NAME
exec, execv, execl, execve, execvp, execve, execvp – create a file

SYNOPSIS
#include <unistd.h>

int exec(const char *path, const char *argv[], ... , const char *argn, char * /""*/ NULL*/);  
int execv(const char *path, char *const argv[], ... , const char *argn, char * /""*/ NULL*/);  
int execl(const char *path, char *const arg0[], ... , const char *argn, char * /""*/ NULL*/);  
int execline(const char *path, char *const argv[], ... , const char *argn, char * /""*/ NULL*/);  
int execvp(const char *file, const char *const argv[], ... , const char *argn, char * /""*/ NULL*/);  
int execlp (const char * file, const char *const argv[], ... , const char *argn, char * /""*/ NULL*/);  
int execve(const char *path, char *const argv[], ... , char * const envp[]);  
int execline(const char *path, char *const argv[], ... , char * const envp[]);  
int execvp(const char *file, char *const argv[], ... , char * const envp[]);  

DESCRIPTION
Each of the functions in the exec family overlays a new process image on an old process. The new process image is constructed from an ordinary, executable file. This file is either an executable object file, or a file of data for an interpreter. There can be no return from a successful call to one of these functions because the calling process image is overlaid by the new process image.

When a C program is executed, it is called as follows:

int main (int argc, char * argv[], char * envp[]) ;

where argc is the argument count, argv is an array of character pointers to the arguments themselves, and envp is an array of character pointers to the environment strings. As indicated, argc is at least one, and the first member of the array points to a string containing the name of the file.

The arguments argv[0], ... , argv[n] point to null-terminated character strings. These strings constitute the argument list available to the new process image. Conventionally at least argv[0] should be present. The argv[0] argument points to a string that is the same as path (or the last component of path). The list of argument strings is terminated by a (char *) 0 argument.

The argv argument is an array of character pointers to null-terminated strings. These strings constitute the argument list available to the new process image. By convention, argv must have at least one member, and it should point to a string that is the same as path (or its last component). The argv argument is terminated by a null pointer.

The path argument points to a path name that identifies the new process file. The file argument points to the new process file. If file does not contain a slash character, the path prefix for this file is obtained by a search of the directories passed in the PATH environment variable (see environ()).

File descriptors open in the calling process remain open in the new process. Signals that are being caught by the calling process are set to the default disposition in the new process image (see signal(2)). Otherwise, the new process image inherits the signal dispositions of the calling process.

RETURN VALUE
If a function in the exec family returns to the calling process, an error has occurred; the return value is -1 and errno is set to indicate the error.

SYNOPSIS
#include <unistd.h>

pid_t fork(void);

DESCRIPTION
fork() creates a new process by duplicating the calling process. The new process, referred to as the child, is an exact duplicate of the calling process, referred to as the parent, except for the following points:

* The child has its own unique process ID, and this PID does not match the ID of any existing process group (setpgid(2)).
* The child's set of pending signals is initially empty (sigpending(2)).
* The child's set of resource limits is inherited from the parent (setrlimit(2)). Otherwise, the new process image inherits the resource limits of the parent.

When a C program is executed, the main() function is called as follows:

int main (int argc, char * argv[], char * envp[]) ;

where argc is the argument count, argv is an array of character pointers to the arguments themselves, and envp is an array of character pointers to the environment strings. As indicated, argc is at least one, and the first member of the array points to a string containing the name of the file.

The arguments argv[0], ... , argv[n] point to null-terminated character strings. These strings constitute the argument list available to the new process image. By convention, argv must have at least one member, and it should point to a string that is the same as path (or its last component). The argv argument is terminated by a null pointer.

The path argument points to a path name that identifies the new process file. If file does not contain a slash character, the path prefix for this file is obtained by a search of the directories passed in the PATH environment variable (see environ()).

File descriptors open in the calling process remain open in the new process. Signals that are being caught by the calling process are set to the default disposition in the new process image (see signal(2)). Otherwise, the new process image inherits the signal dispositions of the calling process.

RETURN VALUE
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, and errno is set appropriately.

ERRORS
EAGAIN
fork() cannot allocate sufficient memory to copy the parent's page tables and allocate a task structure for the child.

EAGAIN
It was not possible to create a new process because the caller's RLIMIT_NPROC resource limit was encountered. To exceed this limit, the process must have either the CAP_SYS_ADMIN or the CAP_SYS_RESOURCE capability.

