Instalación de JACK

JACK

- Para crear agentes de JACK se requiere instalar el servidor JACK.
- Así como las bibliotecas que permiten acceder a las funcionalidades de JACK.
  - Las bibliotecas oficiales son en el lenguaje C.
  - El sistema operativo mayormente soportado es Linux.

JACK y Otros Lenguajes

- Hay otros lenguajes que también pueden acceder a las funcionalidades de JACK.
  - Aunque no están soportados oficialmente.
- JAVA: **java-jack**
  - https://code.google.com/archive/p/java-jack/
- Python: JACK-client
  - https://pypi.python.org/pypi/JACK-Client/
- C++: JackCpp
  - http://x37v.info/projects/jackcpp/

JACK y Otros Sistemas Operativos

- También, se puede instalar JACK en otros sistemas operativos.
- Windows:
  - http://jackaudio.org/downloads/
    - Sección “Windows”
- Mac OS X (Snow Leopard)
  - http://jackaudio.org/downloads/
    - Sección “OS X”

JACK y Proyecto Final

- En este curso utilizaremos el lenguaje oficial de la biblioteca (C) en el sistema operativo mayormente soportado (Linux).
- Pero el proyecto final lo pueden hacer en cualquier lenguaje y en cualquier sistema operativo.
  - Se califica el desempeño final, no la herramienta.

"*Si funciona bien, no puedo argumentar en contra.*"

Instalación

- Cada computadora es su propio mundo, pero normalmente en Ubuntu 12.04 en adelante lo siguiente debe funcionar:

  sudo apt-get install jackd2 libjack-jackd2-dev pulseaudio-module-jack qjackctl

- Respondan que “sí” al preguntarles que si quieren que corra en tiempo real.

Instalación

- jackd2 : servidor de JACK
- qjackctl : una interfaz para configurar y controlar al servidor de JACK
- libjack-jackd2-dev : librerías de C para crear agentes de JACK
- pulseaudio-module-jack : módulo de PulseAudio para que puedan vivir a la vez JACK y PulseAudio

JACK en Tiempo Real

- El servidor de JACK corre en tiempo real sólo si el usuario que lo levanta pertenece al grupo “audio”.
- Utilicen el comando “groups” para saber a cuáles grupos su usuario pertenece.

Agregar Usuarios a Grupos

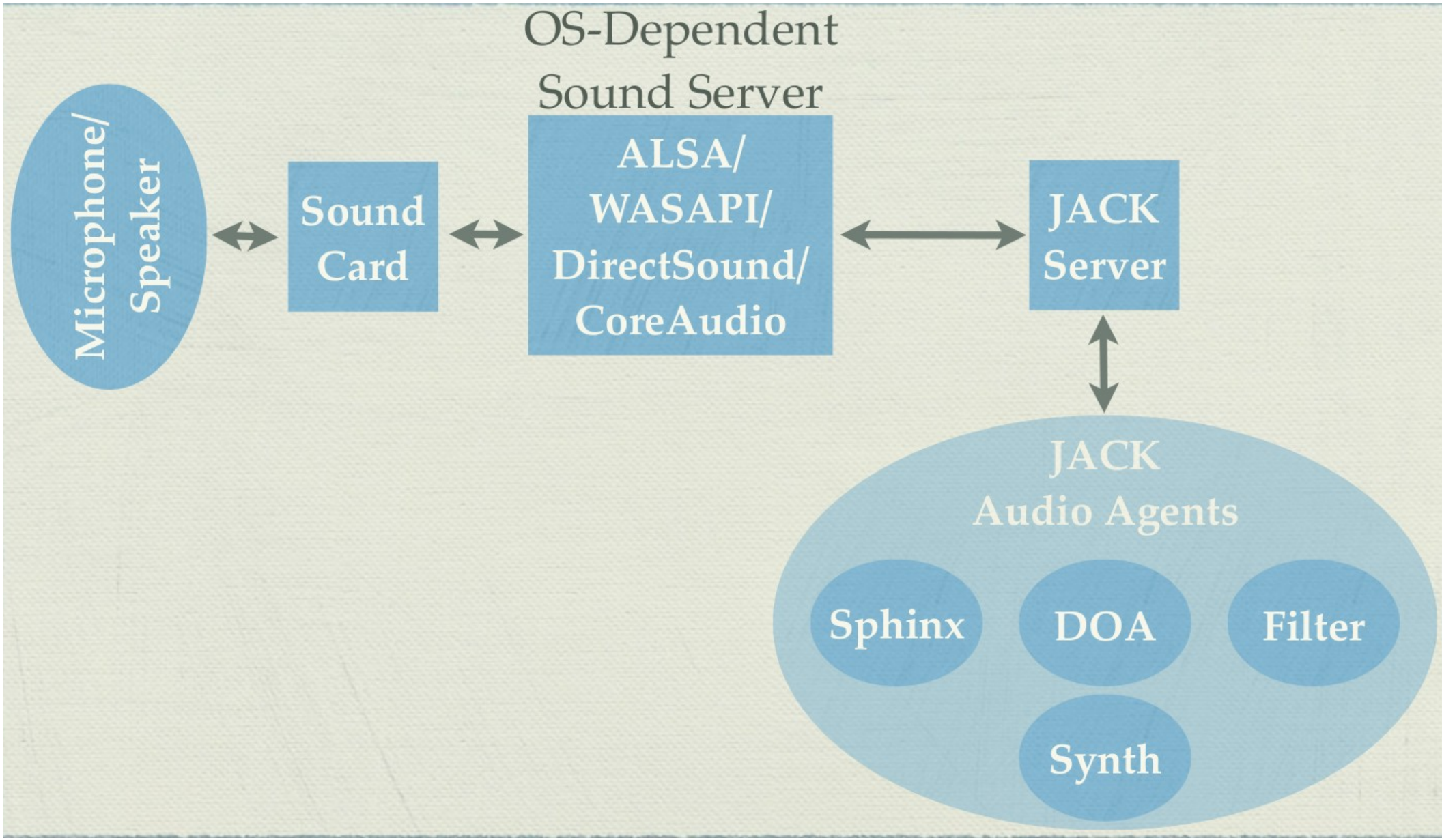
- Corran:

  sudo adduser <usuario> audio

Y para terminar...

- Reinicien su sesión para que los cambios se apliquen.
- Vuelvan a correr “groups” para verificar que su usuario ya está en el grupo “audio”.

Antes de proseguir,
algo de conceptos...


OS-Dependent
Sound Server
Microphone/
Speaker
Sound
Card
ALSA/
WASAPI/
DirectSound/
CoreAudio
JACK
Server
JACK
Audio Agents
Sphinx
DOA
Filter
Synth

JACK no es lo único

- Hay otros sistemas con los que se pueden capturar audio.
  - Por ejemplo, PulseAudio es la forma por defecto de Ubuntu.
- Y están corriendo en paralelo.
- Por lo que no se sorprendan si al subir/bajar el volumen con la interfaz de Ubuntu, no sube/baja el volumen para JACK.

