exec - Man Page

выполнить файл

Пролог

Эта страница руководства является частью Руководства программиста POSIX. Реализация этого интерфейса в Linux может отличаться (обратитесь к соответствующей странице руководства Linux для получения подробной информации о поведении Linux), или интерфейс может быть не реализован в Linux.

Краткое описание

```
#include <unistd.h>

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

Описание

Семейство *функций exec* должно заменить текущий образ процесса новым образом процесса. Новый образ должен быть построен из обычного исполняемого файла, называемого *новым файлом образа процесса*. Не должно быть возврата от успешного *exec*, потому что образ вызывающего процесса накладывается на новый образ процесса.

Функция *fexecve*() должна быть эквивалентна функции *execve*(), за исключением того, что файл, который должен быть выполнен, определяется файловым дескриптором *fd* вместо пути. Смещение файла *fd* игнорируется.

Когда программа на языке C выполняется в результате вызова одной из функций семейства exec, она должна быть введена как вызов функции на языке C следующим образом:

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

где *argc* – это счетчик аргументов, а *argv* – массив символьных указателей на сами аргументы. Кроме того, следующая переменная, которая должна быть объявлена пользователем, если она должна использоваться напрямую:

```
extern char **environ;
```

exec - Man Page

указатель, завершающий массив *argv*, не учитывается в *argc*.

Приложения могут изменить всю среду за одну операцию, назначив переменной *environ* указывать на массив символьных указателей на новые строки среды. После присвоения нового значения *environ*приложения не должны полагаться на новые строки среды, остающиеся частью среды, как это может сделать вызов *getenv*(), *putenv*(), *setenv*(), *unsetenv*() или любой функции, зависящей от переменной среды, заметив, что *environ* имеетизменено, скопируйте строки среды в новый массив и назначьте *environ*, чтобы указать на него.

Любое приложение, которое напрямую изменяет указатели, на которые указывает переменная environ, имеет неопределенное поведение.

Соответствующие многопоточные приложения не должны использовать переменную *environ* для доступа или изменения любой переменной среды, в то время как любой другой поток одновременно изменяет любую переменную среды. Вызов любой функции, зависящей от любой переменной окружения, считается использованием переменной *environ* для доступа к этой переменной окружения.

Аргументы, заданные программой с одной из функций *exec*, должны быть переданы новому образу процесса в соответствующих *аргументах main*().

Путь аргумента указывает на путь, идентифицирующий новый файл образа процесса.

Файл аргументов используется для создания пути, идентифицирующего новый файл образа процесса. Если аргумент *файла* содержит символ <slash>, аргумент *файла* должен использоваться в качестве пути к этому файлу. В противном случае префикс пути для этого файла получается путем поиска каталогов, передаваемых в качестве переменной окружения *PATH* (см. Базовый том определений POSIX.1-2017, *глава 8*, *Переменные окружения*). Если эта переменная окружения отсутствует, результаты поиска определяются реализацией.

Существует два различных способа, которыми содержимое файла образа процесса может привести к сбою выполнения, отличающихся установкой *errno* либо **[ENOEXEC]**, либо **[EINVAL]** (см. Раздел Ошибки). В тех случаях, когда другие члены семейства функций *exec* потерпят неудачу и установят *errno* в **[ENOEXEC]**, функции *execlp*() и *execvp*() должны выполнять интерпретатор команд, а среда выполняемой команды должна быть такой, как если бы процесс вызывал утилиту *sh* с помощью *execl*() следующим образом:

```
execl(<путь к оболочке>, arg0, file, arg1, ..., (char *)0);
```

где <*shell path*> - неуказанный путь для утилиты *sh*, *file* - файл образа процесса, а для *execvp*(), где *arg*0, *arg*1 и так далее соответствуют значениям, переданным *execvp*() в *argv*[0], *argv*[1] и так далее.

Аргументы, представленные *arg0*,..., являются указателями на символьные строки с нулевым окончанием. Эти строки должны составлять список

exec - Man Page

связанную с процессом, запускаемым одной из функций *exec*.

Аргумент *argv* представляет собой массив символьных указателей на строки с нулевым окончанием. Приложение должно убедиться, что последний член этого массива является нулевым указателем. Эти строки должны составлять список аргументов, доступных для нового образа процесса. Значение в *argv*[0] должно указывать на строку имени файла, связанную с процессом, запускаемым одной из функций *exec*.

Аргумент *envp* представляет собой массив символьных указателей на строки с нулевым окончанием. Эти строки должны составлять среду для нового образа процесса. Массив *envp завершается* нулевым указателем.

Для тех форм, которые не содержат указатель *envp* (*execl*(), *execv*(), *execlp*() и *execvp*()), среда для нового образа процесса должна быть взята из внешней переменной *environ* в вызывающем процессе.

Количество байтов, доступных для комбинированных списков аргументов и среды нового процесса, равно {ARG_MAX} . Определяется реализацией, включены ли в эту сумму нулевые терминаторы, указатели и / или любые байты выравнивания.

Файловые дескрипторы, открытые в образе вызывающего процесса, должны оставаться открытыми в новом образе процесса, за исключением тех, для которых установлен флаг close-on-*exec* FD_CLOEXEC. Для тех файловых дескрипторов, которые остаются открытыми, все атрибуты описания открытого файла остаются неизменными. Для любого файлового дескриптора, который закрыт по этой причине, блокировки файлов удаляются в результате закрытия, как описано в *close*() . Блокировки, которые не удаляются закрытием файловых дескрипторов, остаются неизменными.

Если файловый дескриптор 0, 1 или 2 в противном случае был бы закрыт после успешного вызова одной из функций семейства exec, реализации могут открыть неопределенный файл для файлового дескриптора в новом образе процесса. Если стандартная утилита или соответствующее приложение выполняется с файловым дескриптором 0, не открытым для чтения, или с файловым дескриптором 1 или 2, не открытым для записи, среда, в которой выполняется утилита или приложение, считается несоответствующей, и, следовательно, утилита или приложение могут вести себя не так, как описано в настоящем документе.стандарт.

Потоки каталогов, открытые в образе вызывающего процесса, должны быть закрыты в новом образе процесса.

Состояние среды с плавающей запятой в начальном потоке нового образа процесса должно быть установлено по умолчанию.

Состояние дескрипторов преобразования и дескрипторов каталога сообщений в новом образе процесса не определено.

Для нового образа процесса эквивалент:

exec - Man Page

Сигналы, установленные для действия по умолчанию (SIG_DFL) в образе вызывающего процесса, должны быть установлены для действия по умолчанию в новом образе процесса. За исключением SIGCHLD, сигналы, установленные для игнорирования (SIG_IGN) вызывающим образом процесса, должны игнорироваться новым образом процесса. Сигналы, которые будут перехвачены образом вызывающего процесса, должны быть установлены в действие по умолчанию в новом образе процесса (см. *<signal.h>*).

Если сигнал SIGCHLD задан для игнорирования вызывающим образом процесса, не указано, будет ли сигнал SIGCHLD задан для игнорирования или для действия по умолчанию в новом образе процесса.

После успешного вызова любой из функций *exec* альтернативные стеки сигналов не сохраняются, и флаг SA_ONSTACK должен быть очищен для всех сигналов.

После успешного вызова любой из функций *exec* любые функции, ранее зарегистрированные функциями *atexit*() или *pthread_atfork*(), больше не регистрируются.

Если бит ST_NOSUID установлен для файловой системы, содержащей новый файл образа процесса, то эффективный идентификатор пользователя, эффективный идентификатор группы, сохраненный set-user-ID и сохраненный set-group-ID остаются неизменными в новом образе процесса. В противном случае, если установлен бит режима set-user-ID нового файла образа процесса, эффективный идентификатор пользователя нового образа процесса должен быть установлен на идентификатор пользователя нового файла образа процесса. Аналогично, если установлен бит режима set-group-ID нового файла образа процесса, эффективный идентификатор группы нового образа процесса должен быть установлен на идентификатор группы нового файла образа процесса. Реальный идентификатор пользователя, реальный идентификатор группы и дополнительные идентификаторы групп нового образа процесса должны оставаться такими же, как и у вызывающего образа процесса. Эффективный идентификатор пользователя и эффективный идентификатор группы нового образа процесса должны быть сохранены (как сохраненный set-user-ID и сохраненный set-group-ID) для использования *setuid*().

