



Practice 1: Study of the behavior of the soundcard



This practice aims to allow the student to familiarize with the programming of the audio devices. To do so, simple experiments are proposed and simple questions are associated to each experiment. You must take into account that you can solve similar doubts concerning the working of devices or calls with a similar methodology.

Experiment 1: Timing of the sound card while reading

To obtain the (digital) data that the soundcard has converted from an analog sound source, we must perform a `read` operation in the descriptor `/dev/dsp`, which represents that card. When we execute the `read` function, the card tries to obtain as many bytes as have been requested in the parameter 'number of bytes' included in the `read` call. Remember that the reading operation in the soundcard usually blocks the process for time needed to capture the requested data.

The test will be performed with a audio format of 8 bits per sample, 8kHz of sampling frequency, single channel recording (mono). Therefore, the data generation speed is 8kBytes per second. During the experiment we vary:

- Soundcard's fragment size
- Amount of requested data in the `read` call.

In both cases, we request powers of two (although in the second case this is not essential).

The objective of this experiment is to obtain experimental information about the working of the `read` operation over the sound card.

To carry out the experiment:

Modify the source code of the `record` function of the file `audioSimple.c` in the following way:

1. Modify the size of the data requested on each `read` operation. Be sure that the fragment size configured is the same as the data requested for each `read` operation.
2. Measure the blocking time of the `read` operation. To do that you must:
 - a. Obtain the previous and following time instants with respect to the `read` call. In order to do so you can use the function `gettimeofday`.
 - b. Subtract the two time values and show the value obtained on the screen. This value is the blocking time for the `read` operation. To compute the subtraction we recommend the use of the function `timersub`. Note: you must take into account that in some cases the times can be unexpectedly high. This could happen because the operating system assumes the control and assigns the CPU time to another process or performs a computation during the measure interval.

The designed code will try to read data from the sound card at the fastest possible speed.

Once you made the aforementioned modifications, **modify the fragment size and the buffer size.**

Analyze the following cases:

Fragment size < amount of requested data to read.

Fragment size = amount of requested data to read.

Fragment size > amount of requested data to read (remember that in this case there can be continuous reads and it may seem that some data are obtained at a smaller time than the speed at that they are captured –although other data will need more time to be got, so that the sum of the times of several consecutive reads is coherent.)

To understand the behavior, it is interesting that you know the theoretical value of the necessary time to obtain a certain amount of data from the sound card. Compare the obtained values with this expected value. Check with several sizes (of fragment and read request) between 64 bytes and 4096 bytes (you do not have to check all the combination, only some you consider relevant).

Note: the soundcard that you are working with at the computers of the lab has a maximum fragment size (if you set a fragment size larger than the maximum, the sound card behaves as if the fragment would be of the maximum size). Identify this maximum size.

Experiment 2: Timing of the sound card while reading using `select`

This experiment is similar to the previous one, but in this occasion you must execute first a `select` (including in the 'read file descriptor set' the descriptor of the soundcard). Therefore, you will block it in the `select` instead of in the `read`. When the `select` unblocks, you will execute a `read` (since in this case this is the only reasonable cause for the unblocking of a `select`), and you will repeat the process blocking again the code with a `select`. You will take times to estimate the necessary time to execute both a `select` and a `read`.

As with the previous section, modify the fragment size and the buffer size. Analyze what happens in the different situations: fragment size < amount of requested data to read, fragment size = amount of requested data to read, fragment size > amount of requested data to read.

In this case, it is important to measure the times both in the `select` and the `read` operation (remember that when you execute the `select` operation, YOU DOES NOT INDICATE the amount of requested data – so the operating system has no way to know it).

**Note: Both the buffer size and the fragment size must be defined as a power of 2.*

Experiment 3: Timing the sound card while writing

With the writing operation (`write`) of the sound card we indicate to the soundcard to transform the data to audio. The `write` operation differs from the `read` operation in not blocking the process in its first executions.

To perform this experiment, modify the code of the function `audioSimple.c` to capture the execution time of the `write` function. Do it in the same way as for the `read` function.

Record an audio file executing:

```
$>./audioSimple record file
```

and playback the file (take into account that the block size used for recording not need to be the same as the block size used for playing). Execute a loop in which you try to write continuously. Write in the soundcard blocks of different sizes. For each experiment, perform several executions modifying also the fragment size. Execute a sufficient number of writings until you observe that the writing operation blocks.

Analyze the following cases:

Fragment size < amount of requested data to write.

Fragment size = amount of requested data to write.

Fragment size > amount of requested data to write (in this case, observe whether the amount of data you can write without blocking is equal to the previous cases, or on the contrary, is smaller –because each time that you write a fragment is “occupied”, although it is not completely filled)

Check with several sizes (of fragment and write block) between 64 bytes and 4096 bytes (you do not have to check all the combination, only some you consider relevant).

With this experiment, we can also evaluate the total size of the buffer of the soundcard. We will suppose that when the card blocks is due to the lack of hardware memory in the sound card. If we calculate the amount of memory that has the sound card, we can obtain a good estimation of the buffer size. Then, select a fragment size of 4096 bytes, and perform writings repeatedly in the sound card of the same size. Identify the instant where the blocking is produced. In that instant: count the number of bytes sent to the sound card (multiplying the number of writing by 4096). However, the card will have consumed data while we were dumping information in the card. With the time from the start of the test until the instant where the block starts, and knowing that data are consumed at the rate of 8 kbytes per second, we can compute the amount of consumed data. That quantity is subtracted from the quantity of data that we have dumped.

Which is the size of the sound card buffer?

Experiment 4: Timing of the sound card while writing using `select`

This experiment is similar to the previous one, but in this occasion we do the write block with the `select` function.

Analyze the following cases:

Fragment size < amount of requested data to write

Fragment size = amount of requested data to write

Fragment size > amount of requested data to write

Experiment 5: Minimum timing threshold of `select`

In this experiment we will identify the minimum threshold for the `select` operation, that is, which is the minimum time during which `select` could be blocked. For that, program the timer of `select` to wake up with small times, measure the times before and after performing the `select` call, and find out which value is the first that verifies that the real blocking time of the process does not decreases.

Now we are going to evaluate the granularity of the blocking time of the `select` operation. Modify the values of the `select` timer approaching 20ms. Observe which actual values you obtain and whether they round up or round down to any value, and if that happens, which is the rounding “step” of that values (=granularity).