Лабораторная работа №1
Работа со звуком

Цель работы: научиться работать со звуковой подсистемой GNU/Linux, включая вывод аудио-файлов, получение информации об аппаратных возможностях аудио-карты и регулирование громкости отдельных каналов микшера.

Краткие теоретические сведения

1. Звуковая подсистема GNU/Linux

В Linux, как и в других UNIX-совместимых системах, пользовательские программы редко обращаются к аппаратному обеспечению напрямую. У ядра есть драйверы для всех устройств. Все вопросы, касающиеся отличий звуковых карт, берет на себя модуль ядра, отвечающий за поддержку конкретной аудио-карты — то есть *драйвер* этого конкретного аудио-устройства.

В соответствии с традициями Unix, в виртуальной файловой системе /dev/ присутствуют несколько стандартных файлов устройств, обращаясь к которым можно менять настройки аудио-карты (например, уровень громкости на различных каналах), выводить звук на линейный выход или считывать звук с линейного входа. Благодаря драйверу аудио-карты, взаимодействие с этими файлами устройств не зависит от того, какая именно аудио-карта установлена в компьютере, и можно легко создавать универсальные программы, одинаково работающие со звуком на разных системах. Знание конкретных адресов портов и команд микроконтроллера, отвечающего за ввод, обработку и вывод звука (т. н, аудио-процессор, от англ. digital sound processor или DSP) требуется только разработчику аудио-драйвера (и это очень хорошо, поскольку аудио-карты слабо стандартизированы).

Файловая абстракция очень удобна для работы со звуком: например, запись звукового файла байт за байтом в специальный файл устройства приводит к тому, что файл проигрывается на подключенных к компьютеру колонках (или наушниках).

В GNU/Linux на сегодняшний день существует несколько аудио-

подсистем, файлы устройств у которых несколько различаются:

- OSS (от англ. Open Sound System) - исторически первая аудио-подсистема, использовавшаяся раньше в Linux и до сих пор активно используемая во многих Unix-подобных системах;
- Alsa (сокращение от Andvanced Linux Sound Architecture) — более новая аудио-подсистема, разработанная специально для Linux, и изначально поддерживающая ряд дополнительных возможностей более дорогих аудио-карт, таких как аппаратное смешивание (*микширование*) звуковых каналов, полнодуплексный режим и т. д.
- PulseAudio — высокоуровневая надстройка над аудио-подсистемой (обычно над Alsa), умеющая выполнять программное микширование звука, трансляцию звука по сети, эмуляцию других аудио-подсистем и т. д.

Дальше мы будем рассматривать файлы устройств для работы со звуком, существующие в звуковой подсистеме OSS. Остальные системы умеют эмулировать работу OSS, а потому этот подход наиболее универсален.

Наиболее часто используемые файлы устройств в системе OSS:

- /dev/mixer — позволяет настроить уровни громкости для различных входных и выходных каналов звуковой карты; используется для доступа к аппаратному микшеру, встроенному в звуковую карту (по этой причине возможности, поддерживаемые /dev/mixer, различаются для разных аудио-карт).
- /dev/dsp — используется для вывода звука (когда в файл устройства записываются несжатые аудио-данные) или для считывание звука с аудио-входов (когда выполняется чтение из файла устройства).

На системах, в которых используется звуковая система PulseAudio (например, все современные версии Ubuntu Linux) файлы устройств в стиле OSS могут отсутствовать. Для их эмуляции используется специальная утилита `padsp`. Программа, работающая со звуком через /dev/dsp, будет «видеть» необходимые файлы устройств, если будет запущена этой утилитой. Например, запуск

программы myMediaPlayer будет выглядеть так:

```
padsp myMedaPlayer
```

Этот способ запуска мы будем использовать далее при выполнении заданий лабораторной работы.

1.1. Воспроизведение звука командой cat

Простейший способ воспроизвести звук — перенаправить звуковой файл в устройство /dev/dsp с помощью команды cat:

```
cat myNewSound >/dev/dsp
```

Файл myNewSound в приведенном примере должен быть в несжатом аудио-формате. Например, для этой цели подходит обычный WAV-файл (формат, известный также как Windows PCM).

