Sistemi Operativi

Laboratorio 5

System call exec

```
#include <unistd.h>
int execl(const char *pathname, const char *arg, ...);
int execlp(const char *pathname, const char *arg, ...);
int execle(const char *pathname, const char *arg , ..., char *const envp[]);
int execv(const char *pathname, char *const argv[]);
int execvp(const char *pathname, char *const argv[]);
```

Per ricordare:

- `l` → argomenti del programma passati come lista (terminata da un puntatore a NULL)
- `v` → argomenti del programma passati come vettore (terminato da NULL)
- `e` → per passare variabile ambiente al programma da eseguire (terminato da NULL)
- `p` → il programma viene ricercato in base alla variabile ambiente PATH

Dopo la chiamata ad exec:

- In caso di errore nell'invocazione del programma, exec ritorna -1
- Altrimenti, il programma eseguito non effettua il ritorno al chiamante
  - Tutto ciò che c'è dopo NON viene eseguito!!

System call system

```
#include <stdlib.h>

int system(const char *command);
```

Dopo la chiamata a system:

- Il comando eseguito effettua il ritorno al chiamante
  - Tutto ciò che c'è dopo viene eseguito!!

Grafo di precedenza

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

pid_t fork(void);

- Crea un processo figlio

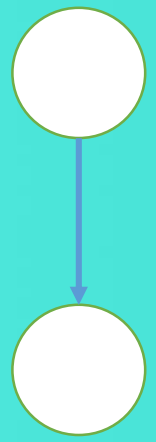
unsigned int sleep(unsigned int seconds);

- Sospende il processo corrente per i secondi specificati

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

- Sospende il processo corrente e attende che che un figlio termini

Grafo di precedenza

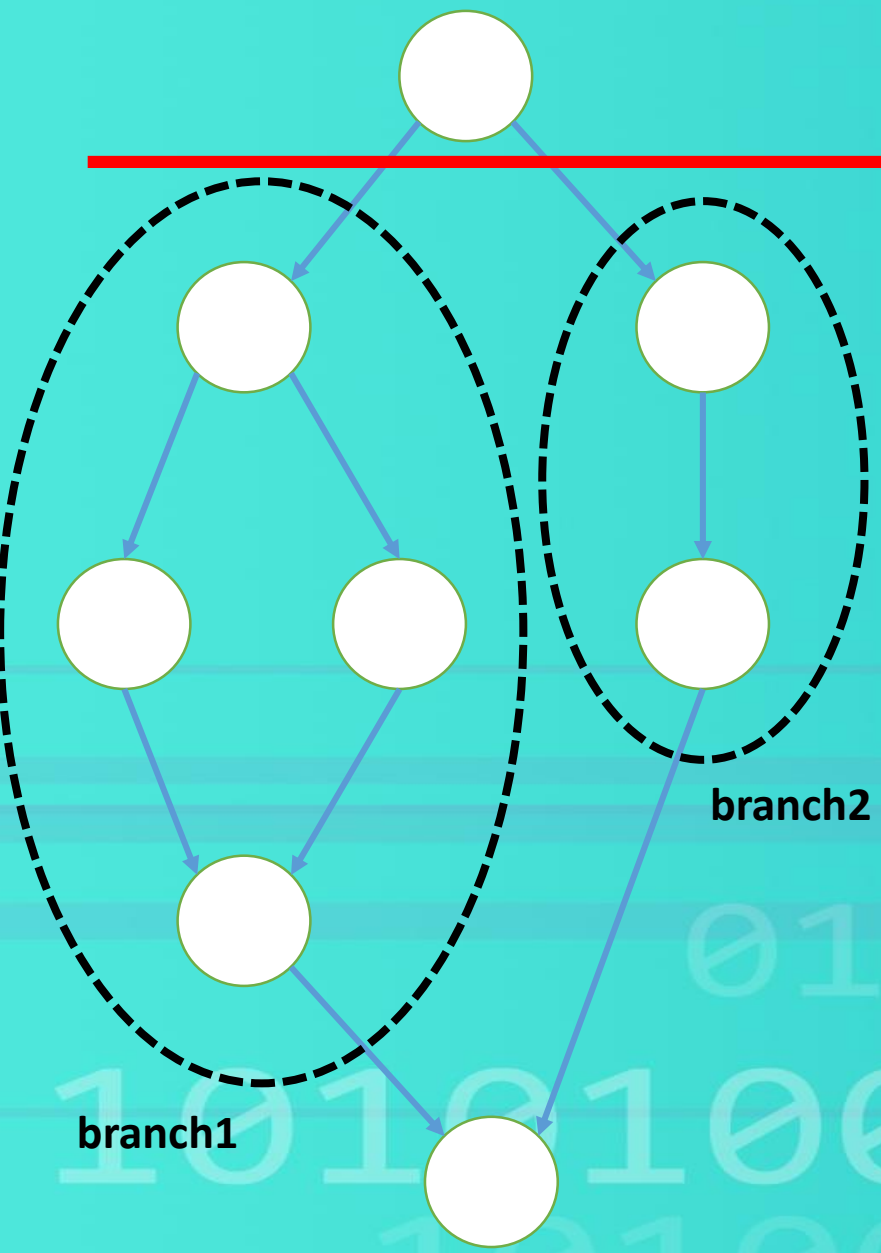


Versione con processi non ciclici (esercizi di questa settimana)