Любые сегменты общей памяти, прикрепленные к образу вызывающего процесса, не должны быть прикреплены к новому образу процесса.

Любые именованные семафоры, открытые в вызывающем процессе, должны быть закрыты, как если бы соответствующие вызовы *sem_close*() .

Любые блоки типизированной памяти, которые были сопоставлены в вызывающем процессе, не сопоставляются, как если *бы munmap*() был неявно вызван для их сопоставления.

Блокировки памяти, установленные вызывающим процессом через вызовы *mlockall*() или *mlock*(), должны быть удалены. Если заблокированные страницы в адресном пространстве вызывающего процесса также отображаются

exec - Man Page

этого процесса функции exec. Если функция exec терпит неудачу, влияние на блокировки памяти не определено.

Сопоставления памяти, созданные в процессе, не отображаются до того, как адресное пространство будет перестроено для нового образа процесса.

Если вызывающий образ процесса не использует политики SCHED_FIFO, SCHED_RR или SCHED_SPORADIC планирования, политика планирования и параметры нового образа процесса и начального потока в этом новом образе процесса определяются реализацией.

Если в образе вызывающего процесса используются политики SCHED_FIFO, SCHED_RR или SCHED_SPORADIC scheduling, параметры политики процесса и планирования не должны изменяться вызовом функции exec. Начальный поток в новом образе процесса должен наследовать политику и параметры планирования процесса. Он должен иметь область системного конфликта по умолчанию, но должен наследовать свой домен распределения от образа вызывающего процесса.

Таймеры для каждого процесса, созданные вызывающим процессом, должны быть удалены перед заменой текущего образа процесса новым образом процесса.

Все открытые дескрипторы очереди сообщений в вызывающем процессе должны быть закрыты, как описано в *mq_close*() .

Любые выдающиеся асинхронные операции ввода-вывода могут быть отменены. Те асинхронные операции ввода-вывода, которые не отменены, должны завершаться так, как если бы функция exec еще не произошла, но любые связанные уведомления о сигналах должны быть подавлены. Не указано, *блокирует ли сама* функция exec ожидание такого завершения ввода-вывода. Однако ни в коем случае на новый образ процесса, созданный функцией *exec, не должно* влиять наличие выдающихся асинхронных операций ввода-вывода во время вызова функции exec. Отменяется ли какой-либо ввод-вывод и какой ввод-вывод может быть отменен при *exec*, определяется реализацией.

Новый образ процесса должен наследовать часы процессорного времени вызывающего образа процесса. Это наследование означает, что часы процессорного времени выполняемого процесса не должны быть повторно инициализированы или изменены в результате *функции exec*, кроме как для отражения времени, затраченного процессом на выполнение самой функции exec.

Начальное значение часов процессорного времени начального потока нового образа процесса должно быть равно нулю.

Если вызывающий процесс отслеживается, новый образ процесса должен продолжать отслеживаться в том же потоке трассировки, что и исходный образ процесса, но новый образ процесса не должен наследовать сопоставление имен событий трассировки с идентификаторами типов событий трассировки, которые были определены вызовами *posix_trace_eventid_open*() или *posix_trace_trid_eventid_openфункции*() в образе вызывающего процесса.

exec - Man Page

Как описано в функции *posix_trace_shutdown*() .

Идентификатор потока начального потока в новом образе процесса не указан.

Размер и расположение стека, в котором выполняется начальный поток в новом образе процесса, не указаны.

Начальный поток в новом образе процесса должен иметь тип отмены PTHREAD_CANCEL_DEFERRED и состояние отмены PTHREAD_CANCEL_ENABLED .

Начальный поток в новом образе процесса должен иметь все значения данных для конкретного потока, установленные в NULL, и все ключи данных для конкретного потока должны быть удалены вызовом *exec* без запуска деструкторов.

Начальный поток в новом образе процесса должен быть соединяемым, как если бы он был создан с атрибутом *detachstate*, установленным в PTHREAD_CREATE_JOINABLE .

Новый процесс должен наследовать по крайней мере следующие атрибуты от образа вызывающего процесса:

* Хорошее значение (см. *nice*())
* значения semadj (см. *semop*())
* Идентификатор процесса
* Идентификатор родительского процесса
* Идентификатор группы процессов
* Членство в сеансе
* Реальный идентификатор пользователя
* Реальный идентификатор группы
* Дополнительные идентификаторы групп
* Время, оставшееся до сигнала будильника (см. *alarm*())
* Текущий рабочий каталог
* Корневой каталог
* Маска создания режима файла (см. *umask*())
* Ограничение размера файла (см. *getrlimit*() и *setrlimit*())
