	5,200 K	utente1	C:\DOCUME~1\utente1\IMPOST~1\Temp\Dire...
services.exe	664	Normal	1	2,744 K	SYSTEM	C:\WINDOWS\system32\services.exe
smss.exe	496	Normal	0	136 K	SYSTEM	C:\WINDOWS\System32\smss.exe
spoolsv.exe	1700	Normal	0	4,640 K	SYSTEM	C:\WINDOWS\system32\spoolsv.exe
svchost.exe	424	Normal	0	376 K	SYSTEM	C:\WINDOWS\system32\svchost.exe -k LocalS...
svchost.exe	888	Normal	0	1,608 K	SYSTEM	C:\WINDOWS\system32\svchost -k DcomLaunch
svchost.exe	964	Normal	0	1,752 K	SYSTEM	C:\WINDOWS\system32\svchost -k rpcss
svchost.exe	1128	Normal	2	27,952 K	SYSTEM	C:\WINDOWS\System32\svchost.exe -k netsvcs

Client: Server Runtime Process 5.1.2600.5512. © Microsoft Corporation. All rights reserved.

getpid

- La funzione **getpid** permette di ottenere il pid assegnato al processo corrente

```
#include <unistd.h>
```

```
pid_t getpid(void);
```

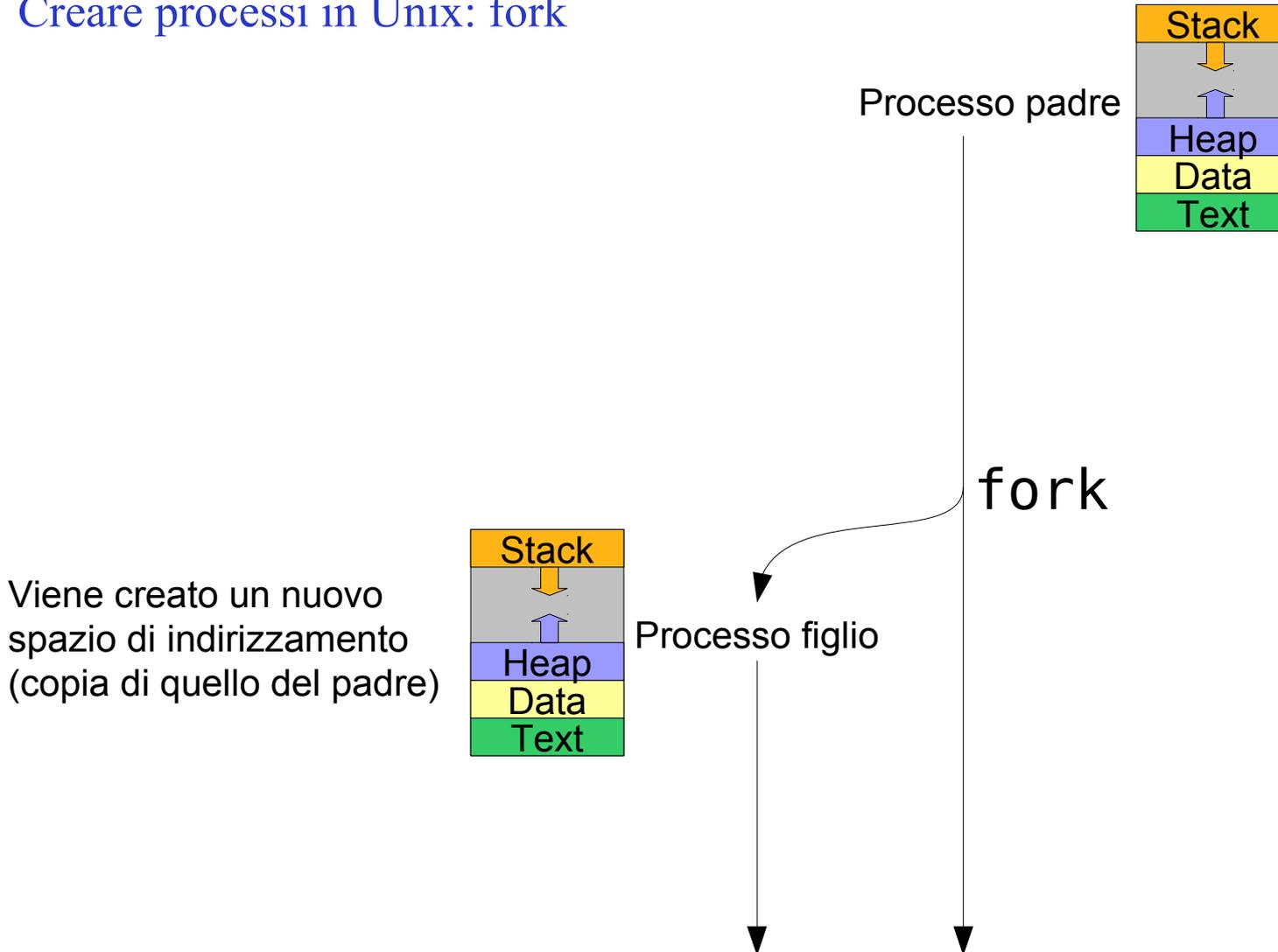
Creare processi in Unix: fork

```
#include <unistd.h>

pid_t fork(void);
```