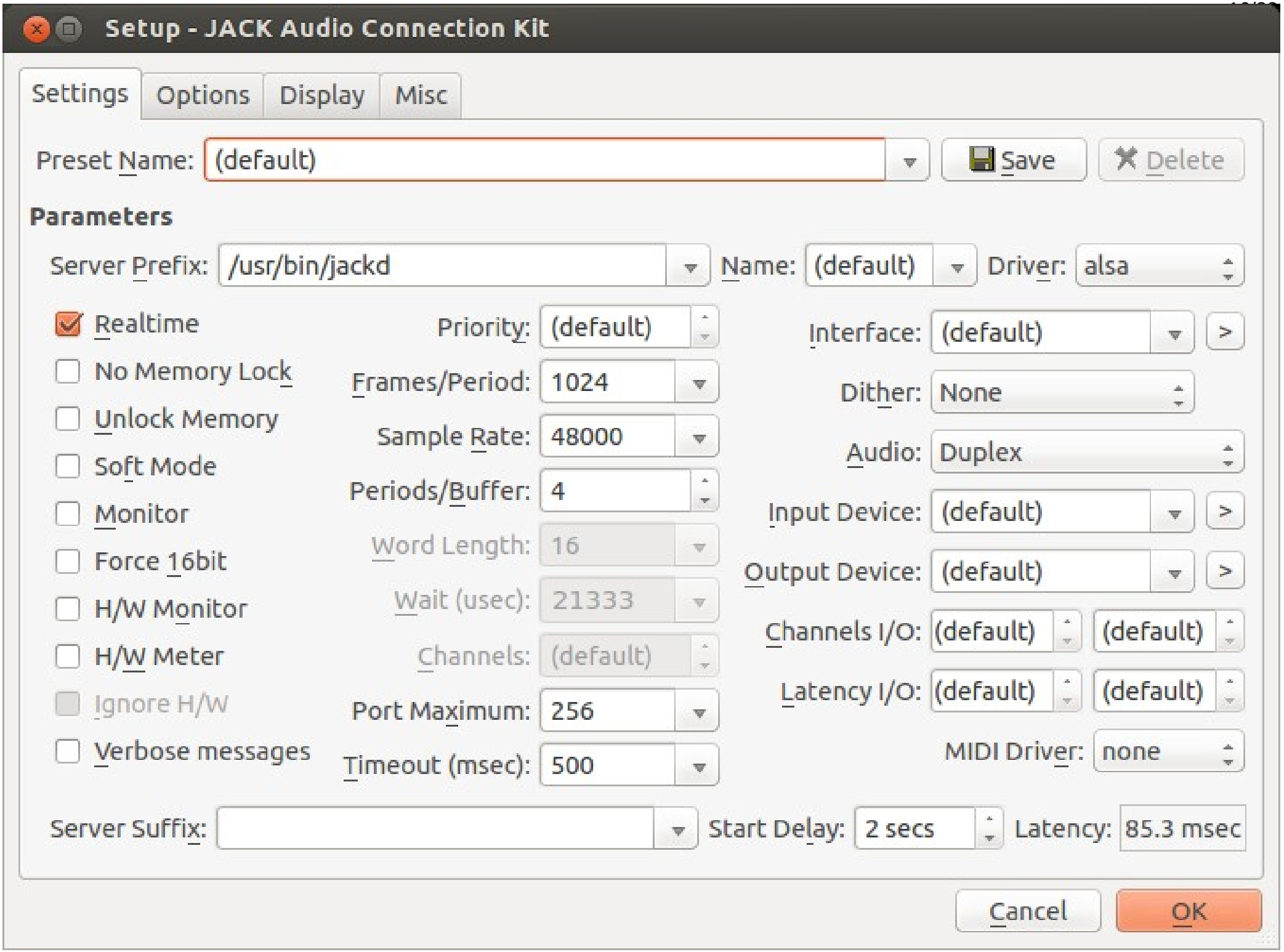
Bien.

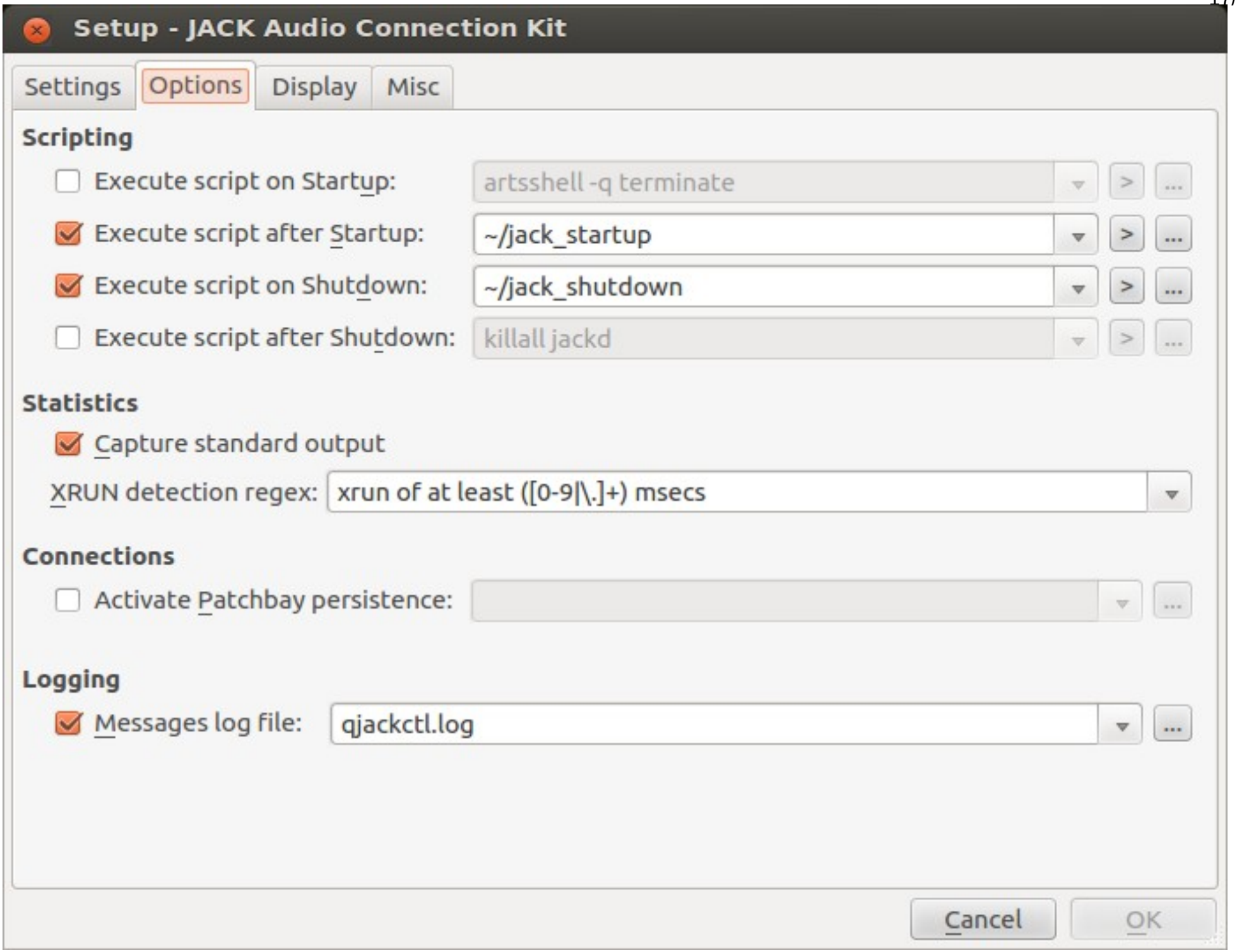
Vamos a correr a JACK con

qjackctl

QJackCtl


JACK Audio Connection Kit [external] Stopped.
Start
Stop
Stopped
0 (0)
Quit
Messages
Session
Setup...
Connect
Patchbay
About...