* Маска сигнала процесса (см. *pthread_sigmask*())
* Отложенный сигнал (см. *sigpending*())
*

exec - Man Page

* Управляющий терминал

* Интервальные таймеры

Начальный поток нового процесса должен наследовать от вызывающего потока по крайней мере следующие атрибуты:

* Маска сигнала (см. *sigprocmask*() и *pthread_sigmask*())

* Отложенные сигналы (см. *sigpending*())

Все другие атрибуты процесса, определенные в этом томе POSIX.1-2017, должны быть унаследованы в новом образе процесса от старого образа процесса. Все другие атрибуты потока, определенные в этом томе POSIX.1-2017, должны быть унаследованы в исходном потоке в новом образе процесса от вызывающего потока в старом образе процесса. Наследование атрибутов процесса или потока, не определенных в этом томе POSIX.1-2017, определяется реализацией.

Вызов любой функции exec из процесса с более чем одним потоком должен привести к завершению всех потоков и загрузке и выполнению нового исполняемого образа. Никакие функции деструктора или обработчики очистки не должны вызываться.

После успешного завершения функции *exec* должны отметить для обновления последнюю метку времени доступа к данным файла. Если функция exec не удалась, но смогла найти файл образа процесса, не указано, отмечена ли последняя метка времени доступа к данным для обновления. В случае успешного выполнения функции exec файл образа процесса считается открытым с *помощью open*(). Считается, что соответствующий *close*() происходит через некоторое время после этого открытия, но до завершения процесса или успешного завершения последующего вызова одной из функций *exec*, *posix_spawn*() или *posix_spawnp*() . В массивы указателей argv[] и *envp*[] и строки, на которые указывают эти массивы, не должны изменяться вызовом одной из функций *exec*, за исключением случаев замены образа процесса.

Сохраненные ограничения ресурсов в новом образе процесса устанавливаются как копия соответствующих жестких и мягких ограничений процесса.

Возвращаемое значение

Если одна из функций *exec* возвращается к образу вызывающего процесса, произошла ошибка; возвращаемое значение должно быть -1, и *errno* должно быть установлено для указания ошибки.

Ошибки

exec - Man Page

Количество байтов, используемых списком аргументов нового образа процесса и списком среды, превышает установленный системой предел {ARG_MAX} байт.

EACCES

Новый файл образа процесса не является обычным файлом, и реализация не поддерживает выполнение файлов его типа.

EINVAL

Новый файл образа процесса имеет соответствующие привилегии и имеет признанный исполняемый двоичный формат, но система не поддерживает выполнение файла с этим форматом.

The *exec* functions, except for *fexecve*(), shall fail if:

EACCES

Разрешение на поиск отказано для каталога, указанного в префиксе пути нового файла образа процесса, или новый файл образа процесса отказывает в разрешении на выполнение.

ELOOP

Цикл существует в символических ссылках, встречающихся при разрешении аргумента *path* или *file* .

ENAMETOOLONG

Длина компонента пути больше {NAME_MAX} .

ENOENT

Компонент *path* или *file* не называет существующий файл или *путь* или *файл* является пустой строкой.

ENOTDIR

Компонент префикса пути к новому файлу образа процесса называет существующий файл, который не является ни каталогом, ни символической ссылкой на каталог, или путь к новому файлу образа процесса содержит по крайней мере один символ, не являющийся символом <slash>, и заканчивается одним или несколькими конечными символами <slash> и последнимкомпонент pathname называет существующий файл, который не является ни каталогом, ни символической ссылкой на каталог.

Функции *exec*, *за исключением execlp*() и *execvp*(), должны завершиться неудачно, если:

ENOEXEC

Новый файл образа процесса имеет соответствующее разрешение доступа, но имеет непризнанный формат.

Функция *fexecve*() завершится ошибкой, если:

exec - Man Page

выполнения.

The *exec* functions may fail if:

ENOMEM

The new process image requires more memory than is allowed by the hardware or system-imposed memory management constraints.

Функции *exec*, за исключением *fexecve*(), могут завершиться неудачно, если:

ELOOP

Во время разрешения аргумента path или file было обнаружено более {SYMLOOP_MAX} символических ссылок.

ENAMETOOLONG

Длина аргумента *path* или длина пути, построенного из аргумента *файла*, превышает {PATH_MAX} , или разрешение пути символической ссылки дало промежуточный результат с длиной, превышающей {PATH_MAX} .

ETXTBSY

Новый файл образа процесса представляет собой чистый файл процедуры (общий текст), который в настоящее время открыт для записи некоторым процессом.

Следующие разделы являются информативными.

Примеры

Использование execl()

Следующий пример выполняет команду *ls*, указывая путь к исполняемому файлу (**/bin/ls**) и используя аргументы, предоставленные непосредственно команде, для получения вывода с одним столбцом.