1. Crea un nuovo spazio privato in memoria per il figlio
2. Crea una nuovo descrittore di processo per il figlio (→**nuovo PID**)
3. Copia lo spazio di memoria del padre in quello del figlio
4. Valore di ritorno:
 - Al padre: ritorna il PID del figlio
 - Al figlio: ritorna 0
 - Se il fork fallisce: ritorna -1

Creare processi in Unix: fork



Esempio fork

```
#include <unistd.h>
```

```
pid_t cpid;
```

```
cpid = fork();
```

```
if (cpid == (pid_t) -1) {  
    printf("Errore!\n");  
} else if (cpid == 0) {  
    printf("Sono il figlio\n");  
} else {  
    printf("Sono il padre\n");  
}
```

Gerarchia di processi

- Con fork viene creata una gerarchia di processi
 - In cima all'albero c'è il processo **init** (PID=1)
- Un processo può determinare il PID del padre con **getppid** (**get** parent **pid**)

```
#include <unistd.h>
```

```
pid_t getppid(void);
```

Eseguire un altro programma: Exec

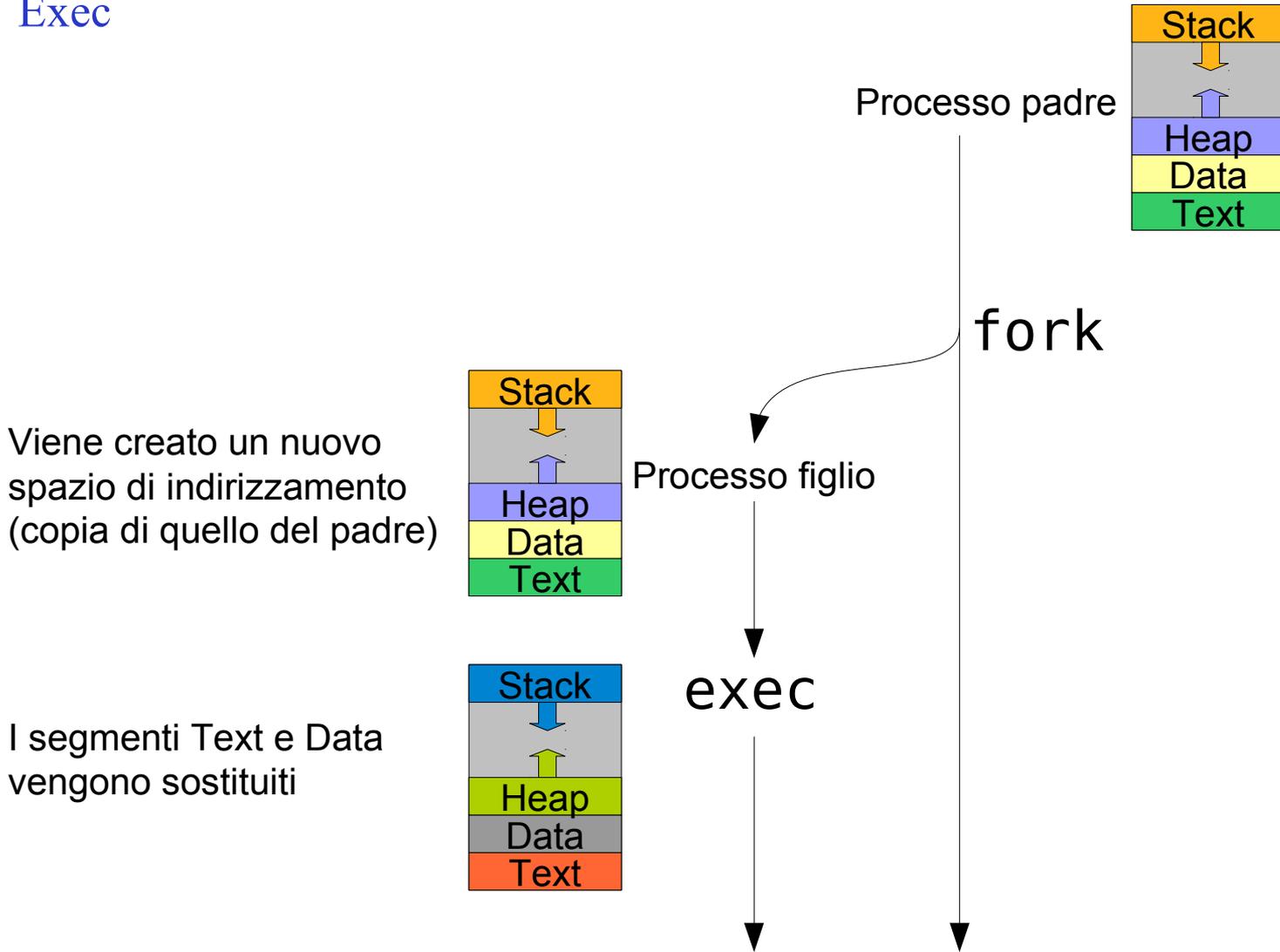
```
#include <unistd.h>
```

```
int execl(const char *path, const char*arg0, ...);  
int execlp(const char *file, const char *arg0, ...);  
...
```

