

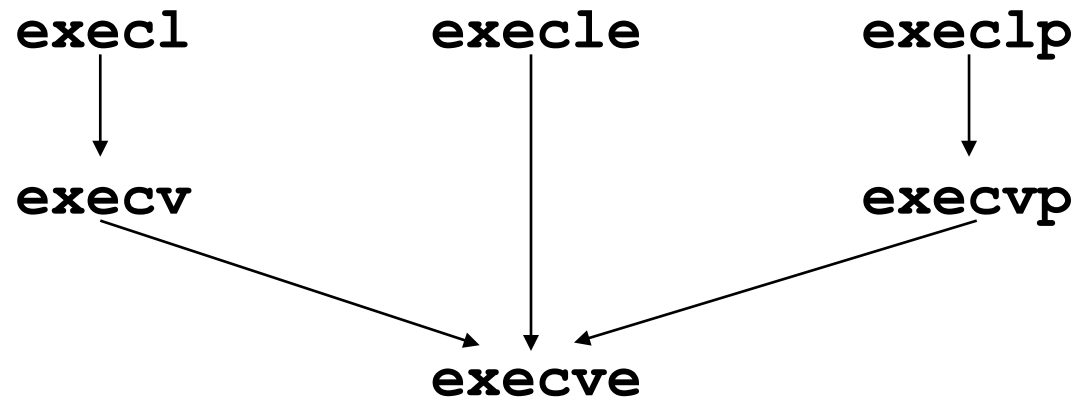
La famiglia di system call exec

Anno Accademico 2010-2011

Prof. Claudio Cilli

La famiglia di system call exec

- Se fork fosse l'unica primitiva per creare nuovi processi, la programmazione in ambiente Unix sarebbe alquanto ostica, dato che si potrebbero creare soltanto copie dello stesso processo
- La famiglia di primitive **exec** può essere utilizzata per superare tale limite in quanto le varie system call **exec** permettono di iniziare l'esecuzione di un altro programma sovrascrivendo la memoria del processo chiamante



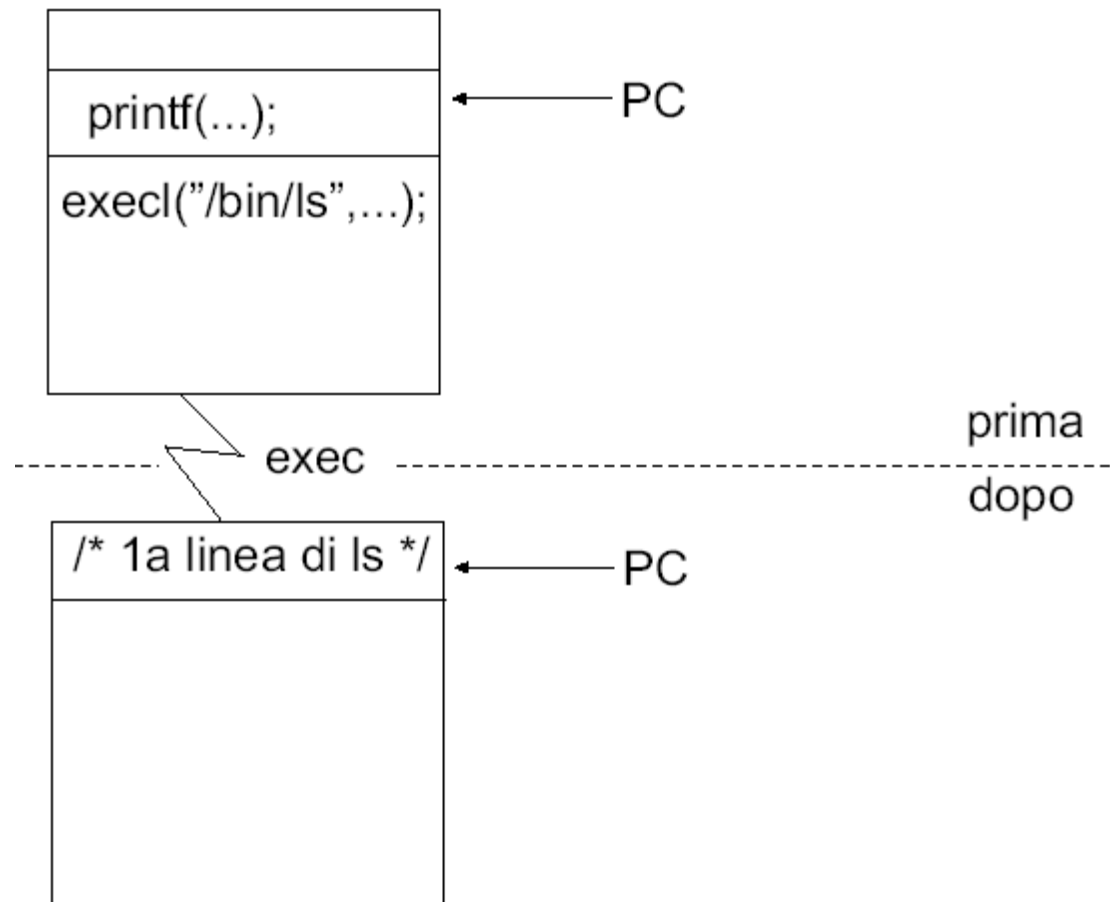
- In realtà tutte le funzioni chiamano in ultima analisi **execve** che è l'unica vera system call della famiglia. Le differenze tra le varianti stanno nel modo in cui vengono passati i parametri

