КОНСТРУИРОВАНИЕ ПРОГРАММ И ЯЗЫКИ ПРОГРАМИРОВАНИЯ

Лекция № 24.2 — Сигналы

Преподаватель: Поденок Леонид Петрович, 505а-5

+375 17 293 8039 (505а-5)
+375 17 320 7402 (ОИПИ НАНБ)
prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by

Кафедра ЭВМ, 2021

Оглавление

Сигналы

С точки зрения пользователя получение процессом сигнала выглядит как возникновение прерывания.

Процесс прекращает свое регулярное исполнение, и управление передается механизму обработки сигнала.

По окончании обработки сигнала процесс может возобновить регулярное исполнение. Типы сигналов и способы их возникновения в системе жестко регламентированы.

Сигналы обычно представляют собой ситуации, когда программа либо получила запрос на завершение, либо произошла неисправимая ошибка, поэтому обработка сигнала позволяет либо выполнить операции очистки перед завершением, либо попытаться каким-либо образом восстановиться после ошибки.

Процесс может получить сигнал от:

1) аппаратной части (при возникновении исключительной ситуации);
2) другого процесса, выполнившего системный вызов передачи сигнала;
3) операционной системы (при наступлении некоторых событий);
4) терминала (при нажатии определенной комбинации клавиш);
5) системы управления заданиями при запросе пользователя на завершение процесса.

Существует три варианта реакции процесса на сигнал:

1) принудительно проигнорировать сигнал;

2) произвести обработку по умолчанию (проигнорировать, остановить процесс (перевести в состояние ожидания до получения другого специального сигнала), либо завершить работу с образование core файла или без оного);

3) выполнить обработку сигнала, специфицированную пользователем.

Изменение реакции процесса на сигнал

Изменить реакцию процесса на сигнал можно с помощью специальных системных вызовов.

Реакция на некоторые сигналы не допускает изменения и они могут быть обработаны только по умолчанию.

Так, например, сигнал **`SIGKILL`** обрабатывается только по умолчанию и всегда приводит к завершению процесса.

Концепции генерации сигналов

События, которые генерируют сигналы, делятся на три основные категории:

- ошибки (errors);
- внешние события (external events);
- явные запросы (explicit requests).

Ошибка означает, что программа сделала что-то недопустимое и не может продолжить выполнение.

Не все виды ошибок генерируют сигналы — на самом деле, большинство из них ничего не генерирует. Например, открытие несуществующего файла является ошибкой, но не вызывает генерацию сигнала. Вместо этого функция/вызов **`fopen`** возвращает -1.

Как правило, об ошибках, которые обязательно связаны с определенными библиотечными функциями, сообщается путем возврата значения, указывающего на ошибку, либо с помощью средств языка программирования, например, программных исключений в C++.

Ошибки, которые вызывают сигналы, могут возникать в любом месте программы, а не только при вызовах библиотеки. К ним относятся деление на ноль и недопустимые адреса памяти.

Внешнее событие обычно связано с вводом-выводом (I/O) или другими процессами. К ним относятся поступление ввода, завершение вывода, истечение таймера, завершение дочернего процесса.

Явный запрос означает использование библиотечной функции, такой как **`raise()`**, специально предназначенной для генерации сигнала.

Сигналы могут генерироваться «*синхронно*» или «*асинхронно*».

Синхронный сигнал относится к определенному действию в программе и доставляется (если не заблокирован) во время этого действия.

Большинство ошибок генерируют сигналы синхронно, как и явные запросы процесса на генерацию сигнала для того же процесса.

На некоторых машинах определенные виды аппаратных ошибок (обычно исключения с плавающей запятой) не выдаются полностью синхронно, тем не менее, они могут появиться через несколько инструкций.

Асинхронные сигналы генерируются событиями, не зависящими от процесса, который их получает.

Эти сигналы поступают в непредсказуемое время во время выполнения. Внешние события генерируют сигналы асинхронно, как и явные запросы, которые вызываются в отношении какого-либо другого процесса.

Обычно определенный тип сигнала либо синхронный, либо обычно асинхронный. Например, сигналы об ошибках обычно синхронны, потому что ошибки генерируют сигналы синхронно.

Но любой тип сигнала может быть сгенерирован синхронно или асинхронно с помощью явного запроса.