Соответственно, простейший способ записи звука выглядит точно так же, но данные не записываются из файла в /dev/dsp, а наоборот — читаются из /dev/dsp в файл:

```
cat /dev/dsp1 >myNewSound
```

1.2. Программирование аудио-устройств

1.2.1 Файл устройства /dev/mixer

Каналы микшера можно разделить на 2 группы: входные (для записи звука) и выходные (для его проигрывания). Из-за большого числа и разнообразия каналов для регулирования уровней звука с помощью `/dev/mixer` применен более сложный подход, чем простые файловые операции чтения и записи. Стандартные операции, необходимые для работы с микшером — это открытие файла (системный вызов `open`), его закрытие (системный вызов `close`), а также не использовавшийся нами ранее системый вызов `ioctl`, через который и выполняются все настройки.

1.2.2 Системный вызов ioctl

Прототип функции `ioctl` выглядит следующим образом:

```
int ioctl(int fd, int request, ...);
```

Функция используется для выполнения таких операций над файлами и устройствами, которые не входят в число операций чтения и записи. Каждый

запрос специфичен для различных устройств.

Первый параметр функции — это файловый описатель (тот самый, который вернула функция `open`). Второй — тип нужной нестандартной операции. Необязательный третий параметр специфичен для конкретных устройств.

1.2.3 Параметры микшера

Константы, соответствующие наиболее типичным каналам микшера, приведены в следующей таблице. Обратите внимание, что наряду с реальными каналами звукового тракта (линейный вход, линейный выход, вход для подключения audio-CD) в микшере присутствуют «виртуальные каналы», используемые для регулировки каких-либо параметров звукового тракта (например, аудио-устройство может иметь «канал» для регулировки высоких или низких частот).

Таблица 1 — Константы, обозначающие названия каналов микшера

Название канала	**Описание**
SOUND_MIXER_VOLUME	Канал одновременной регулировки громкости для всего сразу
SOUND_MIXER_BASS	Канал для регулировки низких частот
SOUND_MIXER_TREBLE	Кнал для регулировки высоких частот
SOUND_MIXER_SYNTH	FM-синтезатор
SOUND_MIXER_PCM	Уровень цифро-аналогового преобразователя (ЦАП)
SOUND_MIXER_SPEAKER	Громкость системного динамика
SOUND_MIXER_LINE	Линейный вход
SOUND_MIXER_MIC	Вход для подключения микрофона
SOUND_MIXER_CD	Вход audio-CD
SOUND_MIXER_IMIX	Громкость воспроизведения с устройства записи
SOUND_MIXER_ALTPCM	Второй ЦАП
SOUND_MIXER_RECLEV	Регулировка громкости при записи звука, действующая сразу на все входные каналы
SOUND_MIXER_IGAIN	Входной уровень усиления звука
SOUND_MIXER_OGAIN	Выходной уровень усиления звука

Приведем пример определения уровня громкости на входе для подключения микрофона:

```
int vol;
ioctl(fd, MIXER_READ(SOUND_MIXER_MIC), &vol);
printf("Уровень усиления звука на входе: %d %%\n", vol);
```

В этом примере предполагается, что файл /dev/mixer был предварительно открыт функцией open(), и эта функция вернула описатель файла в переменную fd.

Теперь поясним, что представляет собой указатель vol (третий параметр функции). При использовании для чтения параметров микшера, функция ioctl() сохраняет по этому адресу считанное значение громкости.

Такой запутанный способ получения значения (вместо возвращаемого значения функции) используется из-за того, что многие каналы микшера стереофонические. Для стерео-каналов громкость включает два значения (по одному на левый и правый канал). Если по адресу vol была сохранена пара значений, то первый байт соответствует правому каналу, а последний — левому:

```
int left, right;
left  =  vol & 0xff;
right = (vol & 0xff00) >> 8;
printf("Громкость левого канала %d %%, правого канала %d %%\n", left, right);
```

Для моно-устройств (один канал) уровень шума находится в младшем байте.

