OSS Sequencer Emulation on ALSA

ver.0.1.8; Nov. 16, 1999

Description

This directory contains the OSS sequencer emulation driver on ALSA. Note that this program is still in the development state.

What this does - it provides the emulation of the OSS sequencer, access via `/dev/sequencer` and `/dev/music` devices. The most of applications using OSS can run if the appropriate ALSA sequencer is prepared.

The following features are emulated by this driver:

- Normal sequencer and MIDI events:

  They are converted to the ALSA sequencer events, and sent to the corresponding port.

- Timer events:

  The timer is not selectable by ioctl. The control rate is fixed to 100 regardless of HZ. That is, even on Alpha system, a tick is always 1/100 second. The base rate and tempo can be changed in `/dev/music`.

- Patch loading:

  It purely depends on the synth drivers whether it's supported since the patch loading is realized by callback to the synth driver.

- I/O controls:

  Most of controls are accepted. Some controls are dependent on the synth driver, as well as even on original OSS.

Furthermore, you can find the following advanced features:

- Better queue mechanism:

  The events are queued before processing them.

- Multiple applications:

  You can run two or more applications simultaneously (even for OSS sequencer)! However, each MIDI device is exclusive - that is, if a MIDI device is opened once by some application, other applications can't use it. No such a restriction in synth devices.

- Real-time event processing:

  The events can be processed in real time without using out of bound ioctl. To switch to real-time mode, send ABSTIME 0 event. The followed events will be processed in real-time without queued. To switch off the real-time mode, send RELTIME 0 event.

- `/proc` interface:

  The status of applications and devices can be shown via `/proc/asound/seq/oss` at any time. In the later version, configuration will be changed via `/proc` interface, too.

Installation

Run configure script with both sequencer support (`--with-sequencer=yes`) and OSS emulation (`--with-oss=yes`) options. A module `snd-seq-oss.o` will be created. If the synth module of your sound card supports for OSS emulation (so far, only Emu8000 driver), this module will be loaded automatically. Otherwise, you need to load this module manually.

At beginning, this module probes all the MIDI ports which have been already connected to the sequencer. Once after that, the creation and deletion of ports are watched by announcement mechanism of ALSA sequencer.

The available synth and MIDI devices can be found in proc interface. Run `cat /proc/asound/seq/oss`, and check the devices. For example, if you use an AWE64 card, you'll see like the following:

```
OSS sequencer emulation version 0.1.8
ALSA client number 63
ALSA receiver port 0
```

```
Number of applications: 0

Number of synth devices: 1
synth 0: [EMU8000]
  type 0x1 : subtype 0x20 : voices 32
  capabilties : ioctl enabled / load_patch enabled

Number of MIDI devices: 3
midi 0: [Emu8000 Port-0] ALSA port 65:0
  capability write / opened none

midi 1: [Emu8000 Port-1] ALSA port 65:1
  capability write / opened none

midi 2: [0: MPU-401 (UART)] ALSA port 64:0
  capability read/write / opened none
```

Note that the device number may be different from the information of `/proc/asound/oss-devices` or ones of the original OSS driver. Use the device number listed in `/proc/asound/seq/oss` to play via OSS sequencer emulation.

Using Synthesizer Devices

Run your favorite program. I've tested playmidi-2.4, awemidi-0.4.3, gmod-3.1 and xmp-1.1.5. You can load samples via `/dev/sequencer` like sfxload, too.

If the lowlevel driver supports multiple access to synth devices (like Emu8000 driver), two or more applications are allowed to run at the same time.

Using MIDI Devices

So far, only MIDI output was tested. MIDI input was not checked at all, but hopefully it will work. Use the device number listed in `/proc/asound/seq/oss`. Be aware that these numbers are mostly different from the list in `/proc/asound/oss-devices`.

Module Options

The following module options are available:

maxqlen
: specifies the maximum read/write queue length. This queue is private for OSS sequencer, so that it is independent from the queue length of ALSA sequencer. Default value is 1024.

seq_oss_debug
: specifies the debug level and accepts zero (= no debug message) or positive integer. Default value is 0.

