Fork()

create a child process

#include <sys/types.h>

#include <unistd.h>

pid_t fork(void);

Process resource utilizations (getrusage(2)) and CPU time counters times(2)) are reset to zero in the child.

The child process is created with a single thread—the one that called fork(). The entire virtual address space of the parent is replicated in the child, including the states of mutexes, condition variables, and other pthreads objects; the use of pthread_atfork(3) may be helpful for dealing with problems that this can cause.

After a fork() in a multithreaded program, the child can safely call only async-signal-safe functions (see signal-safety(7)) until such time as it calls execve(2).

The child inherits copies of the parent's set of open file descriptors.

The child inherits copies of the parent's set of open message queue descriptors

The child inherits copies of the parent's set of open directory streams

On success, the PID of the child process is returned in the parent, and 0 is returned in the child. On failure, -1 is returned in the parent, no child process is created.

ERRORS

the RLIMIT_NPROC soft resource limit

the kernel's system-wide limit on the number of processes and threads

the maximum number of PIDs

failed to allocate the necessary kernel structures because memory is tight.

An attempt was made to create a child process in a PID namespace whose "init" process has terminated.

fork() is not supported on this platform

System call was interrupted by a signal and will be restarted.

Exit()

cause normal process termination

#include <stdlib.h>

void exit(int status);

Note that a call to execve(2) removes registrations created using atexit(3) and on_exit(3).

After exit(), the exit status must be transmitted to the parent process. There are three cases:

- If the parent has set SA_NOCLDWAIT, or has set the SIGCHLD handler to SIG_IGN, the status is discarded and the child dies immediately.

- If the parent was waiting on the child, it is notified of the exit status and the child dies immediately.

· Otherwise, the child becomes a "zombie" process: most of the process resources are recycled, but a slot containing minimal information about the child process (termination status, resource usage statistics) is retained in process table. This allows the parent to subsequently use waitpid(2) (or similar) to learn the termination status of the child; at that point the zombie process slot is released.

Wait()

wait, waitpid, waitid - wait for process to change state

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *wstatus);
pid_t waitpid(pid_t pid, int *wstatus, int options);
int waitid(idtype_t idtype, id_t id, siginfo_t *infop, int options);
```

waitpid

The value of pid can be:

< -1 meaning wait for any child process whose process group ID is equal to the absolute value of pid.

-1 meaning wait for any child process.

0 meaning wait for any child process whose process group ID is equal to that of the calling process.

> 0 meaning wait for the child whose process ID is equal to the value of pid.

The value of options is an OR of zero or more of the following constants:

WNOHANG return immediately if no child has exited.

WUNTRACED also return if a child has stopped (but not traced viaptrace(2)). Status for traced children which have stopped

is provided even if this option is not specified.

WCONTINUED also return if a stopped child has been resumed by delivery

of SIGCONT.

If wstatus is not NULL, wait() and waitpid() store status information in the int to which it points. This integer can be inspected with the] following macros:

WIFEXITED(wstatus)

returns true if the child terminated normally, that is, by calling exit() or by returning from main().

WEXITSTATUS(wstatus)

returns the exit status of the child.

WIFSIGNALED(wstatus)

returns true if the child process was terminated by a signal.

WTERMSIG(wstatus)

returns the number of the signal that caused the child process to terminate.

WCOREDUMP(wstatus)

returns true if the child produced a core dump.

WIFSTOPPED(wstatus)

returns true if the child process was stopped by delivery of a signal

WSTOPSIG(wstatus)

returns the number of the signal which caused the child to stop.

WIFCONTINUED(wstatus)

returns true if the child process was resumed by delivery of SIGCONT.

waitid()

The waitid() system call provides more precise control over which child state changes to wait for.

The idtype and id arguments select the child(ren) to wait for, as follows:

idtype == P_PID

Wait for the child whose process ID matches id.

idtype == P_PGID

Wait for any child whose process group ID matches id.

idtype == P_ALL

Wait for any child; id is ignored.

RETURN VALUE

wait(): on success, returns the process ID of the terminated child; on error, -1 is returned.

waitpid(): on success, returns the process ID of the child whose state has changed; if WNOHANG was specified and one or more child(ren) specified by pid exist, but have not yet changed state, then 0 is returned. On error, -1 is returned.

waitid(): returns 0 on success or if WNOHANG was specified and no child(ren) specified by id has yet changed state; on error, -1 is returned.

Execv()

execl, execlp, execle, execv, execvp, execvpe - execute a file

```
#include <unistd.h>

        extern char **environ;

        int execl(const char *path, const char *arg, ... /* (char  *) NULL */);

        int execlp(const char *file, const char *arg, ... /* (char  *) NULL */);

        int execle(const char *path, const char *arg, ... /*, (char *) NULL, char * const envp[] */);

        int execv(const char *path, char *const argv[]);

        int execvp(const char *file, char *const argv[]);

        int execvpe(const char *file, char *const argv[], char *const envp[]);
```

The const char *arg and subsequent ellipses in the execl(), execlp(), and execle() functions can be thought of as arg0, arg1, ..., argn. Together they describe a list of one or more pointers to null-terminated strings that represent the argument list available to the executed program.

Special semantics for execlp() and execvp()

The execlp(), execvp(), and execvpe() functions duplicate the actions of the shell in searching for an executable file if the specified file name does not contain a slash (/) character. The file is sought in the colon-separated list of directory pathnames specified in the PATH environment variable. If this variable isn't defined, the path list defaults to a list that includes the directories returned by conf-str(_CS_PATH) and possibly also the current working directory

If the specified filename includes a slash character, then PATH is ignored, and the file at the specified pathname is executed.

The exec() functions return only if an error has occurred. The return value is -1