Установка уровней громкости производится аналогично:

```
vol = (right << 8) + left;
ioctl(fd, MIXER_WRITE(SOUND_MIXER_MIC), &vol);
```

Дополнительно можно получить символьные имена каналов:

```
const char *labels[] = SOUND_DEVICE_LABELS;
const char *names[]  = SOUND_DEVICE_NAMES;
```

Для поиска информации о микшере используется несколько вызовов ioctl. Все они возвращают битовые маски, соответствующие каналам микшера. SOUND_MIXER_READ_DEVMASK возвращает битовую маску, где для каждого, поддерживаемого микшером канала установлен соответствующий бит. SOUND_MIXER_READ_RECMASK — биты для каждого канала, который может быть использован, как устройство записи. Для примера, можно

проверить, поддерживается ли микшером канал CD-входа:

```
ioctl(fd, SOUND_MIXER_READ_DEVMASK, &devmask);
if (devmask & SOUND_MIXER_CD)
  printf("CD-вход поддерживается");
```

Так же можно определить, доступен ли этот канал как устройство записи:

```
ioctl(fd, SOUND_MIXER_READ_RECMASK, &recmask);
if (recmask & SOUND_MIXER_CD)
  printf("CD-вход можно использовать для записи звука");
```

SOUND_MIXER_READ_RECSRC — каналы, выбранные сейчас для записи.

SOUND_MIXER_READ_STEREODEVS — каналы, поддерживающие стерео.

Похожий вызов ioctl возвращает информацию о звуковой плате в целом: SOUND_MIXER_READ_CAPS. Каждый бит соответствует возможностям карты. Сейчас существует один бит: SOUND_CAP_EXCL_INPUT. Если он установлен — источников записи может быть несколько.

SOUND_MIXER_WRITE_RECSRC — источник записи.

Пример, устанавливающий CD в качестве источника записи:

```
devmask = SOUND_MIXER_CD;
ioctl(fd, SOUND_MIXER_WRITE_DEVMASK, &devmask);
```

Обратите внимание, что функции необходимо передавать указатель.

1.2.4 Пример

Пример ниже иллюстрирует некоторые концепции системных вызовов:

```
/*
 * syscalls.c
 * Программа только иллюстрирует работу с системными вызовами
 * пока не делает ничего полезного, но далее будет расширена.
 */

#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <linux/soundcard.h>

int main()
{
  int fd;                  /* описатель файла */
  int arg;                 /* аргумент к вызову ioctl */
  unsigned char buf[1000];  /* буфер для данных */
  int status;              /* возвращаемый статус */

  /* открытие устройства */
  status = fd = open("/dev/dsp", O_RDWR);
  if (status == -1) {
    perror("error opening /dev/dsp");
    exit(1);
  }

  /* задание параметров ioctl */
```

```
  arg = 8000; /* частота дискретизации */
  status = ioctl(fd, SOUND_PCM_WRITE_RATE, &arg);
  if (status == -1) {
    perror("error from SOUND_PCM_WRITE_RATE ioctl");
    exit(1);
  }

  /* чтение */
  status = read(fd, buf, sizeof(buf));
  if (status == -1) {
    perror("error reading from /dev/dsp");
    exit(1);
  }

  /* запись */
  status = write(fd, buf, sizeof(buf));
  if (status == -1) {
    perror("error writing to /dev/dsp");
    exit(1);
  }

  /* закрытие */
  status = close(fd);
  if (status == -1) {
    perror("error closing /dev/dsp");
    exit(1);
  }

  /* и выход */
  return(0);
}
```

1.3 Программирование /dev/dsp

/dev/dsp — это цифровое устройство воспроизведения и записи. Запись в устройство приводит к воспроизведению звука (ЦАП). Чтение — запись звука и его анализ (АЦП).

Если данные читать слишком медленно, то переполненный буфер приведет к «отбрасыванию» лишних данных, из-за чего звук станет воспроизводиться рывками. Если читать слишком быстро, то ядро будет блокировать запросы до тех пор, пока не накопится необходимое количество данных. Само собой, в конце воспроизведения или записи не будет сообщения о конце файла.

Формат принимаемых данных зависит от того, как устройство было настроено вызовом ioctl. Стандартные настройки таковы: 8-битные беззнаковые дискретные отсчеты аудио-потока (сэмплы) с одним каналом (моно) и частотой дискретизации 8кГц. Аналогичные настройки по умолчанию приняты и для записи.