Setup - JACK Audio Connection Kit
Settings
Options
Display
Misc
Preset Name:
(default)
Save
Delete
Parameters
Server Prefix:
/usr/bin/jackd
Name:
(default)
Driver:
alsa
Realtime
No Memory Lock
Unlock Memory
Soft Mode
Monitor
Force 16bit
H/W Monitor
H/W Meter
Ignore H/W
Verbose messages
Priority:
(default)
Frames/Period:
1024
Sample Rate:
48000
Periods/Buffer:
4
Word Length:
16
Wait (usec):
21333
Channels:
(default)
Port Maximum:
256
Timeout (msec):
500
Interface:
(default)
Dither:
None
Audio:
Duplex
Input Device:
(default)
Output Device:
(default)
Channels I/O:
(default)
(default)
Latency I/O:
(default)
(default)
MIDI Driver:
none
Server Suffix:
Start Delay:
2 secs
Latency:
85.3 msec
Cancel
OK


Setup - JACK Audio Connection Kit
Settings
Options
Display
Misc
Scripting
Execute script on Startup:
artsshell -q terminate
Execute script after Startup:
~/jack_startup
Execute script on Shutdown:
~/jack_shutdown
Execute script after Shutdown:
killall jackd
Statistics
Capture standard output
XRUN detection regex:
xrun of at least ([0-9|\.]+) msecs
Connections
Activate Patchbay persistence:
Logging
Messages log file:
qjackctl.log
Cancel
OK

Configurar al Servidor JACK

- Cerciorar que el “Realtime” esté seleccionado.
- Periods/Buffer sea 4 (2 normalmente da problemas)
- Seleccionar al dispositivo de salida y entrada en “Input Device” y “Output Device”.
  - Utilizar los dipositivos generales como **“hw:0”** o **“hw:1”**, NO los específicos como “hw:1,0” o “hw:0,2”.
  - Recomendable que el de salida y entrada sea el mismo dispositivo.

Correr JACK

- Suban el volumen de las bocinas y de su micrófono.
  - JACK asume dichos niveles como sus máximos.
- Intenten correr JACK por medio de pulsar el botón de Inicio (*Start*).

JACK con PulseAudio

- De Ubuntu 16.04 en adelante, es posible que JACK se conecte automáticamente con PulseAudio.
  - Utilizando el paquete *pulseaudio-module-jack*
  - Convierte a todo PulseAudio a un agente de JACK
- Si sucede, en la configuración de PulseAudio (clic derecho en botón de audio de Ubuntu → Configuración) tiene como salida y entrada a:
  - Jack Sink
  - Jack Source
- Esta conexión no es esencial para nuestros propósitos, pero se aprecia.

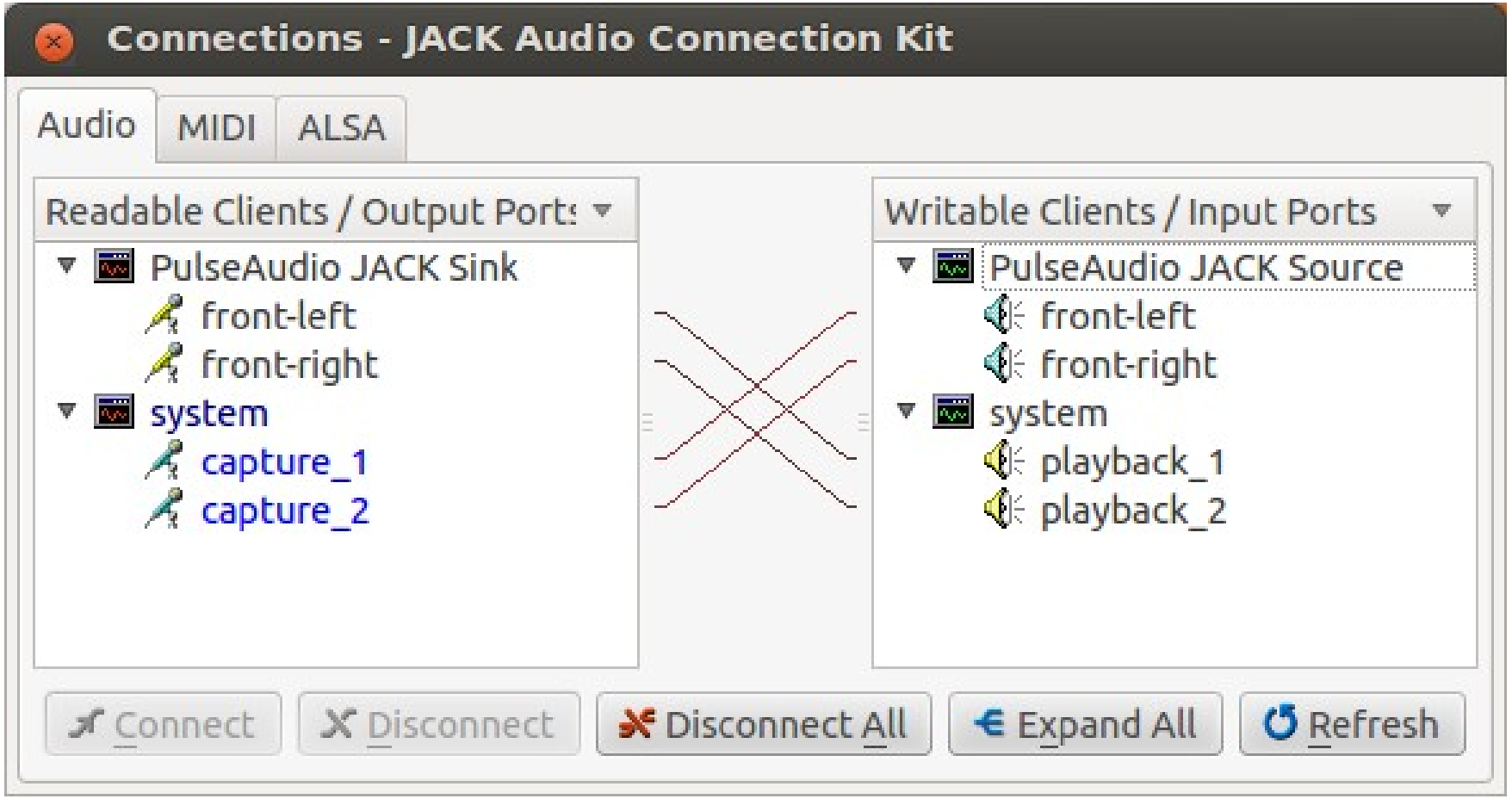
En caso de problemas...

- Es posible que los dispositivos seleccionados no sean válidos.
  - Utilizar otros.
- Es posible que PulseAudio no quiera prestar los dispositivos.
  - Reiniciar PulseAudio por medio de:

    pulseaudio --kill

    pulseaudio --start

QjackCtl → Connect



Probar Salida

- En una terminal corran:
  - jack_simple_client
- Deben escuchar un tono.
- Observen que en la ventana de “Connect” de QjackCtl se ha agregado un nuevo agente.
- Presionar CTRL + C en la terminal cuando se harten del tono.

