2. Сообщения об ошибках

Много функций в GNU C библиотеке обнаруживают и выводят ошибки условий, и иногда ваши программы должны проверить эти ошибки условий. Например, когда Вы открываете входной файл, Вы должны проверить, что файл был фактически открыт правильно, и печатать сообщение об ошибках или выполнять другое соответствующее действие, если обращение к библиотечной функции потерпело неудачу.

Эта глава описывает, как работают средства сообщений об ошибках. Чтобы использовать эти средства, ваша программа должна включить заголовочный файл "errno.h".

2.1 Проверка Ошибок

Большинство библиотечных функций возвращает специальное значение, чтобы указать, что они потерпели неудачу. Специальное значение типично 1, нулевой указатель, или константа типа EOF, которая определена для той цели. Но это значение возврата сообщает Вам только то, что ошибка произошла. Чтобы выяснять что это было, Вы должны рассмотреть код ошибки, сохраненный в переменной errno. Эта переменная объявлена в заголовочном файле "errno.h".

Переменная errno содержит номер ошибки системы. Вы можете изменять значение errno.

С тех пор как errno объявлена изменяемой, она может быть асинхронно изменена драйвером сигнала; см. раздел 21.4 [Определение драйверов]. Однако, правильно написанный драйвер сигнала сохраняет и восстанавливает значение errno, так что Вы вообще не должны волноваться относительно этой возможности, разве что при наaписании драйверов сигнала.

Начальное значение errno при запуске программы - нуль. Большинство библиотечных функций, когда они сталкиваются с некоторыми видами ошибок, помещают туда некоторое отличное от нуля значение. Эти ошибки условий перечислены для каждой функции. Если эти функции успешно выполняются они не изменяют errno; таким образом, значение errno после успешного обращения не обязательно нулевое, и Вы не должны использовать errno, чтобы определить, потерпело ли обращение неудачу. Соответствующий способ проверки зарегистрирован для каждой функции. Если обращение неудачно, Вы может исследовать errno.

Многие библиотечные функции могут устанавливать errno отличным от нуля в результате вызова других библиотечных функций, которые возможно и установили ошибку. Т. е. если функция возвращает ошибку, то любая библиотечная функция могла изменять errno.

Примечание относительно переносимоси: ANSI C определяет errno скорее как "модифицируемое именуемое выражение", чем как переменную, разрешая ему выполняться как макрокоманде. Например, его расширение может включать обращение к функции, подобно * _errno (). Фактически, это встроено систему GNU непосредственно. GNU библиотека, на не-GNU системах, делает то, что правильно для этой специфической системы.

Имеются несколько библиотечных функций, подобно sqrt и atan, которые в случае ошибки возвращают ожидаемое значение, устанавивая также errno. Для этих функций, если Вы хотите выяснить, произошла ли ошибка, рекомендуется обнулить errno перед вызовом функции, и затем проверить значение позже.

Все коды ошибки имеют символические имена; т. е. это макрокоманды, определенные в "errno.h". Имена начинаются с "E" и символа верхнего регистра или цифры; Вы должны рассмотривать имена такой формы, как зарезервированные имена. См. раздел 1.3.3 [Зарезервированные имена].

Значения кода ошибки – это различные положительные целые числа, с одним исключением: EWOULDBLOCK и EAGAIN – имеют одинаковый код. Так как значения отличны, Вы можете использовать их как метки в утверждении выбора; только не используйте, и EWOULDBLOCK и EAGAIN. Ваша программа не должна иметь никаких сомнений относительно специфических значений этих символических констант.

Значение errno не обязательно должно соответствовать одной из этих макрокоманд, так как некоторые библиотечные функции могут возвращать другие их собственные коды ошибки для других ситуаций. Единственные значения, которые будут важны для специфической библиотечной функции это списки кодов ошибок для этой функции.

На не-GNU системах, почти любой системный вызов может возвратить EFAULT, если как параметр задан недопустимый указатель. Так как это могло случиться только в результате ошибки в вашей программе, и так как этого не будет в системе GNU, мы сэкономили место не упоминая EFAULT в описаниях индивидуальных функций.