Как доставляются сигналы

Когда сигнал генерируется, он становится «ожидающим, незавершенным». Обычно он остается в состоянии «ожитдания» в течение короткого периода времени, а затем «доставляется» тому процессу, которому был послан.

Однако, если такой сигнал в данный момент времени «заблокирован», он может оставаться в состоянии ожидания неопределенно долго — до тех пор, пока такие сигналы не будут «разблокированы». После разблокировки он будет доставлен процессу немедленно.

Как только сигнал доставляется, сразу или после долгой задержки, выполняется «указанное действие» для этого сигнала.

Для определенных сигналов, таких как «**SIGKILL**» и «**SIGSTOP**», действие фиксировано, но для большинства сигналов у программы есть выбор, что делать при получении данного вида сигнала:

- игнорировать сигнал;
- указать «функцию-обработчик»;
- принять «действие по умолчанию».

Программа указывает свой выбор с помощью функции **signal()**.

Иногда говорят, что обработчик «ловит» сигнал. Во время работы обработчика сигнал данного типа обычно блокируется.

Если заданное действие для определенного типа сигнала заключается в его игнорировании, то любой такой сигнал, который генерируется, немедленно отбрасывается.

Это происходит, даже если сигнал в это время заблокирован. Сигнал, отброшенный таким образом, никогда не будет доставлен, даже если программа впоследствии укажет другое действие для этого типа сигнала, а затем разблокирует его.

Если поступает сигнал, который программа и не обрабатывает и не игнорирует, для сигнала выполняется «действие по умолчанию».

Каждый вид сигнала имеет собственное действие по умолчанию.

Для большинства сигналов действие по умолчанию — завершить процесс.

Для определенных типов сигналов, которые представляют «безобидные» события, действие по умолчанию — ничего не делать.

<csignal> (signal.h) — библиотека C для обработки сигналов

Типы сигналов (макро)

Сигналы программных ошибок

Сигналы ошибок программы используются для сообщения о серьезных ошибках программы.

Они генерируются при обнаружении серьезной программной ошибки операционной системой или самим компьютером. В общем, все эти сигналы указывают на то, что ваша программа каким-то образом серьезно нарушена, и обычно нет возможности продолжить вычисления, в которых возникла ошибка.

Некоторые программы обрабатывают сигналы программных ошибок, чтобы привести в порядок среду перед завершением работы. Например, программы, которые отключают эхо ввода терминала, чтобы снова включить эхо, должны обрабатывать сигналы программных ошибок.

Прекращение работы — это разумное завершение в случае программной ошибки в большинстве программ. Однако некоторым программным системам, таким, например, как Lisp, которые могут загружать скомпилированные пользовательские программы, может потребоваться продолжать выполнение, даже если пользовательская программа вызывает ошибку.

У этих программ есть обработчики, которые используют **`longjmp()`** для возврата управления на командный уровень.

Действие по умолчанию для всех этих сигналов — завершить процесс. Если заблокировать или проигнорировать эти сигналы, либо установить для них обработчики, которые будут возвращатся нормально, как будто ничего не произошло, программа сломается как только такие сигналы появятся. Конечно, если только они не будут сгенерированы с помощью **raise()** вместо реальной ошибки.

Когда один из таких сигналов программной ошибки завершает процесс, он также записывает «файл дампа памяти», в котором записывается состояние процесса на момент завершения. Файл дампа ядра называется «core» и записывается в тот каталог, который в данный момент используется в процессе. В некоторых системах можно указать имя файла для дампа ядра с помощью переменной среды **COREFILE**. Файлы дампа ядра предназначены для того, чтобы можно было исследовать их с помощью отладчика и выяснить, что вызвало ошибку.

SIGSEGV — (Signal Segmentation Violation) недопустимый доступ к памяти.

Этот сигнал генерируется, когда программа пытается читать или писать за пределами выделенной для нее памяти или записывать в память, доступную только для чтения.

Фактически, сигналы возникают только тогда, когда программа выходит достаточно далеко за пределы, чтобы быть обнаруженной механизмом защиты памяти системы.

Название является аббревиатурой от «segmentation violation — нарушение сегментации».

Общие способы получения условия «**SIGSEGV**» включают разыменование нулевого или неинициализированного указателя или когда используется указатель для прохода по массиву, но нет возможности проверить конец массива.

SIGABRT — (Signal Abort) аномальное завершение. Этот сигнал указывает на ошибку, обнаруженную самой программой и сообщенную с помощью вызова **abort()**. Программа прерывается.

