OS Lab 21

Status	ready
checkbox	☐
class	OS
due date	@Mar 31, 2021

Task

21. Пищалка

Напишите программу, которая входит в бесконечный цикл и издает звуковой сигнал на вашем терминале каждый раз,
когда вы вводите символ, на который у вас настроена посылка сигнала `SIGINT` (по умолчанию `CTRL-C`).
При получении `SIGQUIT`, она должна вывести сообщение, говорящее, сколько раз прозвучал сигнал, и завершиться.

Notes

Сигналы — это сервис, предоставляемый ядром системы и используемый, главным образом, для обработки ошибок и исключительных ситуаций. Сигналы могут быть посланы ядром или процессом процессу или группе процессов. Например, сигнал посылается когда нажимается клавиша `CTRL-C`, чтобы убить исполняющуюся программу. Также, сигнал посылается, когда программа пытается поделить на ноль. В обоих названных случаях, сигналы посылаются ядром. Сигналы можно рассматривать как программный аналог аппаратных прерываний. Они могут быть обработаны получающим процессом.

Процесс может определить действия, которые должны быть выполнены при получении сигнала. Такие действия называются реакцией на сигнал. Каждый сигнал имеет реакцию по умолчанию, которая выполняется, если для этого сигнала в этом процессе не были явно определены какие-то другие действия. Для большинства сигналов реакция по умолчанию — завершение процесса. Процесс имеет возможности либо проигнорировать сигнал, либо исполнить функцию обработки при его получении. Когда для определённого сигнала установлена реакция, она не меняется, пока не будет явным образом установлена другая реакция.
Каждый процесс имеет **сигнальную маску,** определяющую множество сигналов, получение которых заблокировано. Маска сигналов процесса наследуется от родительского процесса.

Сигналы — это механизм передачи сообщений между процессами и от ядра к процессам. Они оповещают процесс о том, что произошло определённое событие. Говорят, что сигнал генерируется для процесса (или посылается процессу), когда в первый раз происходит событие, связанное с этим сигналом. Тип сигнала показывает, какое событие произошло.

ТИПЫ СИГНАЛОВ

сигнал	реакция	событие
SIGHUP	exit	обрыв линии (см. termio(7))
SIGINT	exit	прерывание (см. termio(7))
SIGQUIT	core	завершение (см. termio(7))
SIGILL	core	неправильная инструкция
SIGTRAP	core	прерывание трассировки
SIGABORT	core	аборт
SIGEMT	core	команда EMT (программное прерывание)
SIGFPE	core	арифметическая особая ситуация
SIGKILL	exit	принудительное завершение ("убийство")
SIGBUS	core	ошибка шины
SIGSEGV	core	нарушение сегментации
SIGPIPE	exit	разрыв конвейера

Все имена сигналов определены как константы с положительными числовыми значениями (номерами сигналов) в заголовочном файле `<signal.h>` (Solaris 10 — в файле `<sys/iso/signal_iso.h>`)

Сигналы от `SIGHUP` до `SIGTERM` генерируются ядром, когда возникает соответствующее событие. Сигналы SIGUSR1 и SIGUSR2 могут использоваться программистами для своих целей.

Использование клавиши INTR (`CTRL-C` по умолчанию), при нажатии которой на терминале генерируется сигнал `SIGINT`.

Ядро, обнаружив незаконное обращение к памяти в вашей программе, генерирует сигнал "нарушение сегментации" (segmentation violation) — `SIGSEGV`.

Аналогично, деление на ноль с плавающей точкой генерирует сигнал "особая ситуации при работе с плавающей точкой (floating point exception) `SIGFPE`. Этот же сигнал генерируется и при целочисленном делении на ноль.

Подробнее о типах:

- `SIGHUP` (hang up) — Посылается управляющему процессу (лидеру сеанса), связанному с управляющим терминалом, если обнаружен обрыв связи с терминалом. В далёкие времена когда подключали по модему, если соединение обрывалось, то нужно было

завершить сессию потому что иначе, следующий подключившийся получил бы вашу терминальную сессию, поэтому при разрывах соединения посылается этот сигнал и сессия завершается. Этот сигнал также генерируется в случае завершения лидера сеанса. В такой си-туации сигнал посылается всем процессам из группы процессов переднего плана.Нередко этот сигнал используется для извещения процессов-демонов *(глава 13 UNIX проф. программирование)* о необходимости перечитать конфигурационные файлы. Причина, по которой для этой цели выбирается именно сигнал `SIGHUP`, заключается в том, что если не по-слать этот сигнал явно, демоны никогда не примут его, поскольку у них нет управ-ляющего терминала.