Sostituisce i segmenti Text e Data del processo con quanto caricato dal file del programma specificato

Esistono diverse varianti di exec (cf. man exec)

Exec



Terminare un processo

- Un processo termina quando
 - **return** viene invocato dalla procedura main
 - viene invocata la procedura **exit(int)**
 - Il valore di ritorno/uscita può essere letto dal processo padre

```
#include <stdlib.h>
```

```
void exit(int status);
```

Attendere il valore di uscita di un processo figlio: Wait / Waitpid

- Il processo padre aspetta la terminazione dei processi figli e ottiene lo stato di uscita con le funzioni **wait** e **waitpid**

```
#include <sys/types.h>
```

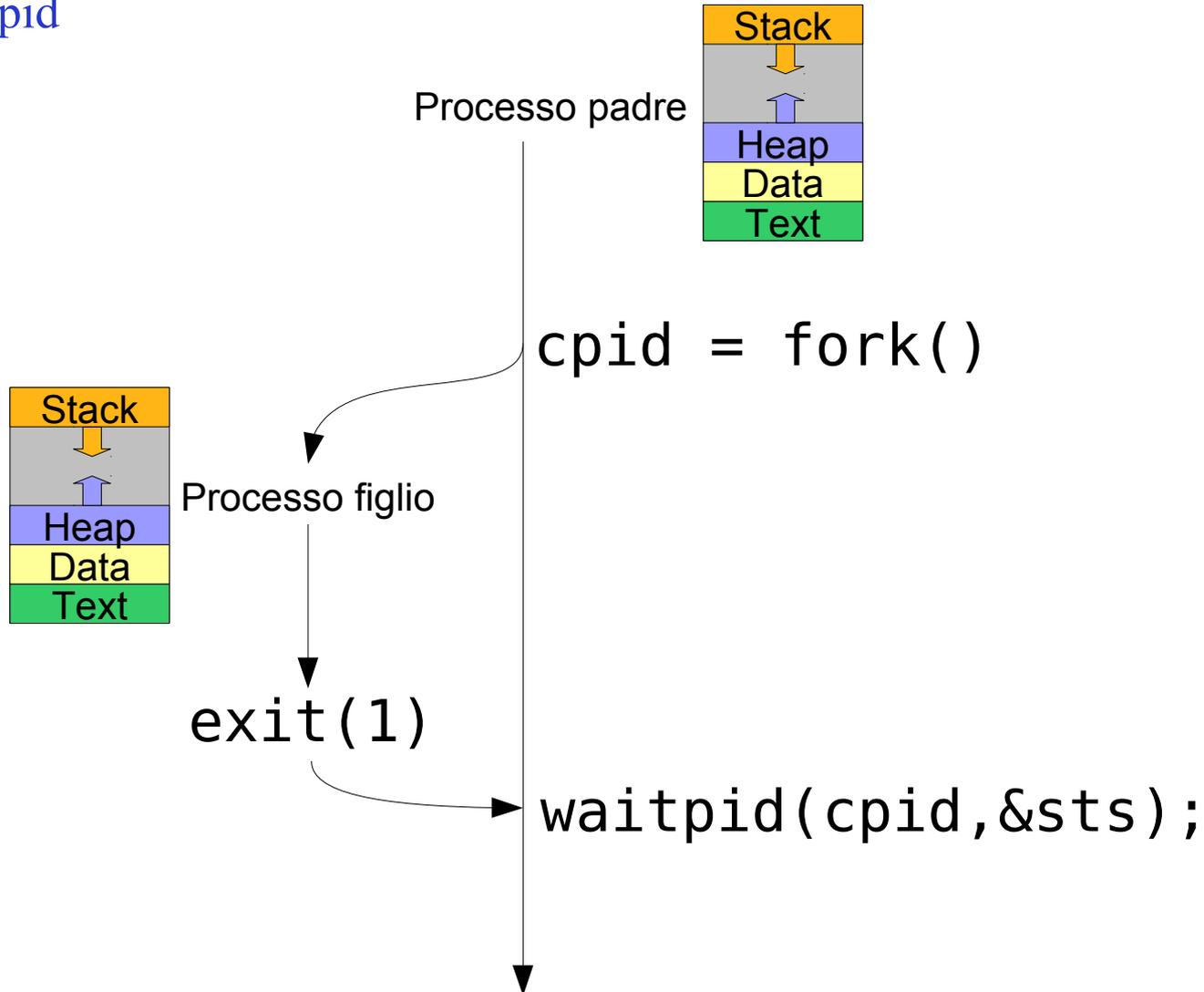
```
#include <sys/wait.h>
```

```
pid_t wait(int *status);
```

```
pid_t waitpid(pid_t pid, int *status,  
              int options);
```