```
#include <unistd.h>

int ret;
...
ret = execl ("/bin/ls", "ls", "-1", (char *)0);
```

Использование execle()

Следующий пример аналогичен *использованию execl()*. Кроме того, он определяет среду для нового образа процесса, используя аргумент *env* .

```
#include <unistd.h>

int ret;
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
```

exec - Man Page

Использование execlp()

В следующем примере выполняется поиск расположения команды *ls* среди каталогов, указанных переменной среды PATH.

```
#include <unistd.h>

int ret;
...
ret = execlp ("ls", "ls", "-l", (char *)0);
```

Использование execv()

Следующий пример передает аргументы команде *ls* в массиве *cmd*.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
...
ret = execv ("/bin/ls", cmd);
```

Использование execve()

Следующий пример передает аргументы команде *ls* в массиве *cmd* и указывает среду для нового образа процесса, используя аргумент *env*.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
char *env[] = { "HOME=/usr/home", "LOGNAME=home", (char *)0 };
...
ret = execve ("/bin/ls", cmd, env);
```

Использование execvp()

В следующем примере выполняется поиск расположения команды *ls* среди каталогов, указанных переменной окружения PATH, и передаются аргументы команде *ls* в массиве *cmd*.

```
#include <unistd.h>

int ret;
char *cmd[] = { "ls", "-l", (char *)0 };
...
ret = execvp ("ls", cmd);
```

Использование приложений

Поскольку состояние дескрипторов преобразования и дескрипторов каталога сообщений в новом образе процесса не определено, соответствующие

exec - Man Page

Приложения, для которых в качестве глобальной локали в новом образе процесса требуется язык POSIX, отличный от стандартного, должны вызывать *setlocale*() с соответствующими параметрами.

При назначении нового значения переменной *environ* приложения должны убедиться, что среда, на которую она будет указывать, содержит по крайней мере следующее:

1. Любые переменные, определенные реализацией, необходимые реализации для обеспечения соответствующей среды. Подробнее см. запись _CS_V7_ENV в *<unistd.h>* и *confstr*() .
2. Значение для *PATH*, которое находит соответствующие версии всех стандартных утилит перед любыми другими версиями.

То же ограничение применяется к массиву *envp*, переданному *execle*() или *execve*(), чтобы гарантировать, что новый образ процесса вызывается в соответствующей среде.

Приложения не должны выполнять программы с файловым дескриптором 0, не открытым для чтения, или с файловым дескриптором 1 или 2, не открытым для записи, так как это может привести к неправильному поведению выполняемой программы. Чтобы не передавать эти файловые дескрипторы выполняемой программе, приложения должны не просто закрывать их, а открывать снова, например, **в файле /dev/null**. Некоторые реализации могут открывать их автоматически, но приложения не должны полагаться на это.

Если приложение хочет выполнить проверку контрольной суммы выполняемого файла перед его выполнением, файл должен быть открыт с разрешением на чтение для выполнения проверки контрольной суммы.

Поскольку разрешение на выполнение проверяется *fexecve*(), описание файла *fd* не обязательно должно быть открыто с флагом O_EXEC . Однако, если выполняемый файл запрещает чтение и запись для процесса, готовящегося выполнить *exec*, единственным способом предоставить *fd fexecve*() будет использование флага O_EXEC при открытии *fd*. В этом случае приложение не сможет выполнить тест контрольной суммы, поскольку оно не сможет прочитать содержимое файла.

Обратите внимание, что при открытии дескриптора файла в режиме O_RDONLY, O_RDWR или O_WRONLY дескриптор файла можно использовать для чтения, чтения и записи или записи файла соответственно, даже если режим файла изменяется после открытия файла. Использование открытого режима O_EXEC отличается; *fexecve*() проигнорирует режим, который использовался при открытии файлового дескриптора, и *exec* завершится неудачей, если режим файла, связанный с *fd*, не предоставит разрешение на выполнение вызывающему процессу во время *вызова fexecve*() .