Queue Mechanism

OSS sequencer emulation uses an ALSA priority queue. The events from `/dev/sequencer` are processed and put onto the queue specified by module option.

All the events from `/dev/sequencer` are parsed at beginning. The timing events are also parsed at this moment, so that the events may be processed in real-time. Sending an event ABSTIME 0 switches the operation mode to real-time mode, and sending an event RELTIME 0 switches it off. In the real-time mode, all events are dispatched immediately.

The queued events are dispatched to the corresponding ALSA sequencer ports after scheduled time by ALSA sequencer dispatcher.

If the write-queue is full, the application sleeps until a certain amount (as default one half) becomes empty in blocking mode. The synchronization to write timing was implemented, too.

The input from MIDI devices or echo-back events are stored on read FIFO queue. If application reads `/dev/sequencer` in blocking mode, the process will be awaked.

Interface to Synthesizer Device

Registration

To register an OSS synthesizer device, use snd_seq_oss_synth_register() function:

```
int snd_seq_oss_synth_register(char *name, int type, int subtype, int nvoices,
        snd_seq_oss_callback_t *oper, void *private_data)
```

The arguments `name`, `type`, `subtype` and `nvoices` are used for making the appropriate synth_info structure for ioctl. The return value is an index number of this device. This index must be remembered for unregister. If registration is failed, -errno will be returned.

To release this device, call snd_seq_oss_synth_unregister() function:

```
int snd_seq_oss_synth_unregister(int index)
```

where the `index` is the index number returned by register function.

Callbacks

OSS synthesizer devices have capability for sample downloading and ioctls like sample reset. In OSS emulation, these special features are realized by using callbacks. The registration argument oper is used to specify these callbacks. The following callback functions must be defined:

```
snd_seq_oss_callback_t:
 int (*open)(snd_seq_oss_arg_t *p, void *closure);
 int (*close)(snd_seq_oss_arg_t *p);
 int (*ioctl)(snd_seq_oss_arg_t *p, unsigned int cmd, unsigned long arg);
 int (*load_patch)(snd_seq_oss_arg_t *p, int format, const char *buf, int offs, int count);
 int (*reset)(snd_seq_oss_arg_t *p);
```

Except for `open` and `close` callbacks, they are allowed to be NULL.

Each callback function takes the argument type `snd_seq_oss_arg_t` as the first argument.

```
struct snd_seq_oss_arg_t {
    int app_index;
    int file_mode;
    int seq_mode;
    snd_seq_addr_t addr;
    void *private_data;
    int event_passing;
};
```

The first three fields, `app_index`, `file_mode` and `seq_mode` are initialized by OSS sequencer. The `app_index` is the application index which is unique to each application opening OSS sequencer. The `file_mode` is bit-flags indicating the file operation mode. See `seq_oss.h` for its meaning. The `seq_mode` is sequencer operation mode. In the current version, only `SND_OSSSEQ_MODE_SYNTH` is used.

The next two fields, `addr` and `private_data`, must be filled by the synth driver at open callback. The `addr` contains the address of ALSA sequencer port which is assigned to this device. If the driver allocates memory for `private_data`, it must be released in close callback by itself.

The last field, `event_passing`, indicates how to translate note-on / off events. In `PROCESS_EVENTS` mode, the note 255 is regarded as velocity change, and key pressure event is passed to the port. In `PASS_EVENTS` mode, all note on/off events are passed to the port without modified. `PROCESS_KEYPRESS` mode checks the note above 128 and regards it as key pressure event (mainly for Emu8000 driver).

Open Callback

The `open` is called at each time this device is opened by an application using OSS sequencer. This must not be NULL. Typically, the open callback does the following procedure:

1. Allocate private data record.
2. Create an ALSA sequencer port.
3. Set the new port address on `arg->addr`.
4. Set the private data record pointer on `arg->private_data`.