* La terminazione è segnalata con SIGCHLD

waitpid



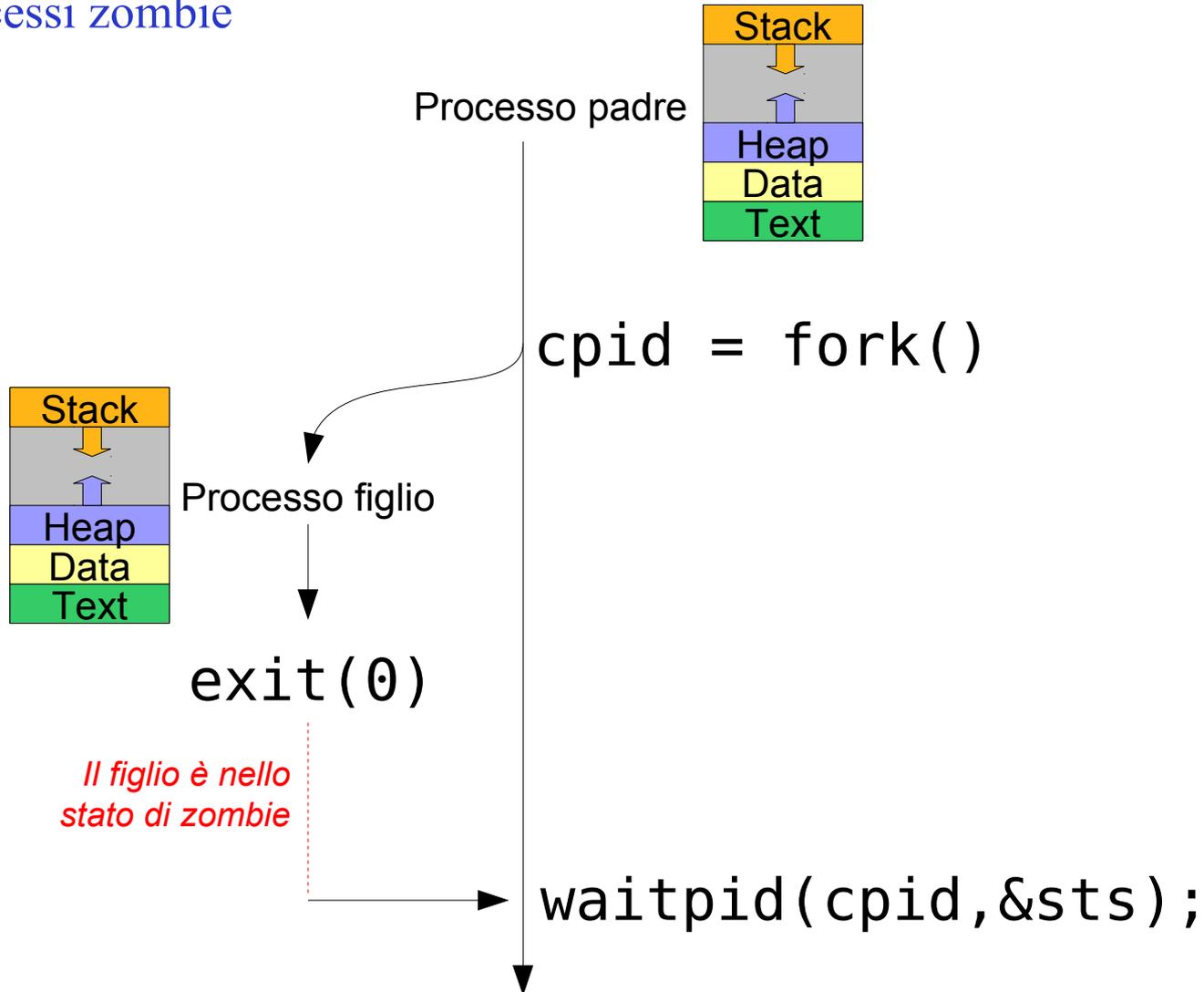
Esempio waitpid

```
#include <unistd.h>
#include <wait.h>
pid_t cpid;
int sts;
int main(void) {
    if (cpid = fork()) {
        waitpid(cpid, &sts, 0);
        printf("Figlio uscito con %d\n", WEXITSTATUS(sts));
    } else if (cpid == 0) {
        printf("Sono il figlio\n");
        exit(1);
    }
}
```