- `SIGINT` — Генерируется драйвером терминала при нажатии клавиши прерывания (часто — DELETE или Control-C). Посылается всем процессам из группы процессов переднего плана. Этот сигнал часто используется для прерывания выполнения приложений, вышедших из-под контроля, особенно когда они начи-нают выводить ненужную информацию на экран.
- `SIGQUIT` — Генерируется драйвером терминала при вводе символа завершения (часто Control-\). Посылается всем процессам из группы процессов переднего плана. При этом происходит не только завершение группы процессов переднего плана (как в случае сигнала `SIGINT`), но и создание файла core.
- `SIGILL` — Указывает, что процесс выполнил недопустимую машинную инструкцию. Либо передали управление в какое то место, которое не является кодом, либо эту команду текущий процессор не поддерживает. Например у интелла, когда мат сопроцессор был не обязателен, то выполняя любую команду мат сопроцессора вы получали `SIGILL`
- `SIGTRAP` — Прерывание трассировки. Брейкпоинт в отладчике.
  "Забавно, но мы это проходить не будем" - Дмитрий Иртегов
- `SIGABRT` — Генерируется вызовом функции abort. Процесс завершается аварийно.
- `SIGFPE` — Свидетельствует об арифметической ошибке, такой как деление на 0 или переполнение числа с плавающей точкой.
- `SIGKILL` — Один из двух сигналов, которые нельзя перехватить или игнорировать в приложении. Дает возможность системному администратору уничтожить любой процесс. Шлётся только программно, всегда убивает процесс которому он послан.
- `SIGBUS` — нет на интеле, на некоторых процессорах посылается если читаете 32-битное слово указатель которого не делится на 4. (У спарков)
- `SIGSEGV` (*segmentation violation*) — Показывает, что процесс обратился к недопустимому адресу в памяти (что обычно служит признаком ошибки в программе, такой как разыменование пустого указателя).
- `SIGPIPE` — разрыв конвеера.
- `SIGALRM` — генерируется по истечении таймера, установленного функцией `alarm`. Также генерируется по истечении таймера, установленного функцией `setitimer(2)`. Поведение по умолчанию — exit, то есть если не обработать будильник то он вас убьет.
- `SIGTERM` — Сигнал завершения процесса, который посылается командой `kill(1)` по умолчанию. Так как его можно перехватить, обработка сигнала `SIGTERM` дает

В случае появления сигнала можно запросить ядро произвести одно из трех действий. Они называются *диспозициями* сигнала, или *действиями*, связанными с сигналом.

1. Игнорировать сигнал. Это действие возможно для большинства сигналов, но два сигнала, `SIGKILL` и `SIGSTOP`, нельзя игнорировать. Причина, почему эти два сигнала не могут быть проигнорированы, заключается в том, что ядру и суперпользователю необходима возможность завершить или остановить любой процесс. Кроме того, если проигнорировать некоторые из сигналов, возникающих в результате аппаратных ошибок (таких, как деление на 0 или попытка обращения к несуществующей памяти), поведение процесса может стать непредсказуемым.
2. Перехватить сигнал. Для этого нужно сообщить ядру адрес функции, которая будет обрабатывать сигнал. В этой функции можно предусмотреть действия по обработке условия, породившего сигнал. Например, создавая командный интерпретатор, можно предусмотреть в нем прерывание выполняемой команды, запущенной пользователем, и возврат к главному циклу, когда пользователь пошлет сигнал прерывания. Если пойман сигнал `SIGCHLD`, который означает завершение дочернего процесса, функция, перехватившая сигнал, может вызвать функцию `waitpid`, чтобы получить идентификатор дочернего процесса и код его завершения. Еще один пример: если процесс создает временные файлы, имеет смысл написать функцию обработки сигнала `SIGTERM` (сигнал завершения, посылаемый командой `kill` по умолчанию), которая будет удалять временные файлы. Имейте в виду, что сигналы `SIGKILL` и `SIGSTOP` нельзя перехватить.
3. Применить действие по умолчанию. Каждому сигналу поставлено в соответствие некоторое действие по умолчанию (перечислены в табл. 10.1). Заметьте, что для большинства сигналов действие по умолчанию заключается в завершении процесса.

SIGSET(3C)

The `signal()` and `sigset()` functions modify signal dispositions. The `sig` argument specifies the signal, which may be any signal except `SIGKILL` and `SIGSTOP.` The `disp` argument specifies the signal's disposition, which may be `SIG_DFL`, `SIG_IGN`, or the address of a signal handler. If `signal()` is used, `disp` is the address of a signal handler, and `sig` is not `SIGILL`, `SIGTRAP`, or `SIGPWR`, the system first sets the signal's disposition to `SIG_DFL` before executing the signal handler. If `sigset()` is used and `disp` is the address of a signal handler, the system adds `sig` to the calling process's signal mask before executing the signal handler; when the signal handler returns, the system restores the calling process's signal mask to its state prior to the delivery of the signal. In addition, if `sigset()` is used and `disp` is equal to `SIG_HOLD`, `sig` is added to the calling process's signal mask and the signal's disposition remains unchanged.

```
void (*signal(int sig, void (*disp)(int)))(int);
void (*sigset(int sig, void (*disp)(int)))(int);

ERRORS

       These functions fail if:

       EINTR
                 A signal was caught during the execution sigpause().


       EINVAL
                 The value of the sig argument is not a valid signal or
                 to SIGKILL or  SIGSTOP.
```

программам возможность корректно завершить работу, освободив занятые ресурсы (в противоположность сигналу `SIGKILL`, который нельзя перехватить или игнорировать).