Probar Entrada

- En la ventana de Connect.
- De la columna izquierda, seleccionen uno de los puertos bajo “system”, que sus nombres comienzan con “capture”.
- De la columna derecha, seleccionen uno de los puertos bajo “system”, que sus nombres comienzan con “playback”.
- Presionen el botón “Connect”.
- ¿Se escuchan?

Por cierto...

- ¡Felicidades!
- Ya están procesando audio.
  - Muy simple, pero se vale aplaudir.

Ahora, a crear nuestro primer agente de JACK

Agentes de JACK

- Cada agente se comunica con JACK por medio de escribir y/o leer valores de energía en arreglos de datos.
- Dichos arreglos representan *ventanas* de audio.

Agentes de JACK

- La comunicación agente-servidor se hace de por medio de una función *callback*.
  - Son funciones especiales que se hacen accesibles al exterior del programa.
  - Y se pueden correr desde otros programas como forma de comunicación.

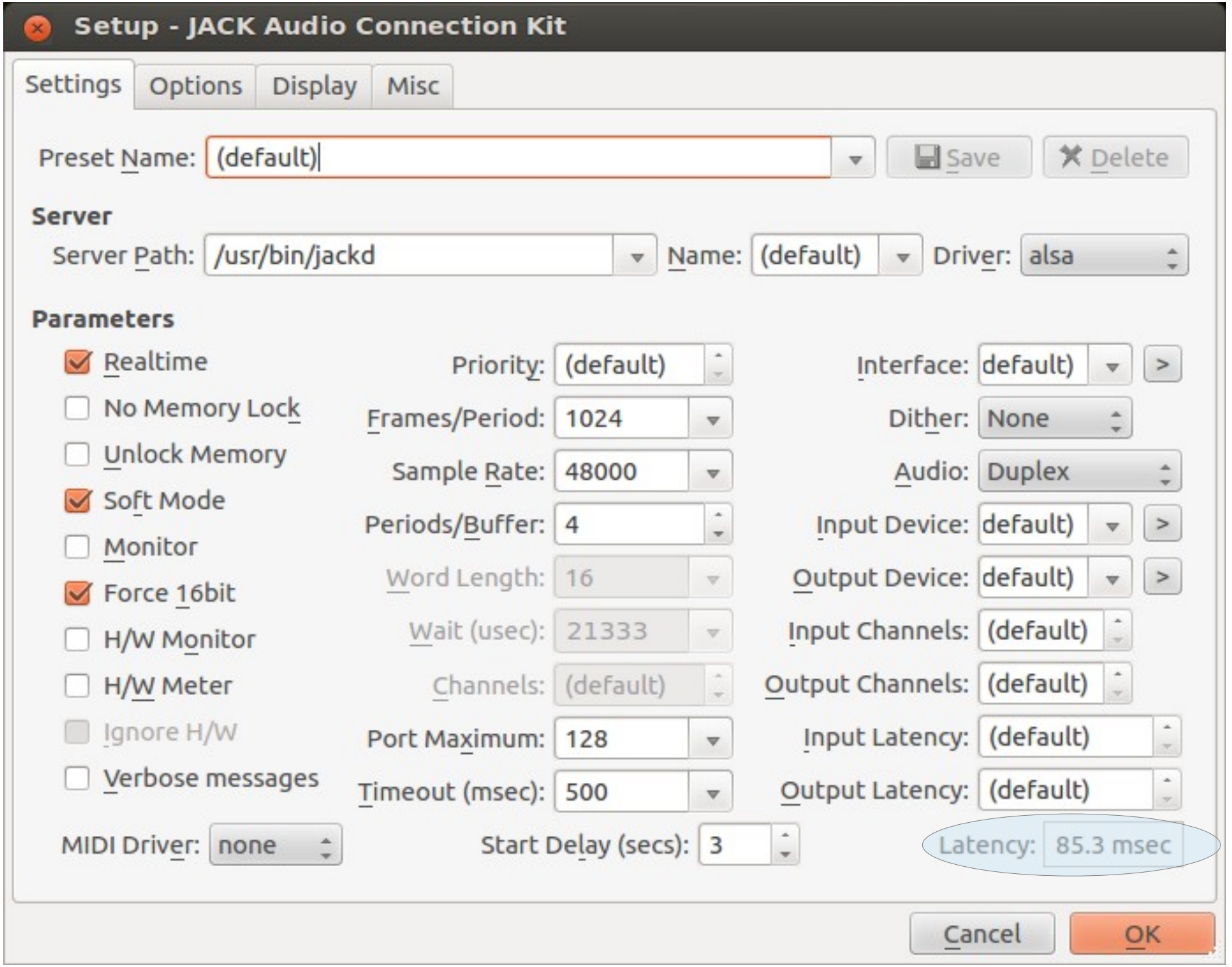
```
void func_callback(int *a){
    *a = 1;
}

int main(){
    /* codigo que determina a func_callback como nuestro callback */

    while(1){
        /* codigo que hace otra cosa */
    }
}
```

Agentes de JACK

- Todo agente de JACK tiene que tener una función callback a la que el servidor JACK puede mandar a llamar en cada *ciclo de audio*.
  - Un ciclo de audio es el intervalo de tiempo que el servidor JACK permite a todos sus agentes trabajar en sus callbacks correspondientes.
  - Usualmente es limitado por la cantidad de tiempo de “respuesta” (*latency*) que tiene que garantizar para que sea tiempo real.


Setup - JACK Audio Connection Kit
Settings
Options
Display
Misc
Preset Name: (default)
Save
Delete
Server
Server Path: /usr/bin/jackd
Name: (default)
Driver: alsa
Parameters
Realtime
No Memory Lock
Unlock Memory
Soft Mode
Monitor
Force 16bit
H/W Monitor
H/W Meter
Ignore H/W
Verbose messages
Priority: (default)
Frames/Period: 1024
Sample Rate: 48000
Periods/Buffer: 4
Word Length: 16
Wait (usec): 21333
Channels: (default)
Port Maximum: 128
Timeout (msec): 500
Interface: default)
Dither: None
Audio: Duplex
Input Device: default)
Output Device: default)
Input Channels: (default)
Output Channels: (default)
Input Latency: (default)
Output Latency: (default)
MIDI Driver: none
Start Delay (secs): 3
Latency: 85.3 msec
Cancel
OK