Esempio di utilizzo di `exec1`

```
#include <stdio.h>
#include <unistd.h>
main()
{
printf("Esecuzione di ls\n");
exec1("/bin/ls", "ls", "-l", (char *)0);
perror("La chiamata di exec1 ha generato un
errore\n");
exit(1);
}
```

- Si noti che `exec1` elimina il programma originale sovrascrivendolo con quello passato come parametro. Quindi le istruzioni che seguono una chiamata a `exec1` verranno eseguite soltanto in caso si verifichi un errore durante l'esecuzione di quest'ultima ed il controllo ritorni al chiamante

Esempio di utilizzo di `exec1`



Chiamata alla system call exec

- Quando un processo chiama una delle system call `exec`:
 - il processo viene rimpiazzato completamente da un nuovo programma (text, data, heap, stack vengono sostituiti)
 - il nuovo programma inizia a partire dalla sua funzione `main`
 - il process ID non cambia
- Esistono sei versioni di `exec`:
 - con/senza gestione della variabile di ambiente `PATH`
 - se viene gestita la variabile d'ambiente, un comando corrispondente ad un singolo filename verrà cercato nel `PATH`
 - con variabili di ambiente ereditate / con variabili di ambiente specificate
 - con array di argomenti / con argomenti nella chiamata (null terminated)

Le varianti della system call

exec

```
int execl(char *pathname, char *arg0, ... );
int execv(char *pathname, char *argv[]);
int execlp(char *pathname, char *arg0, ...,
char* envp[]);
int execve(char *pathname, char *argv[],
char* envp[]);
int execlp(char *filename, char *arg0, ... );
int execvp(char *filename, char *argv[]);
```

Relazione tra suffisso e parametri di exec

Funzione	pathname	filename	arg list	argv[]	environ	envp[]
execl	●		●		●	
execlp		●	●		●	
execle	●		●			●
execv	●			●	●	
execvp		●		●	●	
execve	●			●		●
lettere del suffisso		p	l	v		e

Proprietà ereditate dal processo lanciato con `exec`

- Cosa viene ereditato da `exec`?
 - process ID e parent process ID
 - real uid e real gid
 - supplementary gid
 - process group ID
 - session ID
 - terminale di controllo
 - current working directory
 - root directory
 - maschera creazione file (`umask`)
 - file locks
 - maschera dei segnali
 - segnali in attesa

Proprietà NON ereditate dal processo lanciato con `exec`

- Cosa NON viene ereditato da `exec`?
 - effective user ID e effective group ID, in quanto vengono settati in base ai valori dei bit di protezione
- Cosa succede ai file aperti?
 - Dipende dal flag `close-on-exec` che è associato ad ogni descrittore
 - Se `close-on-exec` è true, vengono chiusi
 - Altrimenti, vengono lasciati aperti (comportamento di default)

Utilizzo combinato di `fork` e `exec`

- L'utilizzo combinato di `fork` per creare un nuovo processo e di `exec` per eseguire nel processo figlio un nuovo programma costituisce un potente strumento di programmazione in ambiente Linux