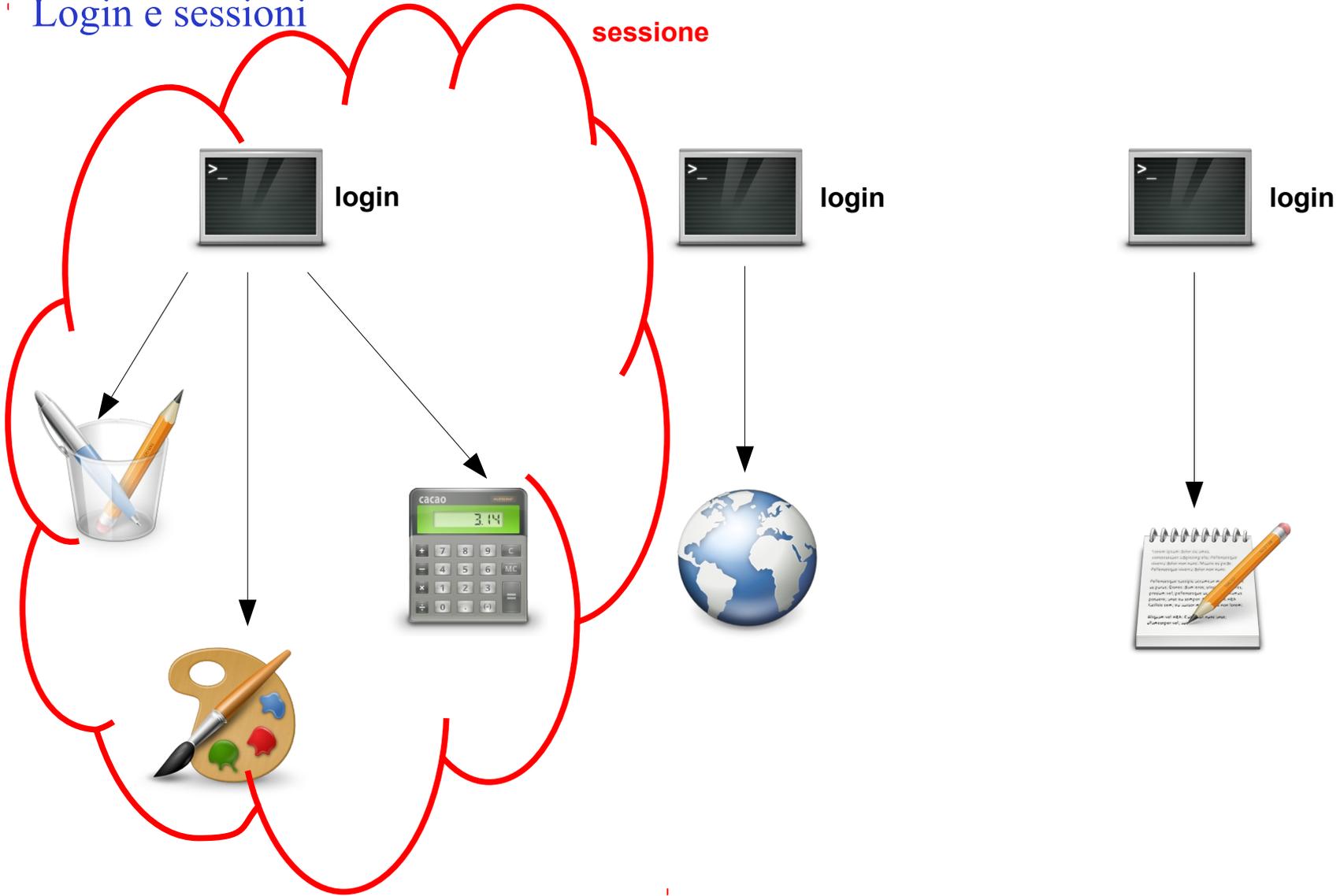
Processi zombie

- Quando il sotto-processo figlio termina è necessario che il padre sia in attesa con **wait** o **waitpid**
- Fintanto che il padre non attende, il figlio rimane nello stato **zombie** (o defunct) e non può essere rimosso dal sistema
 - Quando il padre termina il figlio viene rimosso
 - Se il padre termina prima del figlio, il sotto-processo viene ereditato da init

Processi zombie



Login e sessioni



Sessioni

- Quando un utente si collega al sistema il processo di login crea una nuova sessione
 - Ogni sessione ha un proprio identificatore (**SID**, **S**ession **ID**)
 - Tutti i sottoprocessi ereditano automaticamente l'identificatore di sessione
- Quando l'utente abbandona il sistema (logout) tutti i processi in esecuzione all'interno della sessione possono essere facilmente identificati e terminati

Sessioni

- Con **getsid** posso ottenere il SID

```
#include <unistd.h>
```

```
pid_t getsid(pid_t pid);
```

- Dopo un fork, un processo può creare una nuova sessione con **setsid**

```
#include <unistd.h>
```

```
pid_t setsid(void);
```

Gruppi di processi

- La filosofia Unix si basa sull'utilizzo di programmi semplici che possono essere combinati facilmente per svolgere operazioni complesse

```
ls *.txt | grep Hello | sort | head -n 5
```

Per controllare facilmente tutti i processi creati da questa linea di comando (p.es. per terminarli) è possibile raggrupparli...

Gruppi di processi

- Ogni processo è membro di un gruppo di processi
- Ogni gruppo è identificato univocamente da un **Process Group ID** (PGID)
- Ogni gruppo ha un leader, il cui PID è uguale al PGID del gruppo
- Ogni gruppo ha associato un terminale di controllo

Gruppi di processi

- Un processo eredita il gruppo dal padre, può conoscere il PGID con **getpgid**, e può cambiarlo con **setpgid**

```
#include <unistd.h>
```

```
pid_t getpgid(pid_t pid);
```

```
int setpgid(pid_t pid, pid_t pgid);
```

Se pgid è uguale pid o uguale a 0 viene creato un nuovo gruppo, di cui il processo è leader