Agentes de JACK

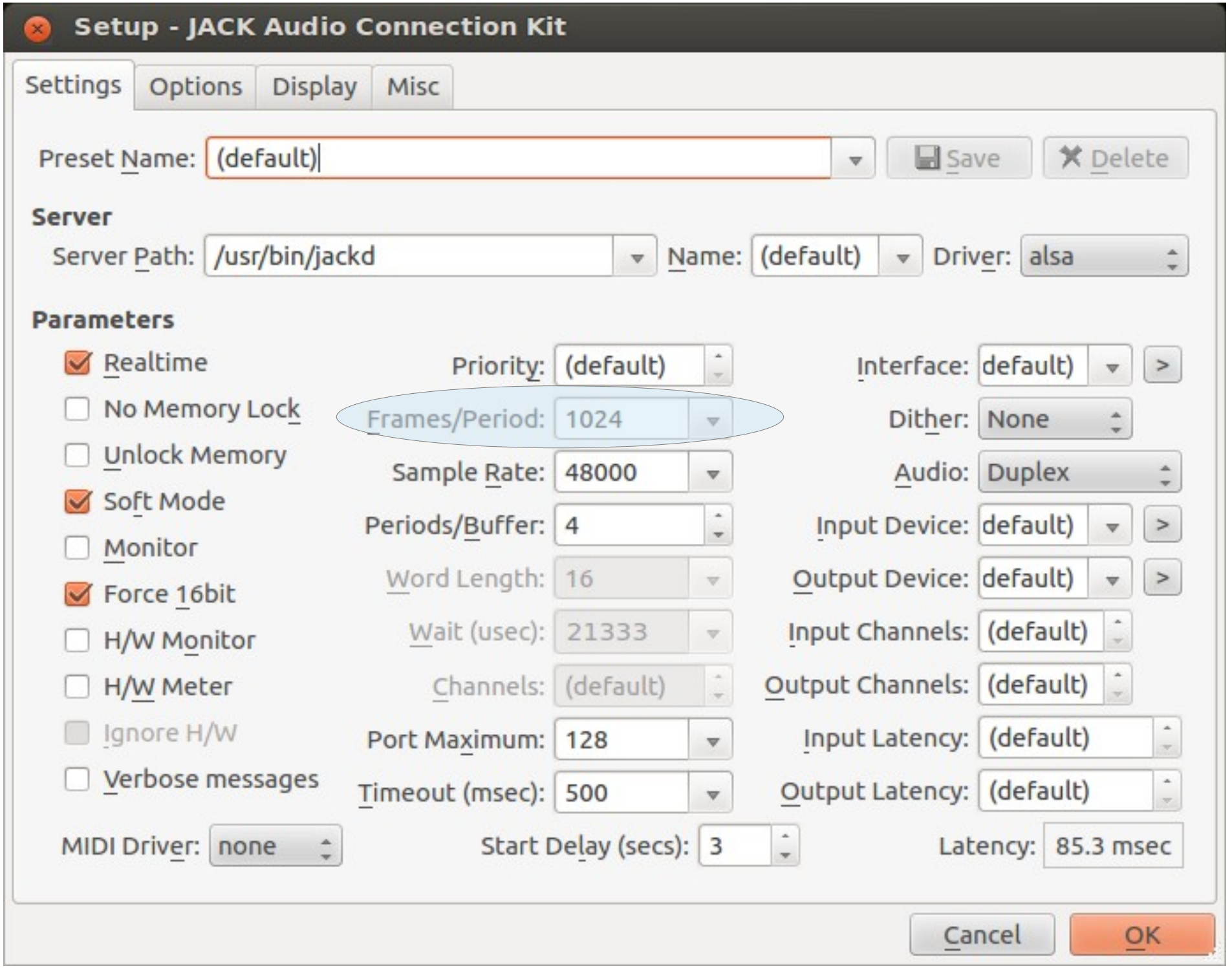
- Aquellos que no respondan en el tiempo propuesto (*latency*), se levanta un error de “overrun”.
- Esto significa que la información en las salidas del agente tardío es ignorada en el siguiente ciclo de audio.

Ciclo de Audio

- En cada ciclo de audio, el servidor JACK manda a llamar a la función callback de TODOS los agentes JACK conectados.
- En dicha función, los agentes de JACK pueden acceder a los datos de audio en su entrada y/o escribir información en su salida.

Ciclo de Audio

- En cada ciclo de audio, el servidor JACK provee a sus agentes buffers de datos de los cuales se puede:
  - Leer, si es buffer de entrada.
  - Escribir, si es buffer de salida.
- El tamaño de dichos buffers es configurado en el servidor JACK.


Setup - JACK Audio Connection Kit
Settings
Options
Display
Misc
Preset Name: (default)
Save
Delete
Server
Server Path: /usr/bin/jackd
Name: (default)
Driver: alsa
Parameters
Realtime
No Memory Lock
Unlock Memory
Soft Mode
Monitor
Force 16bit
H/W Monitor
H/W Meter
Ignore H/W
Verbose messages
Priority: (default)
Frames/Period: 1024
Sample Rate: 48000
Periods/Buffer: 4
Word Length: 16
Wait (usec): 21333
Channels: (default)
Port Maximum: 128
Timeout (msec): 500
Interface: default)
Dither: None
Audio: Duplex
Input Device: default)
Output Device: default)
Input Channels: (default)
Output Channels: (default)
Input Latency: (default)
Output Latency: (default)
MIDI Driver: none
Start Delay (secs): 3
Latency: 85.3 msec
Cancel
OK

¿Frames? ¿Periodos?

- Los buffers son conocidos como *periodos*.
- Los datos dentro los buffers son conocidos como *frames*.

- ADVERTENCIA: la opción de Periods/Buffers habla de los buffers internos de JACK, no de los buffers que provee a los agentes.
  - A los desarrolladores de JACK les gusta confundir a la gente.

Código Ejemplo

- Para proseguir, es más fácil apoyarnos en un ejemplo simple de un agente de JACK.
- Bajarlo de la página del curso, ó:

  http://calebrascon.info/jack/jack_in_to_out.c
- Vamos a revisar sus partes...
- Se recomienda tener a la mano:

  https://jackaudio.org/api/

Creación del Agente

- Se hace uso de *jack_client_open*
- Requiere de:
  - El nombre del agente.

Creación del Agente

- Opciones de comienzo:
  - El más común es *JackNoStartServer* que le dice a la función que no comience al servidor JACK si no está comenzado (lo cual hace por defecto).
  - Otra posible opción es *JackNullOption*, que no manda ninguna opción que no sea las de por defecto.
  - Más opciones en:

    https://jackaudio.org/api/types_8h.html#a396617de2ef101891c51346f408a375e

Creación del Agente

- Un apuntador a una variable que describirá el estado de la creación del agente:
  - Si el resultado de la función es apuntador a NULL, significa que hubo un error. El apuntador del “estado” da una mejor descripción de dicho error.
  - Tipos de Error:

    https://jackaudio.org/api/types_8h.html#aaf80297bce18297403b99e3d320ac8a8
  - Se revisan de la siguiente manera:

```
jack_client_t *client;

int main(){
    jack_options_t options = JackNoStartServer;
    jack_status_t status;

    client = jack_client_open (“name”, options, &status);

    if (status & JackServerFailed){
        ...
    }

    if (status & JackNameNotUnique){
        ...
    }
}
```

Importante: no es doble &, es sólo un &.

Un & por sí sólo hace una operación binaria, el cual regresa un valor diferente a 0 si son equivalentes en un bit específico.

Creación del Agente

- (Opcional) El nombre del servidor de JACK.
  - Útil si hay varios servidores corriendo a la vez.
  - En nuestro caso, se puede ignorar este argumento.

Nombre de Agente

- *jack_client_open* intenta asignarle el nombre a nuestro agente que le dimos.
- Es posible que dicho nombre ya esté ocupado, si fuera éste el caso:
  - El bit de *JackNameNotUnique* nos lo hace saber.
  - La función nos asigna un nombre similar.
  - *jack_get_client_name* nos regresa el nombre que se nos ha asignado.