2.2 Коды ошибок

Макрокоманды кода ошибки определены в заголовочном файле "errno.h". Каждая из них преобразуется в константное целое значение. Некоторые из этих ошибок не могут произойти в системе GNU, но они могут происходить при использовании библиотеки GNU в других системах.

int EPERM (макрос)

Не разрешенная операция; только владелец файла (или другого обьекта) или процессы со специальными привилегиями могут выполнять эту операцию.

int ENOENT (макрос)

Нет такого файла или каталога. Это – ошибка типа "файл не существует" для обычных файлов, которые вызваны в контекстах, где они, как ожидается, уже существуют.

int ESRCH (макрос)

Нет процесса соответствующего заданному.

int EINTR (макрос)

Прерванное обращение к функции; асинхронный сигнал предотвратил завершение обращения. Когда это случается, Вы должны попробовать снова вызвать функцию.

Вы можете выбрать показ резюме функций после сигнала EINTR; см. раздел 21.5 [Прерванные Примитивы] .

int EIO (макрос)

Ошибка ввода-вывода, обычно используется для ошибок физического чтения или записи.

int ENXIO (макрос)

Нет такого устройства или адреса. Система попробовала использовать устройство, указанное файлом, который Вы определили, и не смогло найти это устройство. Это может означать, что файл устройства был установлен неправильно, или физическое устройство отсутствует или не правильно присоединено к компьютеру.

int E2BIG (макрос)

Список параметров слишком длинный; используется, когда параметры, переданные одной из функций (см. раздел 23.5 [Выполнение файла]) занимают слишком много пространства памяти. Это условие никогда не возникает в системе GNU.

int ENOEXEC (макрос)

Недопустимый формат исполняемого файла. Это условие обнарживается запускаемыми функциями; см. раздел 23.5 [Выполнение файла].

int EBADF (макрос)

Плохой описатель файла; например, ввод - вывод на описателе, который был закрыт или чтение из описателя, открытого только для записи (или наоборот).

int ECHILD (макрос)

Не имеется никаких порожденнных процессов. Эта ошибка случается при операциях, которые, как предполагается, управляют порожденнными процессами, когда не имеется ни каких процессов, для управления.

int EDEADLK (макрос)

Тупик; распределение ресурсов системы оценено как ситуация тупика. Система не гарантирует, что она будет обращать внимание на все такие ситуации. Эта ошибка означает, что Вам повезло; система могла зависнуть. См. раздел 8.11 [Блокировки файла].

int ENOMEM (макрос)

Нет доступной памяти. Система не может распределять виртуальную память, потому она полна.

int EACCES (макрос)

Отклоненное право; права файла не позволяют предпринятую операцию.

int EFAULT (макрос)

Плохой адрес; был обнаружен недопустимый указатель.

int ENOTBLK (макрос)

Не специальный файл, был дан в ситуации, которая требует блок файов. Например, при попытке установить обычный файл как файловую систему в UNIX дает эту ошибку.

int EBUSY (макрос)

Ресурсы заняты; ресурс системы, который не может быть разделен уже используется. Например, если Вы пробуете удалить файл, который является корнем установленной в настоящее время файловой системы, Вы получаете эту ошибку.

int EEXIST (макрос)

Файл существует; существующий файл был определен в контексте, где имеет смысл определять только новый файл.

int EXDEV (макрос)

Была обнаружена попытка, сделать неподходящую компоновку файловой системы. Это случается не только, когда Вы используете связи (см. раздел 9.3 [Сложные связи]) но также, когда Вы переименовываете файл (см. раздел 9.6 [Переименование файлов]).

int ENODEV (макрос)

Функции, которая ожидает специфический тип устройства, был дан неправильный тип устройства.

int ENOTDIR (макрос)

Был определен файл, а не каталог, когда требуется каталог.

int EISDIR (макрос)

Указан каталог; попытка открыть каталог для записи дает эту ошибку.

int EINVAL (макрос)

Недопустимый параметр. Используется, чтобы указать различные виды проблем с указанием неправильного параметра для библиотечной функции.

int EMFILE (макрос)

Текущий процесс имеет слишком много открытых файлов и не может открыть больше. Двойные описатели приводят к этому ограничению.

int ENFILE (макрос)

Имеется слишком много различных открытых экземпляров файла во всей системе. Обратите внимание, что любое число связанных каналов считается только одним открытым экземпляром файла; см. раздел 8.5.1 [Связанные каналы]. Эта ошибка никогда не происходит в системе GNU.

int ENOTTY (макрос)

Несоответствующая операция управления ввода – вывода, типа попытки устанавливать режимы терминала в обычном файле.

int ETXTBSY (макрос)

Попытка выполнить файл, который является в настоящее время открытым для записи, или записи в файл который в настоящее время выполняется. Это не является ошибкой в системе GNU; текст по мере необходимости копируется.