- `SIGCHLD` — Когда процесс завершается или останавливается, родительскому процессу посылается сигнал `SIGCHLD`. По умолчанию этот сигнал игнорируется, но родительский процесс может перехватить его, если желает получать извещения об изменении состояния дочерних процессов. Функция, перехватывающая этот сигнал, обычно вызывает одну из функций семейства wait, чтобы получить иден-тификатор дочернего процесса и код завершения. В ранних версиях System V имелся похожий сигнал с именем `SIGCLD` (без H). Семантика этого сигнала отличалась от семантики других сигналов, и еще справоч-ное руководство SVR2 рекомендовало не использовать его в новых программах. (Как ни странно, в SVR3 и SVR4 из справочного руководства это предупреждение исчезло.) Приложения должны использовать сигнал `SIGCHLD`, но нужно знать, что многие версии UNIX определяют сигнал `SIGCLD`, идентичный сигналу `SIGCHLD`, для сохранения обратной совместимости. Если вам понадобится определить семанти-ку сигнала `SIGCLD` в своей системе, обратитесь к страницам справочного руководств-ства.
- `SIGPWR` — Реализация этого сигнала зависит от системы. В основном он используется в системах, снабженных источником бесперебойного питания (UPS). Обнаружив сбой в электросети, источник бесперебойного питания извещает об этом систему и принимает на себя обеспечение питания системы. Пока ничего более не предпринимается, так как система продолжает питаться от аккумуляторных бата-рей. Но когда напряжение в сети отсутствует продолжительное время и напряжение аккумуляторов падает ниже определенного уровня, программное обеспечение обычно извещается об этом повторно, и с этого момента у системы остается при-мерно 15–30 секунд, чтобы корректно завершить работу. В этот момент посылает-ся сигнал `SIGPWR`. В большинстве систем имеется процесс, который получает изве-щение о падении напряжения аккумуляторов и посылает сигнал `SIGPWR` процессу `init`, а `init` берет на себя заботу об остановке системы
- `SIGTSTP` — Этот сигнал приостановки генерируется драйвером терминала при вводе символа приостановки (часто Control-Z). Посылается всем процессам из группы процессов переднего плана
- `SIGSTOP` — Останавливает процесс. Похож на сигнал `SIGTSTP`, порождаемый драйвером терминала, но в отличие от него `SIGSTOP` нельзя перехватить или игнорировать.

Signal(2)

ИСПОЛЬЗОВАНИЕ

```
#include <signal.h>
typedef void (*handler_t)(int)
handler_t signal (int sig, handler_t handler);

handler_t sigset (int sig, handler_t handler);
```

ВОЗВРАЩАЕМОЕ ЗНАЧЕНИЕ

signal

успех - предыдущая реакция на сигнал

неуспех - SIG_ERR и errno установлена

sigset

успех - SIG_HOLD если сигнал заблокирован, иначе предыдущая реакция на этот сигнал

неуспех - SIG_ERR и errno установлена

signal и sigset

- Обработчик, установленный через signal(2), сбрасывает реакцию на сигнал на SIG_DFL
- Обработчик, установленный через sigset(2), блокирует сигнал на время своей работы, и разблокирует после возврата

Pause(2)

```
NAME

       pause - suspend process until signal

SYNOPSIS

       #include <unistd.h>

       int pause(void);


DESCRIPTION

       The pause() function suspends the calling process until it recei
       signal. The signal must be one that is not currently set to be i
       the calling process.


       If the signal causes termination of the calling process, pause()
       return.


       If the signal is caught by the calling process and control is re
       from the signal-catching function (see signal(3C)), the calling
       resumes execution from the point of suspension.

RETURN VALUES

       Since pause() suspends thread  execution indefinitely unless int
       by a signal, there is no successful completion return value.  It
       interrupted, it returns -1 and sets errno to indicate the error

ERRORS

       The pause() function will fail if:

       EINTR
                A signal is caught by the calling process and control i
                returned from the signal-catching function.
```