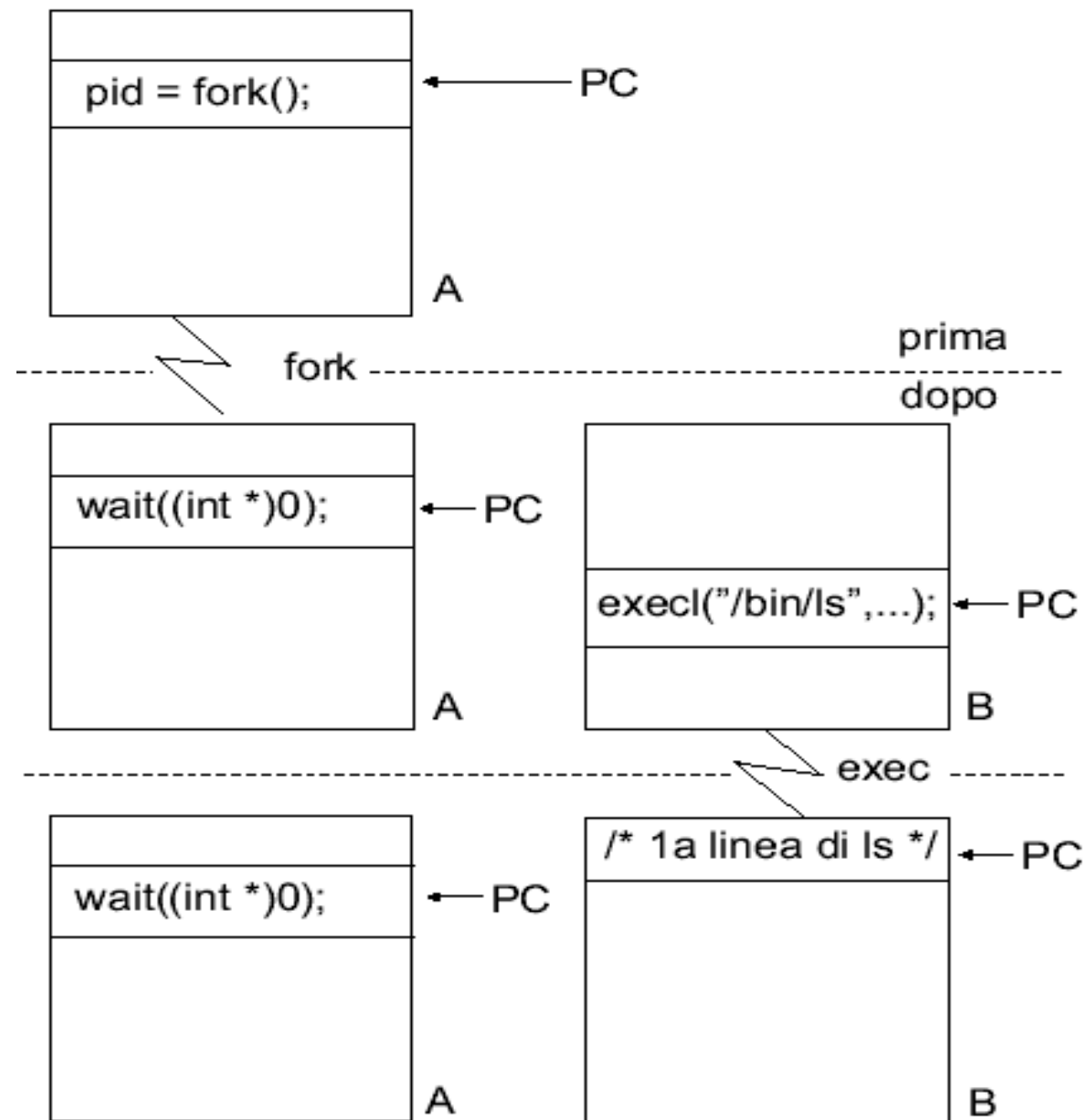
- Esempio:

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
void fatal(char *s)
{
    perror(s);
    exit(1);
}
/* continua ... */
```

Utilizzo combinato di `fork` e `exec`

```
main()
{
    pid_t pid;
    switch(pid = fork()) {
    case -1:
        fatal("fork failed");
        break;
    case 0:
        execl("/bin/ls", "ls", "-l", (char *)0);
        fatal("exec failed");
        break;
    default:
        wait((int *)0);
        printf("ls completed\n");
        exit(0);
    }
}
```

Utilizzo combinato di `fork` e `exec`



Esercitazione 1

- Scrivere un programma che manda in esecuzione un eseguibile il cui filename è inserito come argomento sulla linea comando e ne aspetta la terminazione
- Aggiungere al programma precedente la capacità di lanciare eseguibili residenti in qualsiasi directory memorizzata nella variabile di ambiente `$PATH`

La funzione `system`

- `int system(char* command);`
- Esegue un comando, aspettando la sua terminazione
- E' una funzione della libreria standard definita in ANSI C (quindi non è una system call, anche se svolge una funzione analoga alla `fork+exec`)
- Il suo modo di operare è fortemente dipendente dal sistema; in genere chiama `/bin/sh -c command`
- Non è definita in POSIX.1, perché non è un'interfaccia al sistema operativo, ma è definita in POSIX.2

L'ambiente di un processo

- L'ambiente di un processo è un insieme di stringhe (terminate da \0).
- Un ambiente è rappresentato da un vettore di puntatori a caratteri terminato da un puntatore nullo.
- Ogni puntatore (che non sia quello nullo) punta ad una stringa della forma: identificatore = valore
- Per accedere all'ambiente da un programma C, è sufficiente aggiungere il parametro envp a quelli del main:

```
/* showmyenv */  
#include <stdio.h>  
main(int argc, char **argv, char **envp)  
{  
while(*envp)  
printf("%s\n", *envp++);  
}
```

oppure usare la variabile globale seguente: `extern char **environ;`

L'ambiente di un processo

- L'ambiente di default di un processo coincide con quello del processo padre.
- Per specificare un nuovo ambiente è necessario usare una delle due varianti seguenti della famiglia `exec`, memorizzando in `envp` l'ambiente desiderato:

```
execle(path, arg0, arg1, ..., argn, (char *)  
0, envp);
```

```
execve(path, argv, envp);
```


L'ambiente di un processo

```
/* setmyenv */
#include <unistd.h>
#include <stdio.h>
main()
{ char *argv[2], *envp[3];
  argv[0] = "setmyenv";
  argv[1] = (char *)0;
  envp[0] = "var1=valore1";
  envp[1] = "var2=valore2";
  envp[2] = (char *)0;
  execve("./showmyenv", argv, envp);
  perror("execve fallita");
}
```

- Eseguendo il programma precedente si ottiene quanto segue:

```
$ ./setmyenv
var1=valore1
var2=valore2
```

L'ambiente di un processo

- Esiste una funzione della libreria standard che consente di cercare in `environ` il valore corrispondente ad una specifica variabile d'ambiente:

```
#include <stdlib.h>  
char *getenv(const char *name);
```

- `getenv` prende come argomento il nome della variabile da cercare e restituisce il puntatore al valore (ciò che sta a destra del simbolo `=`) o `NULL` se non lo trova:

```
#include <stdio.h>  
#include <stdlib.h>  
main()  
{  
printf("PATH=%s\n", getenv("PATH"));  
}
```

- Dualmente, `putenv` consente di modificare o estendere l'ambiente:

```
putenv("variabile=valore");
```

Current working directory e root directory

- Ad ogni processo è associata una current working directory che viene ereditata dal processo padre
- La chiamata di sistema seguente consente di cambiarla:

```
#include <unistd.h>  
int chdir(const char *path);
```

- Ad ogni processo inoltre è associata una root directory che specifica il punto di inizio del file system visibile dal processo stesso
- Per cambiare la root directory si può usare la chiamata di sistema seguente:

```
#include <unistd.h>  
int chroot(const char *path);
```

Esercitazione 2

- Modificare il programma dell'Esercitazione 1 (slide 13) in accordo alle seguenti ulteriori specifiche:
 - il programma deve supportare correttamente il lancio di eseguibili che richiedono opzioni ed argomenti
 - il pattern di programmazione composto dalla chiamata alle system call `fork+exec` dovrà essere inserito all'interno di una funzione `spawn ()` il cui prototipo dovrà essere opportunamente definito
- Aggiungere al programma precedente la capacità di leggere dallo standard input (e non come argomento sulla linea di comando) l'eseguibile da lanciare per poi lanciarlo alla pressione di `Enter`; quest'operazione andrà ripetuta iterativamente fino alla lettura di un carattere EOF di fine file (sequenza `ctrl+d`)

Esercitazione 3

- Aggiungere al programma dell'Esercitazione 2:
 - la stampa all'inizio di ogni nuova linea di comando (e quindi alla terminazione di ogni eseguibile lanciato) di un prompt contenente il nome dell'utente e quello del computer, nel formato:
`username@hostname $`
 - la capacità di poter lanciare un nuovo eseguibile senza dover attendere la terminazione di quello precedente se al termine della stringa di input si trova il carattere &
- Sviluppare un propria implementazione della funzione `system ()` (si consulti `man 3 system`), rispettandone il prototipo e la funzionalità; la gestione dei segnali definita per la `system ()` può essere ignorata