Callback

- *jack_set_process_callback* define cual función es la que el servidor JACK va a mandar a llamar para este agente.
- En este caso, nuestra función callback se llama: “jack_callback”
- Recibe dos argumentos:
  - El número de frames que hay en los buffers.
  - Un argumento adicional para otros datos misceláneos.

Callback

- Este es donde hacemos la mayor parte del trabajo, por lo que vamos a dejar la discusión de esta parte del código para el final.

On Shutdown

- *jack_on_shutdown* le hace saber a JACK cual función mandar a llamar cuando:
  - El servidor JACK se muere.
  - Cuando el agente requiere ser desactivado.
- Requiere de otra función callback: “jack_shutdown”
  - Lo único que hace es matar al agente.
    - exit(0) forza al agente a terminar, regresando un valor de 0 al sistema operativo (equivalente a return 0 del main).

Registro de Puertos

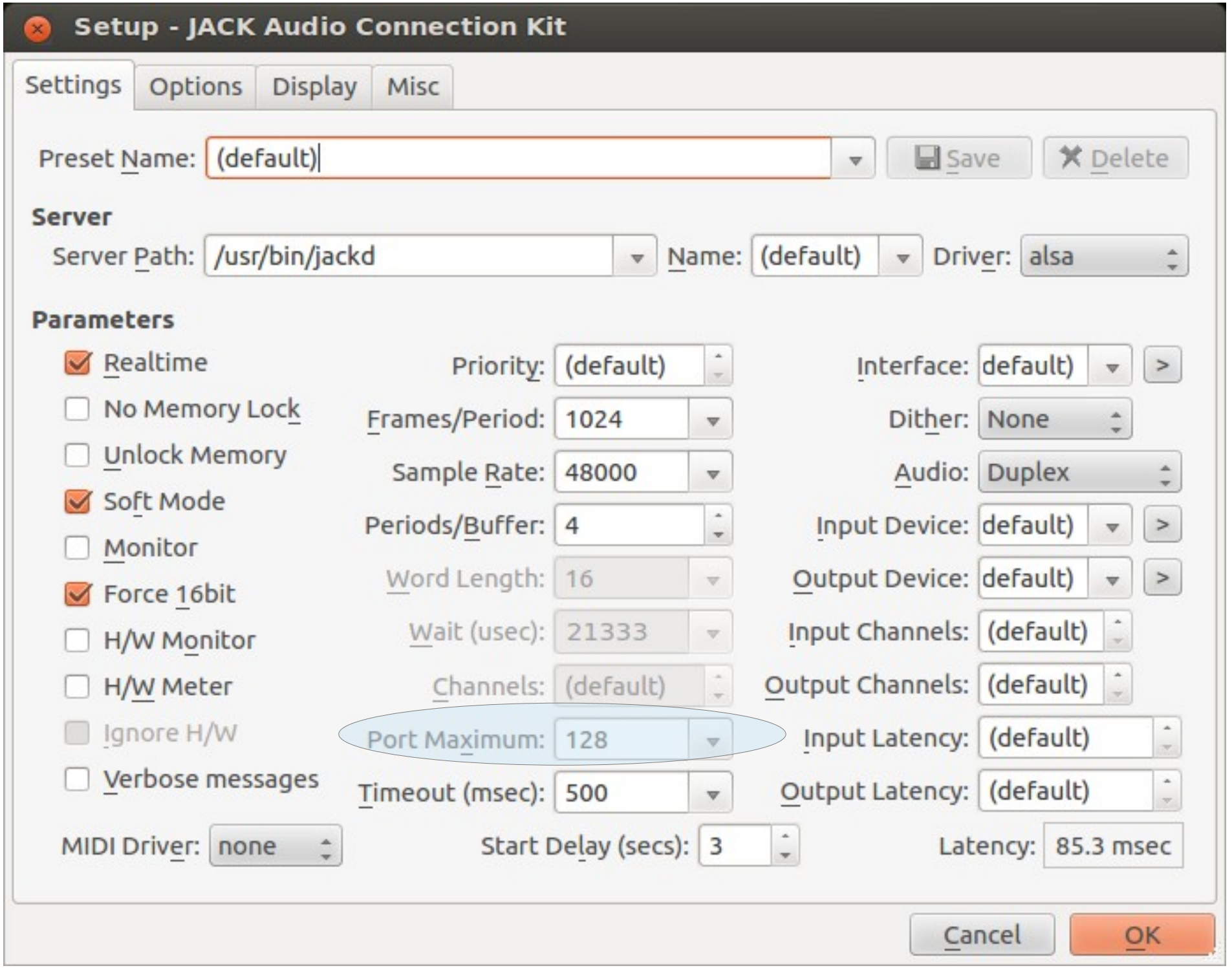
- Todo agente de JACK requiere por lo menos un puerto registrado.
- En nuestro caso, tenemos uno de salida y otro de entrada.
- Los registramos por medio de: *jack_port_register*

jack_port_register

- Requiere de:
  - La variable del cliente creada por *jack_client_open*
  - El nombre del puerto
  - El tipo de dato de audio que va a manejar
    - Usualmente *JACK_DEFAULT_AUDIO_TYPE*
  - Banderas descriptivias del puerto
    - *JackPortIsInput*: si es de entrada
    - *JackPortIsOutput*: si es de salida

jack_port_register

- El último argumento es para el tamaño del buffer, pero es sólo para puerto no-nativos. Se ignora con puertos nativos.
  - Todos los puertos que vamos a manejar son nativos.
  - Se puede dejar en 0.
- Regresa el apuntador que describe al puerto.
  - Hay una cantidad máxima de puertos que JACK puede manejar.
  - Si no se pudo registrar un puerto porque ya no hay más puertos que JACK pueda manejar, el apuntador que *jack_port_register* regresa es igual a NULL.


Setup - JACK Audio Connection Kit
Settings
Options
Display
Misc
Preset Name: (default)
Save
Delete
Server
Server Path: /usr/bin/jackd
Name: (default)
Driver: alsa
Parameters
Realtime
No Memory Lock
Unlock Memory
Soft Mode
Monitor
Force 16bit
H/W Monitor
H/W Meter
Ignore H/W
Verbose messages
Priority: (default)
Frames/Period: 1024
Sample Rate: 48000
Periods/Buffer: 4
Word Length: 16
Wait (usec): 21333
Channels: (default)
Port Maximum: 128
Timeout (msec): 500
Interface: default)
Dither: None
Audio: Duplex
Input Device: default)
Output Device: default)
Input Channels: (default)
Output Channels: (default)
Input Latency: (default)
Output Latency: (default)
MIDI Driver: none
Start Delay (secs): 3
Latency: 85.3 msec
Cancel
OK