Имя	Описание	ISO C	SUS	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Действие по умолчанию
SIGABRT	Аварийное завершение (abort)	✓	✓	✓	✓	✓	✓	Завершить + core
SIGALRM	Истекло время таймера (alarm)		✓	✓	✓	✓	✓	Завершить
SIGBUS	Аппаратная ошибка		✓	✓	✓	✓	✓	Завершить + core
SIGCANCEL	Для внутреннего использования библиотекой threads						✓	Игнорировать
SIGCHLD	Изменение состояния дочернего процесса		✓	✓	✓	✓	✓	Игнорировать
SIGCONT	Возобновить работу приостановленного процесса		✓	✓	✓	✓	✓	Продолжить/игнорировать
SIGEMT	Аппаратная ошибка			✓	✓	✓	✓	Завершить + core
SIGFPE	Арифметическая ошибка	✓	✓	✓	✓	✓	✓	Завершить + core
SIGFREEZE	Закрепление контрольной точки						✓	Игнорировать
SIGHUP	Обрыв связи		✓	✓	✓	✓	✓	Завершить
SIGILL	Недопустимая инструкция	✓	✓	✓	✓	✓	✓	Завершить + core
SIGINFO	Запрос состояния с клавиатуры			✓		✓		Игнорировать
SIGINT	С терминала введен символ прерывания	✓	✓	✓	✓	✓	✓	Завершить
SIGIO	Асинхронный ввод/вывод			✓	✓	✓	✓	Завершить/игнорировать

Имя	Описание	ISO C	SUS	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Действие по умолчанию
SIGIOT	Аппаратная ошибка			✓	✓	✓	✓	Завершить + core
SIGJVM1	Для внутреннего использования виртуальной машиной Java						✓	Игнорировать
SIGJVM2	Для внутреннего использования виртуальной машиной Java						✓	Игнорировать
SIGKILL	Завершение		✓	✓	✓	✓	✓	Завершить
SIGLOST	Ресурс потерян						✓	Завершить
SIGLWP	Для внутреннего использования библиотекой threads			✓			✓	Завершить/игнорировать
SIGPIPE	Запись в канал, из которого никто не читает		✓	✓	✓	✓	✓	Завершить
SIGPOLL	Событие опроса (poll)				✓		✓	Завершить
SIGPROF	Истекло время профилирующего таймера (setitimer)			✓	✓	✓	✓	Завершить
SIGPWR	Падение напряжения питания/перезапуск				✓		✓	Завершить/игнорировать
SIGQUIT	С терминала введен символ завершения		✓	✓	✓	✓	✓	Завершить + core
SIGSEGV	Ошибка доступа к памяти	✓	✓	✓	✓	✓	✓	Завершить + core
SIGSTKFLT	Ошибка, связанная со стеком				✓			Завершить

Имя	Описание	ISO C	SUS	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Действие по умолчанию
SIGSTOP	Приостановить процесс		✓	✓	✓	✓	✓	Остановить процесс
SIGSYS	Неверный системный вызов		XSI	✓	✓	✓	✓	Завершить + core
SIGTERM	Завершение	✓	✓	✓	✓	✓	✓	Завершить
SIGTHAW	Освобождение контрольной точки						✓	Игнорировать
SIGTHR	Для внутреннего использования библиотекой `threads`			✓				Завершить
SIGTRAP	Аппаратная ошибка		XSI	✓	✓	✓	✓	Завершить + core
SIGTSTP	С терминала введен символ приостановки		✓	✓	✓	✓	✓	Остановить процесс
SIGTTIN	Чтение из управляющего терминала фоновым процессом		✓	✓	✓	✓	✓	Остановить процесс
SIGTTOU	Запись в управляющий терминал фоновым процессом		✓	✓	✓	✓	✓	Остановить процесс
SIGURG	Экстренное событие (сокеты)		✓	✓	✓	✓	✓	Игнорировать
SIGUSR1	Определяемый пользователем сигнал		✓	✓	✓	✓	✓	Завершить
SIGUSR2	Определяемый пользователем сигнал		✓	✓	✓	✓	✓	Завершить
SIGVTALRM	Истекло время виртуального таймера (`setitimer`)		XSI	✓	✓	✓	✓	Завершить

Имя	Описание	ISO C	SUS	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10	Действие по умолчанию
SIGWAITING	Для внутреннего использования библиотекой `threads`						✓	Игнорировать
SIGWINCH	Изменение размеров окна терминала			✓	✓	✓	✓	Игнорировать
SIGXCPU	Исчерпан лимит процессорного времени (`setrlimit`)		XSI	✓	✓	✓	✓	Завершить/завершить + core
SIGXFSZ	Превышено ограничение на размер файла (`setrlimit`)		XSI	✓	✓	✓	✓	Завершить/завершить + core
SIGXRES	Превышено ограничение на использование ресурса						✓	Завершить + core/игнорировать

Reading list