Note that the type bit-flags in port_info of this synth port must NOT contain `TYPE_MIDI_GENERIC` bit. Instead, `TYPE_SPECIFIC` should be used. Also, `CAP_SUBSCRIPTION` bit should NOT be included, too. This is necessary to tell it from other normal MIDI devices. If the open procedure succeeded, return zero. Otherwise, return -errno.

Ioctl Callback

The `ioctl` callback is called when the sequencer receives device-specific ioctls. The following two ioctls should be processed by this callback:

IOCTL_SEQ_RESET_SAMPLES
: reset all samples on memory -- return 0

IOCTL_SYNTH_MEMAVL
: return the available memory size

FM_4OP_ENABLE
: can be ignored usually

The other ioctls are processed inside the sequencer without passing to the lowlevel driver.

Load_Patch Callback

The `load_patch` callback is used for sample-downloading. This callback must read the data on user-space and transfer to each device. Return 0 if succeeded, and -errno if failed. The format argument is the patch key in patch_info record. The buf is user-space

pointer where patch_info record is stored. The offs can be ignored. The count is total data size of this sample data.

Close Callback

The `close` callback is called when this device is closed by the application. If any private data was allocated in open callback, it must be released in the close callback. The deletion of ALSA port should be done here, too. This callback must not be NULL.

Reset Callback

The `reset` callback is called when sequencer device is reset or closed by applications. The callback should turn off the sounds on the relevant port immediately, and initialize the status of the port. If this callback is undefined, OSS seq sends a `HEARTBEAT` event to the port.

Events

Most of the events are processed by sequencer and translated to the adequate ALSA sequencer events, so that each synth device can receive by input_event callback of ALSA sequencer port. The following ALSA events should be implemented by the driver:

ALSA event	**Original OSS events**
NOTEON	SEQ_NOTEON, MIDI_NOTEON
NOTE	SEQ_NOTEOFF, MIDI_NOTEOFF
KEYPRESS	MIDI_KEY_PRESSURE
CHANPRESS	SEQ_AFTERTOUCH, MIDI_CHN_PRESSURE
PGMCHANGE	SEQ_PGMCHANGE, MIDI_PGM_CHANGE
PITCHBEND	SEQ_CONTROLLER(CTRL_PITCH_BENDER), MIDI_PITCH_BEND
CONTROLLER	MIDI_CTL_CHANGE, SEQ_BALANCE (with CTL_PAN)
CONTROL14	SEQ_CONTROLLER
REGPARAM	SEQ_CONTROLLER(CTRL_PITCH_BENDER_RANGE)
SYSEX	SEQ_SYSEX

The most of these behavior can be realized by MIDI emulation driver included in the Emu8000 lowlevel driver. In the future release, this module will be independent.

Some OSS events (`SEQ_PRIVATE` and `SEQ_VOLUME` events) are passed as event type SND_SEQ_OSS_PRIVATE. The OSS sequencer passes these event 8 byte packets without any modification. The lowlevel driver should process these events appropriately.

Interface to MIDI Device

Since the OSS emulation probes the creation and deletion of ALSA MIDI sequencer ports automatically by receiving announcement from ALSA sequencer, the MIDI devices don't need to be registered explicitly like synth devices. However, the MIDI port_info registered to ALSA sequencer must include a group name `SND_SEQ_GROUP_DEVICE` and a capability-bit `CAP_READ` or `CAP_WRITE`. Also, subscription capabilities, `CAP_SUBS_READ` or `CAP_SUBS_WRITE`, must be defined, too. If these conditions are not satisfied, the port is not registered as OSS sequencer MIDI device.

The events via MIDI devices are parsed in OSS sequencer and converted to the corresponding ALSA sequencer events. The input from MIDI sequencer is also converted to MIDI byte events by OSS sequencer. This works just a reverse way of seq_midi module.

Known Problems / TODO's

- Patch loading via ALSA instrument layer is not implemented yet.