int EFBIG (макрос)

Файл слишком большой; размер файла больше чем позволено системой.

int ENOSPC (макрос)

Нет места на устройстве; операция записи в файл потерпела неудачу, потому что диск полон.

int ESPIPE (макрос)

Недопустимая операция установки.

int EROFS (макрос)

Была сделана попытка изменить что - нибудь в файловой системе только для чтения.

int EMLINK (макрос)

Слишком много связей; число связей одиночного файла слишком велико. Переименование может вызывать эту ошибку, если переименовываемый файл уже имеет максимальное число связей (см. раздел 9.6 [Переименование файлов]).

int EPIPE (макрос)

Разрушенный канал; не имеется процесса читающего с другого конца канала. Каждая библиотечная функция, которая возвращает этот код ошибки также генерирует сигнал SIGPIPE; если этот сигнал не обработан или не блокирован, то он завершает программу . Таким образом, ваша программа фактически не будет никогда видеть EPIPE, если она не обработала или не блокировала SIGPIPE.

int EDOM (макрос)

Ошибка области; использование математических функций, когда значение параметра не относится к области над которой функция определена.

int ERANGE (макрос)

Ошибка диапазона; использование математических функций, когда значение результата не представимо из-за переполнения.

int EAGAIN (макрос)

Ресурс временно недоступен; обращение может работать, если Вы пробуете позже.

int EWOULDBLOCK (макрос)

Операция, которая бы была блокировала предпринята на объекте, который имеет выбранный режим не-блокирования.

Примечание относительно переносимости: В 4.4BSD и GNU, EWOULDBLOCK и EAGAIN - совпадают. Более ранние версии BSD (см. раздел 1.2.3 [Berkeley UNIX]) имеют два

различных кода, и используют EWOULDBLOCK, чтобы указать операцию ввода-вывода, которая блокировала бы объект с набором режимов неблокирования, а EAGAIN для других видов ошибок.

int EINPROGRESS (макрос)

Операция, которая не может завершаться немедленно, была инициализирована в объекте, который имеет выбранный режим неблокирования. Некоторые функции, которые должны всегда блокировать (типа, connect ; см., раздел 11.8.1 [Соединение]) никогда не возвращает EWOULDBLOCK. Взамен, они возвращают EINPROGRESS, чтобы указать, что операция начата и займет некоторое время. Попытайтесь управлять объектом прежде, чем обращение завершится возвратив EALREADY.

int EALREADY (макрос)

Операция уже выполняется в объекте, который имеет выбранный режим неблокирования.

int ENOTSOCK (макрос)

Был определен файл,а не гнездо, когда требуется гнездо.

int EDESTADDRREQ (макрос)

Нет был обеспечен адрес адресата для операции гнезда.

int EMSGSIZE (макрос)

Размер сообщения, посланного на гнездо был больше чем обеспечиваемый максимальный размер.

int EPROTOTYPE (макрос)

Тип гнезда не поддерживает запрашиваемый протокол связи.

int ENOPROTOOPT (макрос)

Вы определили опцию гнезда, которая не имеет смысла для специфического протокола, используемого гнездом. См. раздел 11.11 [Опции гнезда].

int EPROTONOSUPPORT (макрос)

Область гнезда не поддерживает запрашиваемый протокол связи (возможно, потому что запрашиваемый протокол полностью недопустим.) См. раздел 11.7.1 [Создание гнезда].

int ESOCKTNOSUPPORT (макрос)

Тип гнезда не установлен.

int EOPNOTSUPP (макрос)

Операция, которую Вы запросили, не обеспечивается. Некоторые функции гнезда не имеют смысла для всех типов гнезд, а другие не имеют права выполнения для всех протоколов связи.

int EPFNOSUPPORT (макрос)

Семейство протоколов связи гнезда, которое Вы запросили, не обеспечивается.

int EAFNOSUPPORT (макрос)

Семейство адресов, заданное для гнезда несогласованно с протоколом, используемым на гнезде. См. Главу 11 [Гнезда].

int EADDRINUSE (макрос)

Запрашиваемый адрес гнезда - уже используется. См. раздел 11.3 [Адреса гнезда].

int EADDRNOTAVAIL (макрос)

Запрашиваемый адрес гнезда не доступен; например, Вы попробовали дать гнезду имя, которое не соответствует местному главному имени. См. раздел 11.3 [Адрес Гнезда].