SIGFPE — (Signal Floating-Point Exception) Ошибочная арифметическая операция.

Этот сигнал сообщает о фатальной арифметической ошибке. несмотря на слово «floating» он фактически охватывает все арифметические ошибки, включая деление на ноль и переполнение.

Если программа хранит целочисленные данные в месте, которое затем используется в операции с плавающей запятой, это часто вызывает исключение «недопустимая операция», поскольку процессор не может распознать данные как число с плавающей запятой.

Стандарт IEEE для двоичной арифметики с плавающей запятой (ANSI / IEEE Std 754-2017) определяет различные исключения с плавающей запятой и требует, чтобы соответствующие компьютерные системы сообщали об их возникновении. Однако этот стандарт не определяет, как сообщается об исключениях или какие виды обработки и контроля операционная система может предложить программисту.

Фактические исключения с плавающей запятой — сложная тема, потому что существует много типов исключений с слегка разными значениями, и сигнал **SIGFPE** их не различает.

SIGILL — (Signal Illegal Instruction) попытка вызова недопустимой функции, например недопустимая инструкция.

Обычно это означает, что программа пытается выполнить мусор или привилегированную инструкцию. Поскольку компилятор C генерирует только допустимые инструкции, **SIGILL** обычно указывает, что исполняемый файл поврежден или что программа пытается выполнить данные.

Некоторые распространенные причины — передать недопустимый объект в случае, когда ожидался указатель на функцию, или выполнить запись за конец автоматического массива, повредить другие данные в стеке, например, адрес возврата в кадре стека.

Аналогичные проблемы возникают с указателями на автоматические переменные.

SIGILL также может быть сгенерирован при переполнении стека или когда в системе возникают проблемы с запуском обработчика сигнала.

Сигналы завершения

Сигналы завершения используются, чтобы так или иначе сказать процессу о завершении. У них разные имена, потому что они используются для немного разных целей, и программы могут обращаться с ними по-разному.

Причина обработки этих сигналов обычно заключается в том, чтобы программа могла привести себя в порядок перед фактическим завершением. Например, сохранить информацию о состоянии, удалить временные файлы или восстановить предыдущие режимы терминала.

Такой обработчик должен заканчиваться указанием действия по умолчанию для пришедшего сигнала и его повторным вызовом — это приведет к тому, что программа завершится с этим сигналом, как если бы у нее не было обработчика.

Действие по умолчанию для всех этих сигналов — завершить процесс.

SIGINT — (Signal Interrupt) Интерактивный сигнал внимания. Обычно отправляется, когда пользователь вводит символ **INTR** (обычно «Ctrl-C»).

SIGTERM — (Signal Terminate) в программу отправлен запрос на завершение. Это общий сигнал, используемый для завершения программы. Сигнал можно блокировать, обрабатывать и игнорировать. Это нормальный способ вежливо попросить программу завершить работу.

Команда оболочки **kill** по умолчанию генерирует **SIGTERM**.

Конкретная реализация библиотеки может предоставлять дополнительные макроконстанты значений сигнала. Не все окружения должны генерировать автоматические сигналы, даже в конкретных случаях, описанных выше, однако все должны доставлять сигналы, сгенерированные явным вызовом функции **raise()**.

Типы обработки и возрата

SIG_DFL — Default handling — Обработка по умолчанию — сигнал обрабатывается действием по умолчанию для этого конкретного сигнала.

SIG_IGN — (Ignore Signal) — Игнорировать сигнал

SIG_ERR — специальное возвращаемое значение, указывающее на сбой.

Функции

signal() — установить функцию обратного вызова для обработки сигнала

rise() — генерирует сигнал

Поддержка генерации сигналов не является обязательной, с одной стороны, но и не ограничивает типы сигналов, если они генерируются, с другой — все зависит от реализации. Многие среды исполнения генерируют не только представленные ниже, но и многие другие специфические сигналы. Однако, в любом случае все сигналы, сгенерированные явным образом при вызове функции **raise()**, доставляются соответствующему обработчику сигналов.

Тип данных

sig_atomic_t — целочисленный тип объекта, к которому можно получить доступ как к атомарной сущности, даже при наличии асинхронных сигналов.

signal() — установить функцию обратного вызова для обработки сигнала

```
#include <csignal> // #include <signal.h>

void (*signal(int sig, void (*func)(int)))(int);
```

Задает способ обработки сигналов с номером сигнала, указанным в **sig**.