Обоснование

exec - Man Page

проектах стандарта ISO C. Фактически, исторические реализации передавали нулевое значение, когда вызывающему функции exec не предоставлялись аргументы. Это требование было удалено из стандарта ISO C и впоследствии удалено из этого тома POSIX.1-2017. Формулировка, в частности использование слова *should*, требует, чтобы строго соответствующее приложение POSIX передавало хотя бы один аргумент функции *exec*, тем самым гарантируя, что *argc* быть одним или больше при вызове таким приложением. На самом деле, это хорошая практика, поскольку многие существующие приложения ссылаются *на argv*[0] без предварительной проверки значения *argc*.

Требование к строго соответствующему приложению POSIX также гласит, что значение, передаваемое в качестве первого аргумента, является строкой имени файла, связанной с запускаемым процессом. Хотя некоторые существующие приложения передают путь, а не строку имени файла, в некоторых случаях строка имени файла более полезна, поскольку обычно используется *argv*[0] в диагностике печати. В некоторых случаях переданное имя файла не является фактическим именем файла; например, многие реализации утилиты *входа* в систему используют соглашение о префиксе <дефис-минус> (**'-'**) к фактическому имени файла, которое указывает вызываемому интерпретатору команд, что это “оболочка входа”.

Кроме того, обратите внимание, что утилиты *test* и *[* требуют определенных строк для аргумента *argv*[0], чтобы иметь детерминированное поведение во всех реализациях.

Исторически сложилось так, что реализации могут выполнять сценарии оболочки двумя способами.

Одной из распространенных исторических реализаций является то, что функции *execl*(), *execv*(), *execle*() и *execve*() возвращают **ошибку [ENOEXEC]** для любого файла, который не распознается как исполняемый, включая сценарий оболочки. Когда функции *execlp*() и *execvp*() сталкиваются с таким файлом, они предполагают, что файл является сценарием оболочки и вызывают известный интерпретатор команд для интерпретации таких файлов. Теперь это требуется POSIX.1-2008. Эти реализации *execvp*() и *execlp*() дают **ошибку [ENOEXEC]** только в редком случае проблемы с исполняемым файлом интерпретатора команд. Из-за этих реализаций **Ошибка [ENOEXEC]** не упоминается для *execlp*() или *execvp*() , хотя реализации все еще могут дать ее.

Еще один способ, которым некоторые исторические реализации обрабатывают сценарии оболочки, - это распознавание первых двух байтов файла как символьной строки **"#!"** и использование оставшейся части первой строки файла в качестве имени командного интерпретатора для выполнения.

Один потенциальный источник путаницы, отмеченный разработчиками стандартов, заключается в том, как содержимое файла образа процесса влияет на поведение семейства функций *exec*. Ниже приводится описание предпринятых действий:

exec - Man Page

соответствующие привилегии) для этой системы, то система выполняет файл.

2. Если файл образа процесса имеет соответствующие привилегии и находится в формате, который является исполняемым, но недопустимым для этой системы (например, распознанный двоичный файл для другой архитектуры), то это ошибка, и *errno* устанавливается в **[EINVAL]** (см. Более позднее ОБОСНОВАНИЕ **[EINVAL]**).
3. Если файл образа процесса имеет соответствующие привилегии, но не распознан иным образом:
   a. Если это вызов *execlp*() или *execvp*() , то они вызывают интерпретатор команд, предполагая, что файл образа процесса является сценарием оболочки.

   b. Если это не вызов *execlp*() или *execvp*(), то возникает ошибка и *errno* устанавливается в **[ENOEXEC]**.

Приложения, которым не требуется доступ к своим аргументам, могут использовать форму:

```
main(void)
```

как указано в стандарте ISO C. Однако реализация всегда будет предоставлять два аргумента *argc* и *argv*, даже если они не используются.

Некоторые реализации предоставляют третий аргумент *main*(), называемый *envp*. Это определяется как указатель на среду. Стандарт ISO C указывает вызов *main*() с двумя аргументами, поэтому реализации должны поддерживать приложения, написанные таким образом. Поскольку этот том POSIX.1-2017 определяет глобальную переменную *environ*, которая также предоставляется историческими реализациями и может использоваться везде, где может использоваться envp , нет функциональной необходимости в аргументе envp. Приложения должны использовать функцию *getenv*(), а не обращаться к среде напрямую через *envp* или *environ*. Реализации необходимы для поддержки последовательности вызовов с двумя аргументами, но это не запрещает реализации поддерживать *envp* в качестве необязательного третьего аргумента.

Этот том POSIX.1-2017 указывает, что сигналы, установленные в SIG_IGN, остаются установленными в SIG_IGN, и что новый образ процесса наследует маску сигнала потока, который вызвал *exec* в старом образе процесса. Это согласуется с историческими реализациями и допускает некоторые полезные функции, такие как *nohup* команда. Однако следует отметить, что многие существующие приложения ошибочно предполагают, что они запускаются с определенными сигналами, установленными для действия по умолчанию и / или разблокированными. В частности, приложения, написанные с более простой моделью сигнала, которая не включает блокировку сигналов, таких как в стандарте ISO C, могут вести себя неправильно при вызове с некоторыми заблокированными сигналами. Поэтому лучше не блокировать или игнорировать сигналы через *exec*s без явной причины для этого, и особенно не

exec - Man Page

Функции *exec* всегда сохраняют значение эффективного идентификатора пользователя и эффективного идентификатора группы процесса при завершении *exec*, независимо от того, установлен ли бит set-user-ID или set-group-ID файла образа процесса.

Утверждение о *том, что argv*[] и *envp*[] являются константами, включено, чтобы дать понять будущим авторам языковых привязок, что эти объекты полностью постоянны. Из-за ограничения стандарта ISO C невозможно сформулировать эту идею в стандарте C. Указание двух уровней *const-квалификации* для параметров *argv*[] и *envp*[] для *exec* функции могут показаться естественным выбором, учитывая, что эти функции не изменяют ни массив указателей, ни символы, на которые указывает функция, но это запретит существующий правильный код. Вместо этого только массив указателей отмечается как константа. Таблица совместимости назначений для *dst=src*, полученная из стандарта ISO C, суммирует совместимость:

dst:	**char *[]**	**const char *[]**	**char *const[]**	**const char *const[]**
***Src*:**				
char *[]	ДОПУСТИМО	–	ДОПУСТИМО	–
const char *[]	–	ДОПУСТИМО	–	ДОПУСТИМО
char * const []	–	–	ДОПУСТИМО	–
const char *const[]	–	–	–	ДОПУСТИМО

Поскольку весь существующий код имеет тип источника, соответствующий первой строке, столбец, который дает наиболее допустимые комбинации, является третьим столбцом. Единственной другой возможностью является четвертый столбец, но его использование потребует приведения аргументов *argv* или envp. К сожалению, четвертый столбец не может быть использован, потому что объявление, которое, естественно, использовал бы неспециалист, было бы во второй строке.