Activación del Agente

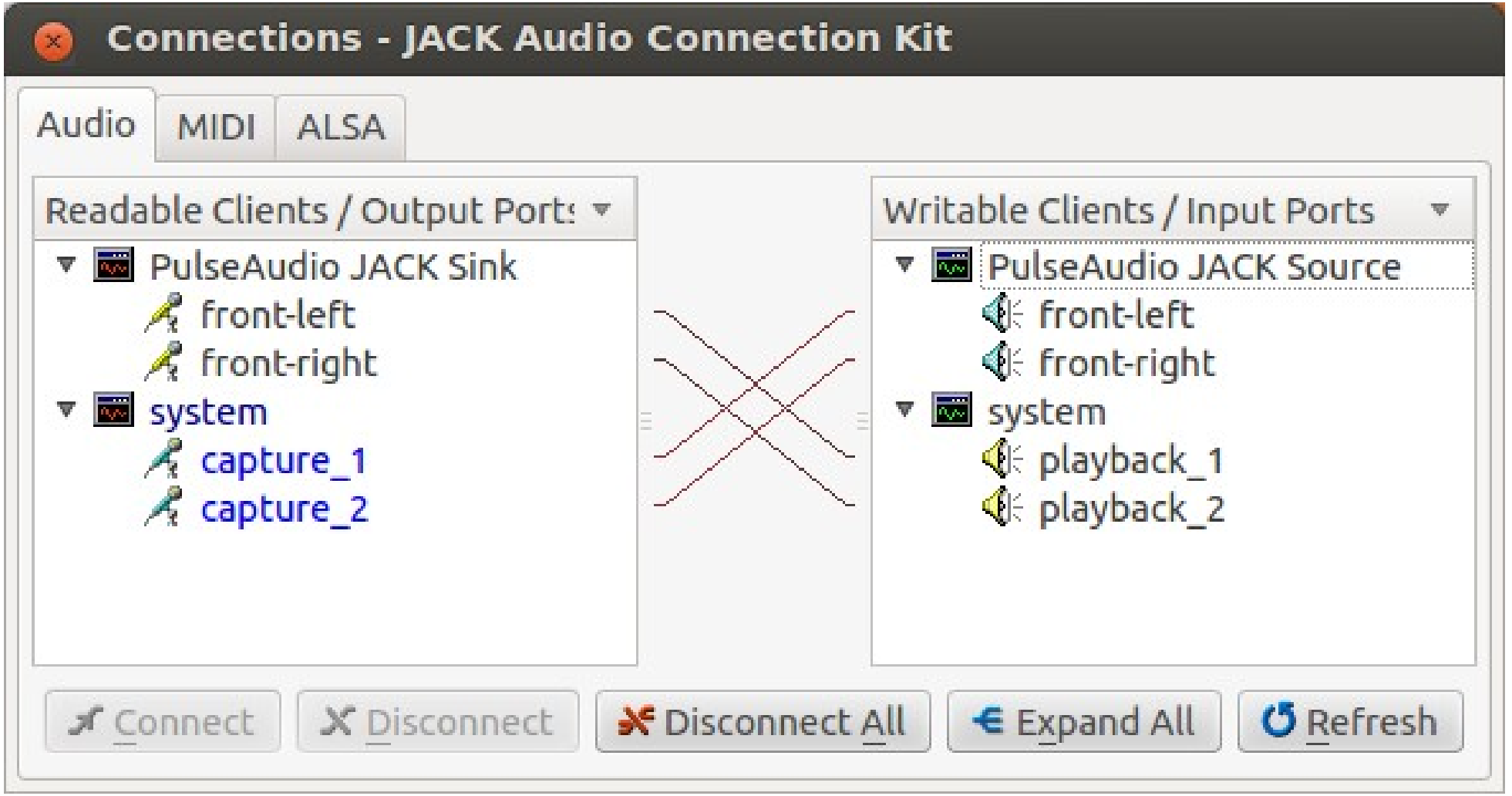
- *jack_activate* activa al agente.
- Si la activación es exitosa (si jack_activate regresa 1), entonces el agente es agregado a la lista de agentes del servidor JACK.
  - Y el servidor JACK comienza a llamar a su función callback correspondiente.
- Se pueden registrar puertos después de haber activado el cliente, pero no es recomendable.

Conexión de Puertos

- Tras haber activado el cliente, se puede conectar sus puertos a otros agentes.
- Se conectan salidas de agentes a entradas de agentes, en ese orden.
  - No es posible conectar salidas con salidas o entradas con entradas.
- Se puede hacer de manera manual:
  - En la interfaz “Connect” en QjackCtl.
  - O con los comandos de terminal jack_connect

SALIDAS

ENTRADAS



jack_connect

- *jack_connect* requiere de los nombres de los puertos a conectar.
  - Primero el de salida, y luego el de entrada.
- El formato del nombre de un puerto para *jack_connect* es:

  <nombre de agente>:<nombre de puerto>

  - Por ejemplo system:playback_1

Conexión de Puertos

- También podemos hacerlo desde C, por medio de la función *jack_connect* que manda a llamar el programa *jack_connect*.
  - Valga la redundancia.
- Requiere:
  - La variable del cliente creada por *jack_client_open*
  - El nombre del puerto de salida
  - El nombre del puerto de entrada
- Regresa un 1 si fue exitoso, 0 si no.

Conexión de Puertos

- Para obtener nombres de puertos disponibles, usamos *jack_get_ports*
- Requiere:
  - La variable del cliente creada por *jack_client_open*
  - Un patrón del nombre del puerto a buscar
  - Un patrón del tipo de datos de audio que utiliza el puerto a buscar
  - Banderas descriptivas del puerto a buscar

jack_get_ports

- Usualmente queremos que nuestro agente se conecte al micrófono y bocinas.
  - Para JACK, éstos son los puertos del agente “system”, también conocidos como “puertos físicos”.
- Para buscar estos puertos es más fácil utilizar la bandera descriptiva: *JackPortIsPhysical*
- Así como añadirle la bandera descriptiva de que si es de entrada o de salida: *JackPortIsOutput* ó *JackPortIsInput*.

