**PROCESSI**

Identificatore del processo

pid\_t getpid(); ritorna il process id del processo

pid\_t getppid(); ritorna il process id del padre

- Con il comando 'top' o 'ps aux' vedo la lista dei programmi che stanno --girando.

Creazione

pid\_t fork();

Il figlio ritorna 0.

Il padre ritorna il pid del figlio.

In caso di errore ritorna -1 (setta errno).

- Lo spazio di indirizzamento del figlio è un duplicato di quello del --padre. Noi vediamo gli stessi indirizzi (virtuali) ma sono due copie --diverse.

- Stack = pila di frame, uno per ogni chiamata di funzione da cui non abbiamo ancora fatto ritorno.

- Area vuota

- Heap = variabili allocate dinamicamente.

- Data = variabili globali.

- Text = traduzione in codice macchina delle istruzioni.

- Padre e figlio hanno due tabelle dei descrittori di file diverse (il --figlio ne ha una copia).

- Se ci sono cose in buffer duplico anche quelle, risolvo facendo --fflush(stdout) prima della fork().

- Non possiamo assumere alcun ordinamento delle stampe.

- La fork() è costosa, la parte Text può essere condivisa ma Stack, Heap --e Data vanno duplicati.

- Implementata con copy-on-write.

- Serve memoria virtuale paginata.

- Nel figlio si copia solo la tabella delle pagine.

- le pagine sono marcate in sola lettura.

- se il figlio tenta la scrittura sono duplicate al volo dal kernel.

- Efficace perché spesso il figlio butta subito via tutto lo spazio di indirizzamento del padre.

Differenziazione

(1) Il contenuto del file eseguibile viene usato per sovrascrivere lo --- spazio di indirizzamento del processo che invoca exec\*().

(2) Si carica in PC l'indirizzo della prima istruzione compilata del ----main().

int execl(

char \*path, path dell'eseguibile

char \*arg0, primo argomento (nome file)

char \*arg1, secondo argomento (se c'è)

..., altri argomenti (se ci sono)

(char \*)NULL termina la lista

);

NON ritorna in caso di successo, -1 in caso di errore (setta errno).

- Fare fflush(stdout) prima di exec\*().

- execl(), execlp(), execle(), execv(), execvp(), execve().

- 'p' nel nome: cerca il file nelle directory specificate dalla variabile di ambiente PATH.

- 'v' nel nome: passare un array di argomenti secondo il formato di arg[].

- 'e' nel nome: passare un array di stringhe che descrivono l'ambiente.

- 'l' nel nome: passare gli argomenti o l'ambiente come lista terminata da NULL.

Terminazione

void \_exit(int status); non ritorna

- Si trova in <unistd.h>.

- Termina il processo, chiude tutti i file descriptor, libera lo spazio di indirizzamento.

- Invia un segnale SIGCHLD al padre.

- Salva status nella tabella dei processi in attesa che il padre lo accetti (wait(), waitpid()).

- I figli diventati orfani vengono adottati da init (ppid a 1).

void exit(int status); non ritorna

- Fa tutto quello che fa \_exit() più:

- atexit() se c'è.

- esegue il flush dei buffer I/O con fflush() o fclose().

int atexit(void \*function());

Ritorna 0 in caso di successo, ≠ 0 in caso di fallimento (NON setta errno).

- Registra la funzione function() in modo che sia chiamata quando il programma termina con exit() o return dal main().

- tipicamente usata per codice di pulizia.

- Se registro più funzioni verranno chiamate in ordine inverso.

Attesa del figlio

int waitpid(

pid\_t pid, pid o id del gruppo

int \*statusp, [1] puntatore status o NULL

int options [2] opzioni

);

Ritorna pid o 0 in caso di successo, -1 in caso di errore (setta errno).

- Attende che un figlio cambi di stato (terminato, sospeso, riattivato).

- Si possono attendere solo i figli direttamente attivati con la fork().

- pid > 0 attende il figlio pid.

- pid = -1 attende un qualsiasi figlio (ritorna il pid di quello che ha cambiato stato).

- pid = 0 attende un qualsiasi processo figlio nello stesso process group.

- pid < -1 attende un qualsiasi processo figlio nel gruppo -pid.

[1] statusp prende il codice di ritorno (1 byte, specificato nella \_exit() o exit()) più un insieme di altre informazioni recuperabili tramite maschere.

- WIFEXITED(status) vero se terminato con \*exit().

- WEXITSTATUS(status) recupera lo stato se WIFEXITED.

- WIFSIGNALED(status) vero se terminato con segnale.

- WTERMSIG(status) recupera il segnale se WIFSIGNALED.

[2] Se almeno un figlio è già terminato e il suo stato non è stato ancora letto con wait\*(), waitpid() termina subito altrimenti si blocca in attesa.

options uno o più flag combinati in or (|).

- WNOHANG se lo stato non è disponibile non si blocca ma ritorna subito 0.

int wait(int \*statusp); puntatore status o NULL

- Corrisponde a waitpid() con pid = -1 e nessuna opzione, quindi aspetta un qualsiasi figlio bloccandosi se necessario.

- Un figlio non può ritornare il suo stato più di una volta, quindi usare --wait() o waitpid() con pid = -1 è pericoloso.