Константы ioctl описаны в заголовочном файле: *linux/soundcard.h*.

```
SOUND_PCM_WRITE_BITS
SOUND_PCM_READ_BITS
SOUND_PCM_WRITE_CHANNELS
SOUND_PCM_READ_CHANNELS
SOUND_PCM_WRITE_RATE
SOUND_PCM_READ_RATE
```

1.3.1 Пример

Ниже приводится короткий пример программирования DSP, записывающий несколько секунд звука в массив в памяти и затем воспроизводящий его.

```
/*
 * выходит по ctrl-c
 */

#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/ioctl.h>
#include <stdlib.h>
#include <stdio.h>
#include <linux/soundcard.h>

#define LENGTH 3    /* сколько секунд записывать */
#define RATE 8000   /* частота дискретизации */
#define SIZE 8      /* глубина: 8 или 16 бит */
#define CHANNELS 1  /* 1 = моно 2 = стерео */

/* буфер для цифрового звука */
unsigned char buf[LENGTH*RATE*SIZE*CHANNELS/8];

int main()
{
  int fd;       /* файловый описатель */
  int arg;      /* аргумент к ioctl */
  int status;   /* возвращаемый системой результат */

  /* открытие устройства */
  fd = open("/dev/dsp", O_RDWR);
  if (fd < 0) {
    perror("open of /dev/dsp failed");
    exit(1);
  }

  /* задание параметров */
  arg = SIZE;      /* размер сэмпла */
  status = ioctl(fd, SOUND_PCM_WRITE_BITS, &arg);
  if (status == -1)
    perror("SOUND_PCM_WRITE_BITS ioctl failed");
  if (arg != SIZE)
    perror("unable to set sample size");

  arg = CHANNELS;  /* моно или стерео */
  status = ioctl(fd, SOUND_PCM_WRITE_CHANNELS, &arg);
  if (status == -1)
    perror("SOUND_PCM_WRITE_CHANNELS ioctl failed");
  if (arg != CHANNELS)
    perror("unable to set number of channels");

  arg = RATE;      /* частота дискретизации */
  status = ioctl(fd, SOUND_PCM_WRITE_RATE, &arg);
```

```
  if (status == -1)
    perror("SOUND_PCM_WRITE_WRITE ioctl failed");

  while (1) { /* loop until Control-C */
    printf("Say something:\n");
    status = read(fd, buf, sizeof(buf)); /* запись звука */
    if (status != sizeof(buf))
      perror("read wrong number of bytes");
    printf("You said:\n");
    status = write(fd, buf, sizeof(buf)); /* воспроизведение */
    if (status != sizeof(buf))
      perror("wrote wrong number of bytes");
    /* wait for playback to complete before recording again */
    status = ioctl(fd, SOUND_PCM_SYNC, 0);
  if (status == -1)
    perror("SOUND_PCM_SYNC ioctl failed");
  }
}
```

Задания

1. Внести незначительные изменения в программу для работы с /dev/dsp: частота дискретизации, размер, время записи. Описать изменение качества звука при изменении частоты дискретизации.
2. Воспроизвести звуковой сэмпл в обратном порядке (для /dev/dsp).
3. Написать программу, позволяющую выбирать устройство записи в микшере (/dev/mixer).
4. Получить данные о возможностях аудио-карты, воспользовавшись для этого следующими константами:
    - SOUND_MIXER_READ_RECSRC,
    - SOUND_MIXER_READ_DEVMASK,
    - SOUND_MIXER_READ_RECMASK,
    - SOUND_MIXER_READ_STEREODEVS,
    - SOUND_MIXER_READ_CAPS.
5. Вывести информацию о возможностях аудио-карты в виде следующей таблицы:

```
Status of /dev/mixer:
Mixer      Device     Recording  Active     Stereo     Current
Channel    Available  Source     Source     Device     Level
---------  ---------  ---------  --------   ---------  ---------
 0 Vol      yes        no         no         yes        90% 90%
 1 Bass     no         no         no         no
 2 Trebl    no         no         no         no
 3 Synth    yes        no         no         yes        75% 75%
```

```
 4 Pcm      yes         no          no          yes         75% 75%
 5 Spkr     no          no          no          no
 6 Line     yes         yes         no          yes         75% 75%
 7 Mic      yes         yes         yes         no          16%
 8 CD       yes         yes         no          yes         75% 75%
 9 Mix      no          no          no          no
10 Pcm2     no          no          no          no
11 Rec      no          no          no          no
12 IGain    no          no          no          no
13 OGain    no          no          no          no
14 Line1    no          no          no          no
15 Line2    no          no          no          no
16 Line3    no          no          no          no
```