int ENETDOWN (макрос)

Операция с гнездом потерпела неудачу, потому что нет сети.

int ENETUNREACH (макрос)

Операция гнезда потерпела неудачу, потому что подсеть, содержащая главную ЭВМ была недоступна.

int ENETRESET (макрос)

Сетевое соединение было сброшено, потому что отдаленная главная ЭВМ умерла.

int ECONNABORTED (макрос)

Сетевое соединение было прервано локально.

int ECONNRESET (макрос)

Сетевое соединение было закрыто по внешним причинам контроля над местной главной ЭВМ, например из-за неисправимого нарушения протокола.

int ENOBUFS (макрос)

Буфера ядра для операций ввода - вывода занят весь.

int EISCONN (макрос)

Вы пытаетесь соединить гнездо, которое уже соединено. См. раздел 11.8.1 [Соединение].

int ENOTCONN (макрос)

Гнездо не соединено с чем - нибудь. Вы получаете эту ошибку, когда Вы пробуете передавать данные на гнездо, без первого определения адресата для данных.

int ESHUTDOWN (макрос)

Гнездо уже было закрыто.

int ETIMEDOUT (макрос)

Операция гнезда с заданной блокировкой по времени не получила никакого ответа в течение периода блокировки по времени.

int ECONNREFUSED (макрос)

Отдаленная главная ЭВМ отказала в сетевом соединении (обычно из-за того, что не запущено запрашиваемое обслуживание).

int ELOOP (макрос)

При поиске имени файла столкнулись со слишком многими уровнями символических связей. Часто это указывает на цикл символических связей.

int ENAMETOOLONG (макрос)

Имя файла слишком длинное (больше чем PATH_MAX; см. раздел 27.6 [Ограничения для файлов]) или главное имя слишком длинное (в gethostname или sethostname; см. раздел 26.1 [Главная идентификация]).

int EHOSTDOWN (макрос)

Отдаленная главная ЭВМ для запрашиваемого сетевого соединения не реагирует.

int EHOSTUNREACH (макрос)

Отдаленная главная ЭВМ для запрашиваемого сетевого соединения не доступна.

int ENOTEMPTY (макрос)

Каталог, не пустой, а ожидался пустой каталог. Обычно эта ошибка происходит, когда Вы пробуете удалять каталог.

int EUSERS (макрос)

Файловая система спутана, потому что имеется слишком много пользователей.

int EDQUOT

Пользовательское дисковое пространство превышено.

int ESTALE (макрос)

Просроченная NFS программа обработки файла. Это указывает на внутренний беспорядок в NFS системе, который появляется из-за перестановок файловой системы на главной ЭВМ станции. Восстановление этого условия обычно требует переустановки файловой системы NFS на местной главной ЭВМ.

int EREMOTE (макрос)

Была сделана попытка NFS-подсоединения удаленной файловой системой с именем файла, которое уже определяет установленный файл NFS. (Эта ошибка воникает на некоторых операционных системах, но мы думаем что это будет работать правильно на системе GNU, делающей этот код ошибки невозможным.)

int ENOLCK (макрос)

Нет доступной блокировки. Это используется средствами закрытия файла; см. раздел 8.11 [Блокировки файла]. Эта ошибка никогда не происходит в системе GNU.

int ENOSYS (макрос)

Функция не выполнена. Некоторые функции имеют команды или определяющие их опции, которые не могут обеспечиваться во всех реализациях, и это – ошибка, которую Вы получаете, если Вы запрашиваете то, что не обеспечивается.

int EBACKGROUND (макрос)

В системе GNU, станции, обеспечивающие протокол терминала возвращают эту ошибку для некоторых операций, когда вызывающий оператор не входит в группу приоритетного процесса терминала. Пользователи обычно не видят эту ошибку, потому что функции типа чтения и записи транслируют ее в SIGTTIN или SIGTTOU сигнал. См. Главу 24 [Управление заданиями], для уточнения информации относительно групп процессов и этих сигналов.

int ED(макрос)

Опытный пользователь будет знать, что неправильно.

int EGREGIOUS (макрос)

Что Вы делаете?!!!

int EIEIO (макрос)

Идите домой и выпейте стакан теплого молока.

int EGRATUITOUS (макрос)

Этот код ошибки не имеет никакой цели.

