LABORATORIO SISTEMI OPERATIVI: IPC SIGNALS

- **SYSCALL EXEC**:
  - È una **famiglia di funzioni** (definite della libreria unistd.h) che **sostituiscono l'immagine di un processo**.
  - Quando si esegue un programma, in memoria viene creata un'immagine relativa all'esecuzione di questo programma. Quest'immagine contiene una serie di informazioni che caratterizzano il programma (tabella di file descriptor, variabili, istruzioni da eseguire, ecc.).
  - Tramite la **chiamata exec** è possibile **richiamare un eseguibile esterno** e far si che **l'immagine del processo attualmente in esecuzione venga sostituita completamente con l'eseguibile richiamato**.
    - Il processo tuttavia **mantiene la sua identità** (PID) e la sua presenza all'interno di una certa gerarchia di processi se dovesse averla.
    - Dopo l'esecuzione della syscall exec, **l'esecuzione del processo sarà legata al nuovo eseguibile** che abbiamo specificato.
    - Viene usata per far si che al momento della creazione di un processo figlio, questo venga legato con un programma esterno.

  - Esistono **due famiglie exec** che si differenziano per numero di argomenti che si possono passare:
    - **execl** / **execle** / **execlp** / **execlpe**:
      - prendono un **numero variabile di argomenti**.
    - **execv** / **execve** / **execvp** / **execvpe**:
      - prendono **un vettore di stringhe come argomento**.

  - Le funzioni **execle** / **execlpe** e **execve** / **execvpe**:
    - Consentono di passare un **vettore aggiuntivo di stringhe** (della forma "**key=value**") che permette di **lanciare il nuovo eseguibile all'interno di un nuovo ambiente** in cui sono state definite o ridefinite alcune variabili d'ambiente.

  - Le funzioni **execlp** / **execlpe** o **execvp** / **execvpe**:
    - Fanno sì che **l'argomento principale** che passiamo (eseguibile che vogliamo sostituire all'immagine del processo attualmente in esecuzione) **venga cercato nel path di sistema**.
    - Tutte le **altre funzioni richiedono che l'eseguibile sia indicato con un percorso completo** (bisogna indicare dove si trova l'eseguibile).

```c
#include <stdio.h>

int main(){
  char *args[] = {"./exec1", NULL};
  //vuole come parametri il nome dell'eseguibile da usare e poi una lista di argomenti
  //dopo execvp() tutto viene sovrascritto con la nuova immagine
  execvp(args[0], args);
  printf("I'm ALICE\n");
  return 0;
}
```

```c
#include <stdio.h>

int main(){
  printf("I'm BOB\n");
  return 0;
}
```

- **SIGNAL**:
  - Sono **interruzioni a livello software** (contenute nella libreria signal.h) e sono **identificate** da un **numero** e **un'etichetta mnemonica**.
  - **Esempi**:
    - **SIGALRM**: permette di creare un segnale dopo un certo lasso di tempo.
    - **SIGCHLD**: notifica se il figlio ha terminato la sua esecuzione.
    - **SIGCONT**: se il processo era fermo lo fa ripartire.
    - **SIGINT**: interrompe il processo attualmente in esecuzione (CTRL+C).
    - **SIGKILL**: chiede l'eliminazione di un processo (non può essere catturato o ignorato).
    - **SIGQUIT**: segnale di chiusura del terminale.
    - **SIGSTOP**: segnale di stop (non può essere catturato o ignorato).
    - **SIGTERM**: segnale di terminazione.
    - **SIGUSR1/SIGUSR2**: segnali utilizzati per comodità personale, usati come segnali di servizio.

  - **Ogni processo** ha associato una **lista di segnali pendenti**, lista che contiene tutti i segnali che sono stati emessi, eventuali segnali che sono stati bloccati o eventi che sono stati associati a quel processo dal SO.
  - **Ogni segnale ha un suo gestore** (una funzione), che può essere **sovrascritto** e **adattato** alle nostre necessità.
  - Se il **gestore** non viene ridefinito, viene usato quello di **default** che può:
    - **Ignorare il segnale** (se può).
    - **Terminare** l'esecuzione del processo.
    - **Bloccare** l'esecuzione del processo.
    - **Continuare** l'esecuzione del processo.

  - Queste **liste** vengono **controllate ad ogni schedulazione** del processo:
    - Se **non ci sono segnali pendenti** o se ci sono ma sono tutti bloccati (non devono essere considerati) allora **l'esecuzione del processo va avanti** come nulla fosse successo.
    - Se **ci sono segnali pendenti** (devono essere gestiti) allora **viene invocato l'handler** del segnale.

  - **L'handler di default può essere sovrascritto tramite la syscall signal()** (per **SIGKILL** e **SIGSTOP** non si può sovrascrivere l'handler):

```
sighandler_t signal(
        int signum,             //segnale che vogliamo gestire
        sighandler_t action     //riferimento ad una funzione (definita precedentemente) che implementa l'handler vero e proprio.
);
```

    - La **funzione che realizza l'handler** è una funzione che deve **accettare un solo argomento**, che poi sarà popolato con il **valore del segnale per cui è stata richiamata**.
    - È possibile chiamare la syscall signal() e **utilizzare lo stesso handler per più processi differenti** (tramite l'argomento che viene passato all'handler, si riesce a sapere per quale di questi segnali è stata richiamata).

  - Sono definite anche le **macro**:
    - **SIG_DFL**: reset del handler
    - **SIG_IGN**: ignora il segnale

  - La funzione **signal() restituisce un riferimento al vecchio handler** (possibile creazione di una catena di handler).

```c
#include <stdio.h>
#include <signal.h>
#include <unistd.h>

int counter=1;

//funzione che intercetta un segnale modifica il suo handler
void sighandle_int(int sigs){
  printf("\n[SIGINT %d]\n", counter);
  if(counter++ == 3) signal(SIGINT, SIG_DFL); //se counter=3, ripristino l'handler del signale
}

int main(){
   //quando riceve un segnale di interruzione, esegue il mio handler e non quello di default
   signal(SIGINT, sighandle_int);
   while(1){
     printf(".");
     sleep(1);
     fflush(stdout); //force stdout
   };
   return 0;
};
```

- **Syscall kill():**
  - Usata per **mandare un segnale** da un processo ad un altro.

    int kill(
    pid_t pid pid //pid del processo a cui vogliamo mandare il segnale
    int signal //il segnale che si vuole inviare
    );

  - La funzione restituisce 0 se il segnale è stato emesso con successo.

- **Syscall alarm():**
  - **Emette un segnale SIGALRM al processo stesso che esegue la funzione dopo un numero di secondi** specificato come argomento nel momento dell'invocazione della funzione.

    unsigned int alarm(unsigned int seconds);

  - È possibile settare **un allarme alla volta**.
  - Quando richiamo la funzione questa mi **restituisce il numero di second**i eventualmente **mancanti** alla richiesta pendente che fosse già in corso.

- **Syscall pause():**
  - Permette di **interrompere l'esecuzione** del processo (può essere risvegliato tramite un altro segnale).

    int pause();