Параметр **func** определяет один из трех способов обработки сигнала программой:

SIG_DFL — обработка по умолчанию — сигнал обрабатывается действием по умолчанию для этого конкретного сигнала.

SIG_IGN — игнорировать сигнал — сигнал игнорируется, и выполнение кода будет продолжено, даже если оно не имеет смысла.

Обработчик функции — определяется конкретная функция для обработки сигнала, которая должна соответствовать следующему прототипу с типом связывания C.

```
void handler_function(int parameter);
```

При запуске программы для каждого из поддерживаемых сигналов в качестве поведения обработки сигналов по умолчанию устанавливается либо **SIG_DFL**,либо **SIG_IGN**.

Обработчик должен завершаться указанием действия по умолчанию для произошедшего сигнала и его повторным вызовом — это приведет к тому, что программа завершится с этим сигналом, как если бы у нее не было обработчика.

Возвращаемое значение

```
void (*signal(int sig, void (*func)(int)))(int);
```

Тип возвращаемого значения такой же, как тип параметра **func**.

Если запрос выполнен успешно, функция возвращает указатель на конкретную функцию-обработчик, которая отвечала за обработку этого сигнала перед вызовом, если таковая имеется.

В противном случае возвращается либо **SIG_DFL**, либо **SIG_IGN**, если перед вызовом сигнал, соответственно, обрабатывался обработчиком по умолчанию или игнорировался.

Если функции не удалось зарегистрировать новую процедуру обработки сигнала, она возвращает **SIG_ERR**, и для **errno** может быть установлено положительное значение.

Вызов этой функции в многопоточной программе вызывает неопределенное поведение.

Функция никогда не генерирует исключений.

Если сигнал возникает в результате вызова функции **abort()** или **raise()**, обработчик сигнала не должен вызывать функцию **raise()**.

Сигналы и возможности ОС

Системы BSD предоставляют обработчику **SIGFPE** дополнительный аргумент, который различает различные причины исключения.

Чтобы получить доступ к этому аргументу, следует определить обработчик для приема двух аргументов, что означает, что для того, чтобы установить такой обработчик, необходимо привести его к типу функции с одним аргументом.

Библиотека GNU C предоставляет этот дополнительный аргумент, но это имеет смысл только в операционных системах, которые предоставляют информацию (системы BSD и системы GNU).

FPE_INTOVF_TRAP — Целочисленное переполнение (невозможно в программе на C, если не включен перехват переполнения в зависимости от оборудования).

FPE_INTDIV_TRAP — Целочисленное деление на ноль.

FPE_SUBRNG_TRAP — Нижний индекс (это то, что программы C никогда не проверяют).

FPE_FLTOVF_TRAP — Ловушка переполнения с плавающей запятой.

FPE_FLTDIV_TRAP — Плавающее/десятичное деление на ноль.

FPE_FLTUND_TRAP — Ловушка недопереполнения с плавающей запятой (отлов обычно не включен).

FPE_DECOVF_TRAP — Десятичная ловушка переполнения.

Надо отметить, что только несколько архитектур имеют десятичную арифметику, и C никогда ее не использует.

Пример

```
#include <stdio.h>      /* printf */
#include <signal.h>     /* signal, raise, sig_atomic_t */

sig_atomic_t signaled = 0;

void my_handler(int param) {
  signaled = 1;
}

int main () {

  void (*prev_handler)(int);

  prev_handler = signal(SIGINT, my_handler);

  /* ... */
  raise(SIGINT);
  /* ... */

  printf("signaled is %d.\n",signaled);

  return 0;
}
```

raise — генеривать сигнал

```
#include <csignal> // #include <signal.h>

int raise(int sig);
```

Посылает сигнал **sig** текущей исполняющейся программе.

Сигнал обрабатывается, как указано в функции **signal()**.

Возвращаемое значение
В случае успеха возвращает ноль, в противном случае — значение, отличное от нуля.

Гонки данных
Одновременный вызов этой функции безопасен и не вызывает гонок данных.

Исключения (C ++)
Если для обработки сгенерированного сигнала не были определены обработчики функций, **raise()** никогда не генерирует исключения.
В противном случае поведение зависит от конкретной реализации библиотеки.