2.3 Сообщения об ошибках

Библиотека имеет функции и переменные, разработанные, чтобы облегчить для вашей программы вывод информативных сообщений об ошибках в обычном формате. Функции strerror и perror дают Вам стандартное сообщение об ошибках для данного кода ошибки; переменная program_invocation_short_name дает Вам удобный доступ к имени программы, которая столкнулась с ошибкой.

```
char * strerror ( int errnum) (функция)
```

функция strerror отображает код ошибки (см. раздел 2.1 [Прверка Ошибок]) заданный параметром errnum в описательную строку сообщения об ошибках. Значение возврата – указатель на эту строку.

Значение errnum обычно исходит из переменной errno.

Вы не должны изменять строку, возвращаемую strerror. Также, если Вы делаете последующие обращения к strerror, новая строка могла быть записана поверх старой. (Но гарантируется, что никакая библиотечная функция не вызовет strerror за вашей спиной.) Функция strerror объявлена в "string.h".

```
void perror (const char * message) (функция)
```

Эта функция печатает сообщение об ошибках в поток stderr; см. раздел 7.2 [Стандартные Потоки].

Если Вы вызываете perror с сообщением, которое является или нулевым указателем или пустой строкой, perror печатает сообщение об ошибках, соответствуя errno, добавляя конечный символ перевода строки.

Если Вы обеспечиваете не-нулевой параметр сообщения, то perror начинает вывод с этой строки. Она добавляет двоеточие и пробел, чтобы отделить сообщение от строки ошибки, соответствующей errno.

Функция perror объявлена в "stdio.h".

strerror и perror производят точно то же самое сообщение для любого данного кода ошибки; точный текст изменяется от системы до системы. На системе GNU, сообщения довольно коротки; не имеются никаких многострочных сообщений или вложенных символов перевода строки. Каждое сообщение об ошибках начинается заголовочной буквой и не включает ни какой пунктуации завершения.

Примечание относительно совместимости: функция strerror - новая особенность ANSI C, многие более старые C системы не поддерживают эту функцию.

Множество программ, которые не читают ввод с терминала, разработаны, чтобы выйти, если любой системный вызов терпит неудачу. В соответствии с соглашением, сообщение об ошибках из такой программы должно начаться с имени программы. Вы можете найти это имя в переменной program_invocation_short_name; полное имя файла сохранено в переменной program_invocation_name:

```
char * program_invocation_name (переменная)
```

Значение этой переменной - имя, которое использовалось, чтобы вызвать программу, выполняющуюся в текущем процессе. Аналогично argv [0]. Обратите внимание, что это не обязательно какое-то полезное имя файла; часто она не содержит никаких имен. См. раздел 22.1 [Аргументы программы].

```
char * program_invocation_short_name (переменная)
```

Значение этой переменной - имя, которое использовалось, чтобы вызвать программу, выполняющуюся в текущем процессе, без имен каталогов. (То есть то-же что и в program_invocation_name минус все до последней наклонной черты вправо, если что-то есть в наличии.)

Библиотечные код инициализации устанавливает обе из этих переменных перед вызовом main.

Примечание относительно переносимости: Эти две переменные расширения GNU. Если Вы хотите, чтобы ваша программа работала с библиотеками, не относящимися к GNU, Вы должны сохранить значение argv [0] в main (основной программе), и удалить имена каталогов самостоятельно. Мы добавили эти расширения, чтобы сделать возможной написание замкнутых сообщений об ошибках подпрограммы, которые не требуют никакого явного сотрудничества с основной программой.

Имеется пример, показывающий, как обработать отказ открывать файл. Функция open_ sesame пробует открывать указанный файл для чтения и возвращает поток. Библиотечная функция

fopen возвращает нулевой указатель, если она не может открыть файл по некоторым причинам. В той ситуации, open_sesame создает соответствующее сообщение об ошибках, используя функцию strerror, и завершает программу. Если мы хотим сделать другие вызовы из библиотек перед передачей кода ошибки к strerror, мы должны сохранить его в местной переменной, потому что те другие библиотечные функции могут записывать поверх errno.

```
#include <errno.h>
#include <stdio.h>
#include < stdlib.h >
#include <string.h>
FILE * open_sesame (char *name)
 {
   FILE *stream;
   errno = 0;
   stream = fopen (name, "r");
   if (stream == NULL)
    {
      fprintf (stderr, "%s: Couldn't open file %s; %s\n",
        program_invocation_short_name, name, strerror (errno));
      exit (EXIT_FAILURE);
    }
  else return stream;
}
```

Вперед Назад Содержание