NAME

Tcl_Fork, Tcl_WaitPids, Tcl_DetachPids − manage child processes

SYNOPSIS

#include <tcl.h>

int
Tcl_Fork( )

int
Tcl_WaitPids(*numPids, pidPtr, statusPtr*)

int
Tcl_DetachPids(*numPids, pidPtr*)

ARGUMENTS

int	*numPids*	(in)	Number of process ids contained in the array pointed to by *pidPtr*.
int	**pidPtr*	(in)	Address of array containing *numPids* process ids.
int	**statusPtr*	(out)	Address of place to store status returned by exited/suspended process.

DESCRIPTION

These procedures keep track of child processes in order to make it easier for one application to manage several children. If an application uses the UNIX *fork* and *wait* kernel calls directly, problems occur in situations like the following:

[1] One part of an application creates child C1. It plans to let the child run in background, then later wait for it to complete.

[2] Some other part of the application creates another child C2, not knowing anything about C1.

[3] The second part of the application uses *wait* to wait for C2 to complete.

[4] C1 completes before C2, so C1 is returned by the *wait* kernel call.

[5] The second part of the application doesn't recognize C1, so it ignores it and calls *wait* again. This time C2 completes.

[6] The first part of the application eventually decides to wait for its child to complete. When it calls *wait* there are no children left, so *wait* returns an error and the application never gets to examine the exit status for C1.

The procedures **Tcl_Fork**, **Tcl_WaitPids**, and **Tcl_DetachPids** get around this problem by keeping a table of child processes and their exit statuses. They also provide a more flexible waiting mechanism than the *wait* kernel call. Tcl-based applications should never call *fork* and *wait* directly; they should use **Tcl_Fork**, **Tcl_WaitPids**, and **Tcl_DetachPids**.

Tcl_Fork calls *fork* and returns the result of the *fork* kernel call. If the *fork* call was successful then **Tcl_Fork** also enters the new process into its internal table of child proceses. If *fork* returns an error then **Tcl_Fork** returns that same error.

Tcl_WaitPids calls *wait* repeatedly until one of the processes in the *pidPtr* array has exited or been killed or suspended by a signal. When this occurs, **Tcl_WaitPids** returns the process identifier for the process and stores its wait status at **statusPtr*. If the process no longer exists (it exited or was killed by a signal), then **Tcl_WaitPids** removes its entry from the internal process table. If *wait* returns a process that isn't in the *pidPtr* array, **Tcl_WaitPids** saves its wait status in the internal process table and calls *wait* again. If one of the processes in the *pidPtr* array has already exited (or suspended or been killed) when **Tcl_WaitPids** is called, that process and its wait status are returned immediately without calling *wait*.

Tcl_WaitPids provides two advantages. First, it allows processes to exit in any order, and saves their wait statuses. Second, it allows waiting on a number of processes simultaneously, returning when any of the processes is returned by *wait*.

Tcl_DetachPids is used to indicate that the application no longer cares about the processes given by the *pidPtr* array and will never use **Tcl_WaitPids** to wait for them. This occurs, for example, if one or more children are to be executed in background and the parent doesn't care whether they complete successfully. When **Tcl_DetachPids** is called, the internal process table entries for the processes are marked so that the entries will be removed as soon as the processes exit or are killed.

If none of the pids passed to **Tcl_WaitPids** exists in the internal process table, then -1 is returned and *errno* is set to ECHILD. If a *wait* kernel call returns an error, then **Tcl_WaitPids** returns that same error. If a *wait* kernel call returns a process that isn't in the internal process table, **Tcl_WaitPids** panics and aborts the application. If this situation occurs, it means that a process has been created without calling **Tcl_Fork** and that its exit status is about to be lost.

Tcl_WaitPids defines wait statuses to have type *int*, which is correct for POSIX and many variants of UNIX. Some BSD-based UNIX systems still use type *union wait* for wait statuses; it should be safe to cast a pointer to a *union wait* structure to *(int *)* before passing it to **Tcl_WaitPids** as in the following code:

```
union wait status;
int pid1, pid2;
...
pid2 = Tcl_WaitPids(1, &pid1, (int *) &status);
```

KEYWORDS

background, child, detach, fork, process, status, wait