Стандарт ISO C и этот том POSIX.1-2017 не конфликтуют при использовании *environ*, но некоторые исторические реализации *environ* могут вызвать конфликт. Пока *environ* обрабатывается так же, как точка входа (например, *fork*()), он соответствует обоим стандартам. Библиотека может содержать *fork*() , но если есть предоставленная пользователем *fork*() , этой *fork*() присваивается приоритет, и никаких проблем не возникает. Аналогичная ситуация с *environ*: определение в этом томе POSIX.1-2017 должно использоваться, если нет предоставленной пользователем *среды* чтобы иметь приоритет. Известно, что существуют по крайней мере три реализации, которые решают эту проблему.

E2BIG

Ограничение {ARG_MAX} применяется не только к размеру списка аргументов, но и к сумме этого и размера списка окружения.

EFAULT

exec - Man Page

EINVAL

Это условие ошибки было добавлено в POSIX.1-2008, чтобы позволить реализации обнаруживать исполняемые файлы, созданные для разных архитектур, и указывать эту ситуацию приложению. Исторические реализации оболочек, *execvp*() и *execlp*(), которые сталкиваются **с ошибкой [ENOEXEC]**, будут выполнять оболочку в предположении, что файл является сценарием оболочки. Это не даст желаемого эффекта, если файл является допустимым исполняемым файлом для другой архитектуры. Теперь реализация может решить избежать этой проблемы, вернув **[EINVAL]** когда встречается допустимый исполняемый файл для другой архитектуры. Некоторые исторические реализации возвращают **[EINVAL]**, чтобы указать, что аргумент *path* содержит символ с набором битов высокого порядка. Разработчики стандарта решили отклониться от исторической практики по следующим причинам:

1. Новое использование **[EINVAL]** обеспечит некоторую полезность для сообщества пользователей.
2. Историческое использование **[EINVAL]** неприемлемо в интернационализированной операционной среде.

ENAMETOOLONG

Поскольку путь к файлу может быть создан путем взятия элементов в переменной *PATH* и объединения их с именем файла, условие ошибки [ENAMETOOLONG] также может быть достигнуто таким образом.

ETXTBSY

System V возвращает эту ошибку, когда исполняемый файл в данный момент открыт для записи каким-либо процессом. Этот том POSIX.1-2017 не требует и не запрещает такое поведение.

Другие системы (например, System V) могут возвращать **[EINTR]** из *exec*. Это не рассматривается в этом томе POSIX.1-2017 , но реализации могут иметь окно между вызовом *exec* и временем, когда сигнал может вызвать возврат одного из *вызовов exec* с **помощью [EINTR]**.

Явный оператор, касающийся среды с плавающей запятой (как определено в заголовке *<fenv.h>*), был добавлен, чтобы прояснить, что среда с плавающей запятой устанавливается по умолчанию при успешном вызове одной из функций *exec*. Требования к наследованию или установке по умолчанию для других функций запуска процессов и потоков покрываются более общими операторами в их описаниях и могут быть обобщены следующим образом:

posix_spawn()

Значение по умолчанию.

fork()

Наследовать.

pthread_create()

exec - Man Page

Цель функции *fexecve*() – разрешить выполнение файла, который был проверен как предполагаемый файл. Можно активно проверять файл, читая из файлового дескриптора, и быть уверенным, что файл не обменивается на другой между чтением и выполнением. Кроме того, такая функция, как *openat*(), может использоваться для открытия файла, который был найден путем чтения содержимого каталога с помощью *readdir*() .

Будущие направления

Нет.

См. Также

сигнализация(), atexit(), chmod(), Закрыть(), confstr(), выход(), fcntl(), вилка(), fstatvfs(), getenv(), getitimer(), getrlimit(), mknod(), mmap(), неплохо(), Открыть(), posix_spawn(), posix_trace_create(), posix_trace_event(), posix_trace_eventid_equal(), pthread_atfork(), pthread_sigmask(), putenv(), readdir(), semop(), setlocale(), shmat(), sigaction(), sigaltstack(), sigpending(), система(), times(), ulimit(), umask()

Базовый том определений POSIX.1-2017, *Глава 8*, *Переменные среды*, <unistd.h>

Объем оболочки и утилит POSIX.1-2017, тест

Авторские права

Информация

exec - Man Page