Ogni nodo deve:

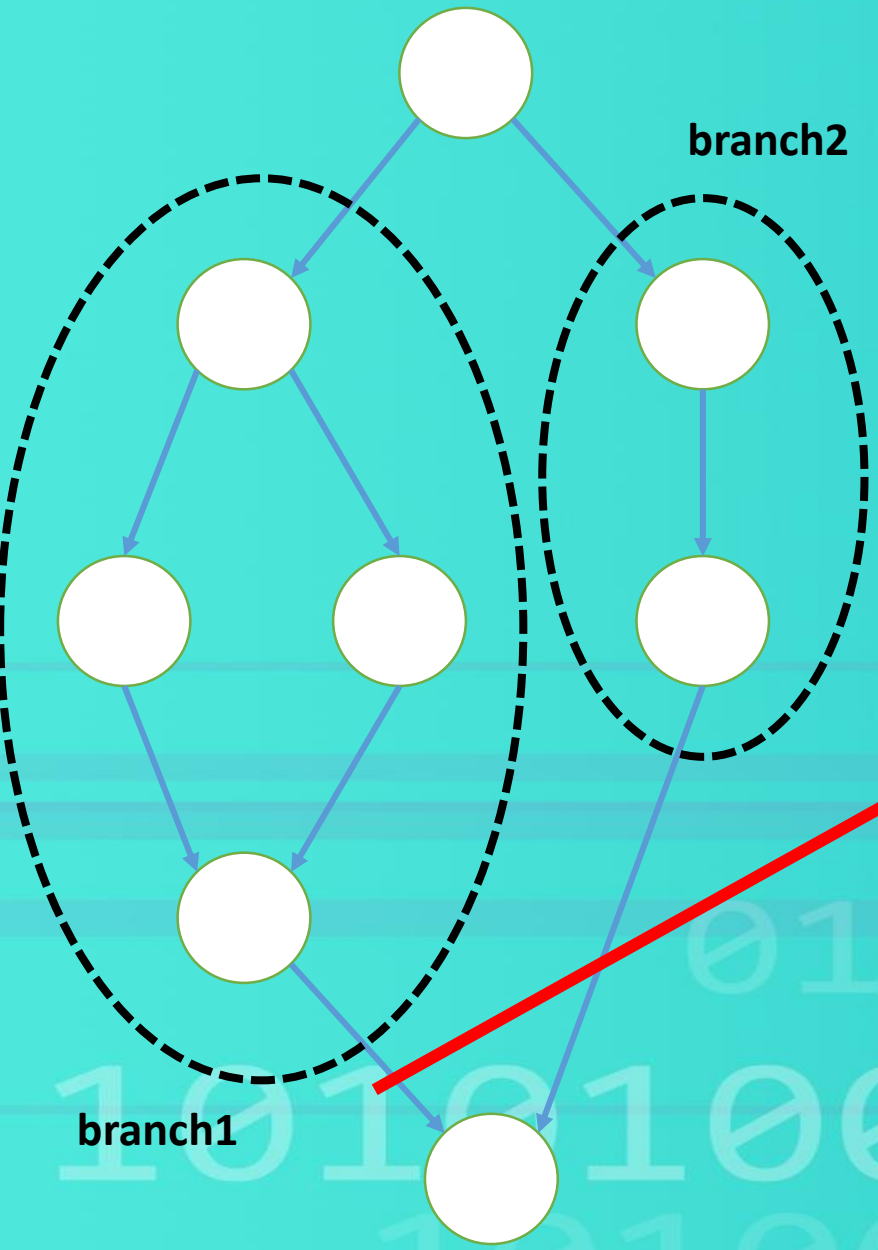
- **Attendere la terminazione del nodo precedessore**
- **Gestire la creazione di un processo**
- **Terminare l'esecuzione del processo creato (per sbloccare il nodo successore)**

Suggerimento:
Creare dei processi aggiuntivi per gestire dei sotto-grafi (vedere esempio)




```
/* nodo precedente completato */
…
branch1 = fork();
if (!branch1) {
 /* gestire nodi interni */
 …
 exit(0);
}

branch2 = fork();
if (!branch2) {
 /* gestire nodi interni */
 …
 exit(0)
}
```




```
…
waitpid(branch1, …);
waitpid(branch2, …);
…
/* gestire nodo successivo */
```