***Controllo dei processi,***

**IDENTIFICARE UN PROCESSO:**

* pid\_t ***getpid*** (void); //process ID del processo chiamante
* pid\_t ***getppid*** (void); //process ID del padre del processo chiamante

**CREAZIONE DI UN PROCESSO:**

* pid\_t ***fork***(void);

*Restituisce*: 0 nel figlio, *pid* del figlio nel padre, -1 in caso di errore.

**TERMINAZIONE DI UN PROCESSO:**

* void ***exit*** (int *status*);

*Descrizione*: restituisce status al processo che chiama il programma includente exit, effettua prima una pulizia e poi ritorna al kernel.

* void ***\_exit*** (int *status*);

*Descrizione*: ritorna immediatamente al kernel.

* int ***atexit*** (void (\**funzione*) (void));

*Restituisce*: 0 se OK, diverso da 0 su errore.

*funzione* = punta ad una funzione che è chiamata per effettuare operazioni di cleanup per il processo alla sua normale terminazione.

**SOSPENSIONE DI UN PROCESSO:**

* pid\_t ***wait*** (int \**statloc*);

*Descrizione*: chiamata da un processo padre ottiene in statloc lo stato di terminazione di un figlio.

*Restituisce*: PID se OK, -1 in caso di errore.

* pid\_t ***waitpid*** (pid\_t *pid*, int \**statloc*, int *options*);

*Descrizione*: chiamata da un processo padre chiede lo stato di terminazione in *statloc* del figlio specificato dal *pid* 1° argomento, tale processo padre si blocca in attesa o meno secondo il contenuto di *options*.

Restituisce: PID se OK, 0 oppure -1 in caso di errore.

* pid > 0 (pid del figlio che si vuole aspettare);
* pid == -1 (qualsiasi figlio...la rende simile a wait);
* pid == 0 (figlio con GroupID uguale al padre);
* pid < 0 (figlio con GroupID uguale a abs(pid));
* options = 0 (niente... come wait);
* options = WNOHANG (non blocca se il figlio indicato non è disponibile).

**ESECUZIONE DI UN PROCESSO:**

* int ***execl*** (const char \**path*, const char \**arg0*, ../\* (char \*) 0 \*/);
* int ***execv*** (const char \**path*, char \*const *argv*[ ]);
* int ***execle*** (const char \**path*, const char \**arg0*, ../\*(char \*) 0, char \*const *envp*[ ] \*/);
* int ***execve*** (const char \**path*, char \*const *argv*[ ], char \*const *envp*[ ]);
* int ***execlp*** (const char \**file*, const char \**arg0*, ../\*(char \*)0 \*/);
* int ***execvp*** (const char \**file*, char \*const *argv*[ ]);

*Restituiscono*: -1 in caso di errore, non ritornano se OK.

* *execl, execlp, execle* prendono come parametro la lista degli argomenti da passare al file da eseguire.
* *execv, execvp, execve* prendono come parametro l’array di puntatori agli argomenti da passare al file da eseguire.

**ESECUZIONE DI UNA SHELL:**

* int *system* (const char \**cmdstring*);

*Descrizione***:** *serve ad eseguire un comando shell dall’interno di un programma.*