ENOMEM
fork() failed to allocate the necessary kernel structures because memory is tight.

CONFORMING TO
SVr4, 4.3BSD, POSIX.1-2001.

SEE ALSO
done(2), execve(2), setrlimit(2), umeshare(2), vfork(2), wait(2), daemon(3), capabilities(7), credentials(7)
NAME
getc, getc, gets, getc, fgets, fputs, putc — input and output of characters and strings

SYNOPSIS
#include <stdio.h>
int fgetc(FILE **stream);
doc
int fgets(char **s, int size, FILE **stream);
int getc(FILE **stream);
int getchar(void);
int fputc(int c, FILE **stream);
int fputs(const char *s, FILE **stream);
int putc(int c, FILE **stream);
int putchar(int c);

DESCRIPTION
fgetc() reads the next character from *stream and returns it as an unsigned char cast to an int, or EOF on end of file or error.
getc() is equivalent to fgetc() except that it may be implemented as a macro which evaluates *stream more than once.
getchar() is equivalent to fgetc(stdin).
fgets() reads in at most one less than size characters from *stream and stores them into the buffer pointed to by s. Reading stops after an EOF or a newline. If a newline is read, it is stored into the buffer. A "\0" is stored after the last character in the buffer.
putc() writes the character c, cast to an unsigned char, to *stream.
fputc() writes the string s to *stream, without its terminating null byte ("\0").
putc() is equivalent to fputc() except that it may be implemented as a macro which evaluates *stream more than once.
putchar() is equivalent to putc(c, stdin).

Calls to the functions described here can be mixed with each other and with calls to other output functions from the stdio library for the same output stream.

RETURN VALUE
fgetc(), getc() and getchar() return the character read as an unsigned char cast to an int or EOF on end of file or error.
fgets() returns s on success, and NULL on error or when end of file occurs while no characters have been read.
putc(), putc() and putchar() return the character written as an unsigned char cast to an int or EOF on error.
fputc() returns a nonnegative number on success, or EOF on error.

SEE ALSO

waitpid(2)

NAME
waitpid — wait for child process to change state

SYNOPSIS
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid(pid_t pid, int *wstatus, int options);

DESCRIPTION
waitpid() suspends the calling process until one of its children changes state; if a child process changed state prior to the call to waitpid(), return is immediate. pid specifies a set of child processes for which status is requested.

If pid is equal to (pid_t)−1, status is requested for any child process.
If pid is greater than (pid_t)0, it specifies the process ID of the child process for which status is requested.
If pid is equal to (pid_t)0, status is requested for any child process whose process group ID is equal to that of the calling process.

If pid is less than (pid_t)−1, status is requested for any child process whose process group ID is equal to the absolute value of pid.

If waitpid() returns because the status of a child process is available, then that status may be evaluated with the macros defined by wstat(5). If the calling process had specified a non-zero value of stat_loc, the status of the child process will be stored in the location pointed to by stat_loc.

The options argument is constructed from the bitwise inclusive OR of zero or more of the following flags, defined in the header <sys/wait.h>:

WCONTINUED The status of any continued child process specified by pid, whose status has not been reported since it continued, is also reported to the calling process.

WNOHANG waitpid() will not suspend execution of the calling process if status is not immediately available for one of the child processes specified by pid.

WNOWAIT Keep the process whose status is returned in stat_loc in a waitable state. The process may be waited for again with identical results.

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

WIFEXITED(wstatus), WEXITSTATUS(wstatus), WIFSIGNALED(wstatus), WTERMSIG(wstatus)

RETURN VALUES
If waitpid() returns because the status of a child process is available, this function returns a value equal to the process ID of the child process for which status is reported. If waitpid() returns due to the delivery of a signal to the calling process, −1 is returned and errno is set to EINTR. If this function was invoked with WNOHANG set in options, it has at least one child process specified by pid for which status is not available, and status is not available for any process specified by pid, 0 is returned. Otherwise, −1 is returned, and errno is set to indicate the error.

ERRORS
waitpid() will fail if one or more of the following is true:

ECHILD The process or process group specified by pid does not exist or is not a child of the calling process or can never be in the states specified by options.
EINTR waitpid() was interrupted due to the receipt of a signal sent by the calling process.
EINVAL An invalid value was specified for options.