jack_get_ports

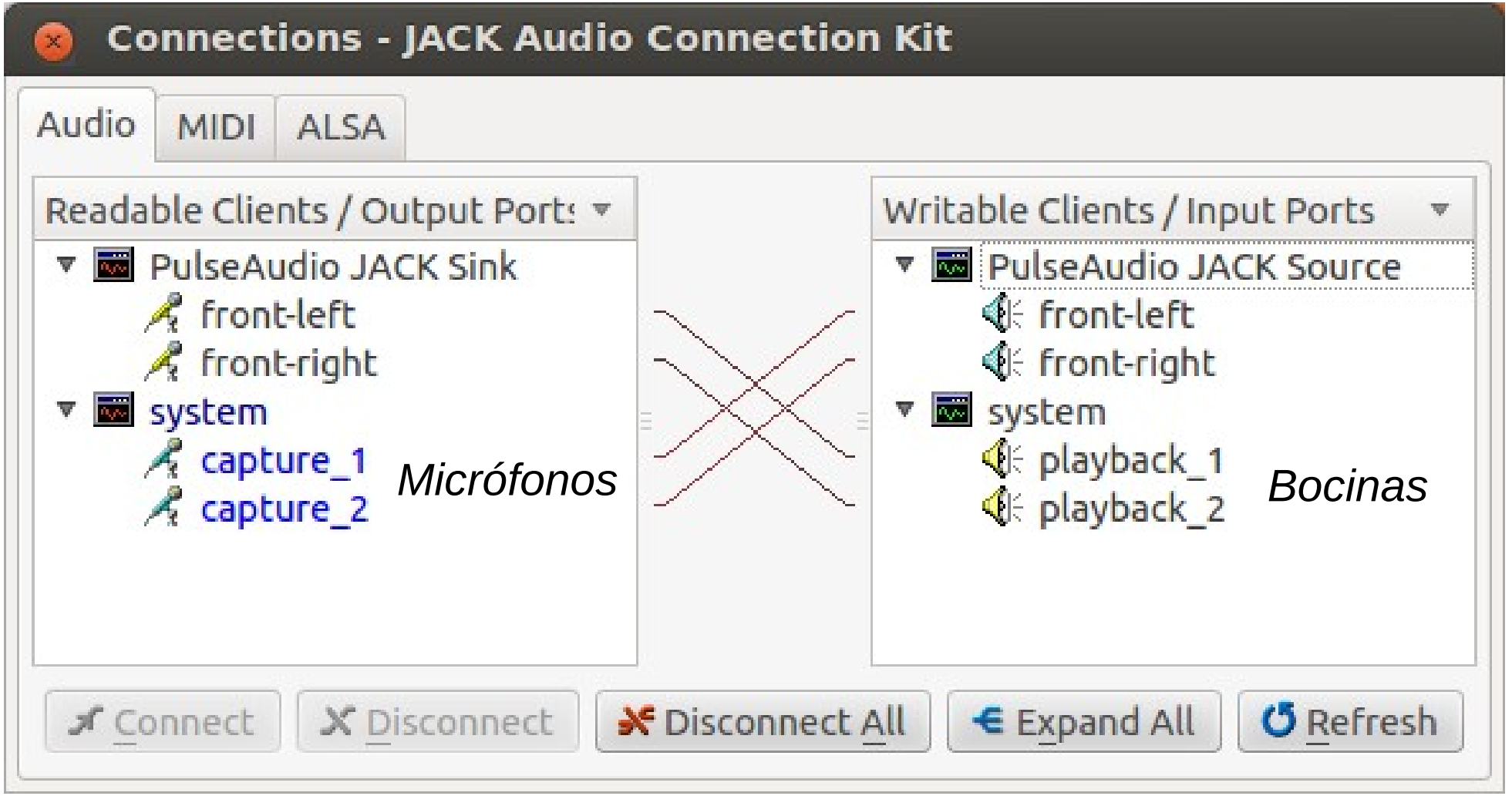
- Regresa una serie de strings con los nombres de los puertos que cumplen con las características que le indicamos.
- Si no hay puertos con esas características, regresa NULL.

Salidas a Entradas … físicas

- *De nuevo:* siempre se debe conectar una salida a una entrada.
- *De nuevo:* los puertos físicos son presentados a los agentes como los puertos del agente “system”.
  - Una **salida** del agente sistema es un **micrófono**.
  - Una **entrada** del agente sistema es una **bocina**.
  - No es un typo...

SALIDAS

ENTRADAS



Salidas a Entradas … físicas

- Por lo tanto, debemos conectar:
  - Nuestras salidas a las entradas de “system” (bocinas).
  - Las salidas de “system” (micrófonos) a nuestras entradas.

¿Cierre del Agente?

- Ya teniendo todo listo, es sólo cuestión de esperar a que el servidor JACK mande a llamar nuestro callback.
- ¿Mientras tanto?
  - Hacemos nada.

Sleep

- La función sleep hace que el programa duerma por una cantidad de segundos específica.
  - Viene en la librería unistd.h, por eso el include.
- Requiere:
  - Número de segundos. Si este número es -1, se duerme indefinidamente.

¿“Dormir”?

- Es posible que en vez de utilizar sleep, pusieramos un while(1){}
- Pero, el programa estaría gastando recursos al estar verificando la condicional en cada ciclo de operación.
- Además, “dormir” es sólo un decir, el agente sigue alerta de cualquier callback.
  - El callback funciona como un despertador.

jack_client_close

- Esta parte del código realmente es de adorno, ya que del sleep(-1) en adelante, nada se corre.
  - Ya que es indefinido.
- Pero lo ponemos ahí por completitud.

Callback... de nuevo

- Dijimos que íbamos a volver al tema del callback.

- *jack_port_get_buffer* es una función que regresa un apuntador a un buffer de datos de audio de algún puerto.
  - Dicho buffer puede ser de entrada o salida, dependiendo del tipo de puerto que se especifique.

jack_port_get_buffer

- Requiere:
  - La variable del puerto creada por *jack_port_register*
  - De cuantos frames se quiere el buffer
- El apuntador apunto a un arreglo de tipo *jack_default_audio_sample_t*
  - Este tipo es casi igual a un tipo double, y se puede utilizar como tal.
  - Sus valores van del -1 a 1.
- Siendo un arreglo, se puede acceder como cualquier arreglo: in[0], out[4], etc.

Buffers

- Si el buffer es de un puerto de entrada:
  - Leer de éste proporciona los valores de energía de una señal capturada por nuestro agente.
  - Escribir a éste no tiene ningún efecto.
- Si el buffer es de un puerto de salida:
  - Escribir a éste proporciona valores de energía de una señal reproducida por nuestro agente.
  - Leer de éste es basura y no tiene ningún efecto.

Recordatorio

- Una variable de tipo *jack_port_t* es sólo un **descriptor** de un puerto.
  - No se puede leer/escribir a éste por si sólo.
- Para esto, se requiere:
  - Obtener un buffer del puerto con *jack_port_get_buffer*
    - El buffer será del tipo *jack_default_audio_sample_t*
  - Y, luego, leer/escribir al buffer.

memcpy

¿Quién puede decirme qué es lo que hace?

memcpy

- Copia datos de un segmento a memoria a otra.
- Recibe de argumentos:
  - Apuntador al arreglo destino.
  - Apuntador al arreglo origen.
  - Cantidad de bytes a copiar.
    - Se puede usar “sizeof” para saber el número de bytes que ocupa un tipo de variable, y luego multiplicar por el tamaño del arreglo

Entonces:

¿Qué es lo que hace este agente?

Compilar

- El comando completo es:

  gcc -o jack_in_to_out jack_in_to_out.c -ljack

- Si les sale varios warnings que involucran a ISO C90:
  - Utilicen la opción -std=gnu99 justo después de gcc

¿Cómo le harían para hacer eso utilizando un bucle tipo “for”?

¿Qué necesitan hacer para subirle el volumen a la salida?

¿Para bajarle el volumen?

Volumen

Si multiplico por 2 la salida, ¿suena el doble de fuerte?

Volumen

- El oído humano no funciona de una manera lineal.
  - Es más logarítmico.
- Una manera muy popular de representar relaciones entre energía que son comparables con el oído humano es utilizando el concepto de **Decibeles**.

Decibel (dB)

- A pesar de ser *incorrectamente* considerado como una medida de energía, es realmente una medida de **razón de energía**.

$$L = 20 \log_{10} \left(\frac{P_f}{P_r} \right)$$

L = razón de energía (dB)
P_f = energía final
P_r = energía de *referencia**

***Una *referencia* muy utilizada es 20 micropascales en el aire (la intensidad más pequeña que un humano puede detectar).**

Decibel (dB)

- Por lo tanto, si queremos incrementar el volumen por una razon L, se puede hacer por medio de:

$$\frac{P_f}{P_r} = 10^{\frac{L}{20}}$$

L = razón de energía (dB)
P_f = energía final
P_r = energía original

Ejercicios/Tareas

¿Cómo le harían para...? Tarea 1

- Hacer que el ejemplo registre dos puertos de entrada y dos de salida.
- Conecte apropriadamente a los puertos del agente “system”.
- Copie la información de la entrada #1 a la salida #1, así como de la entrada #2 a la salida #2.

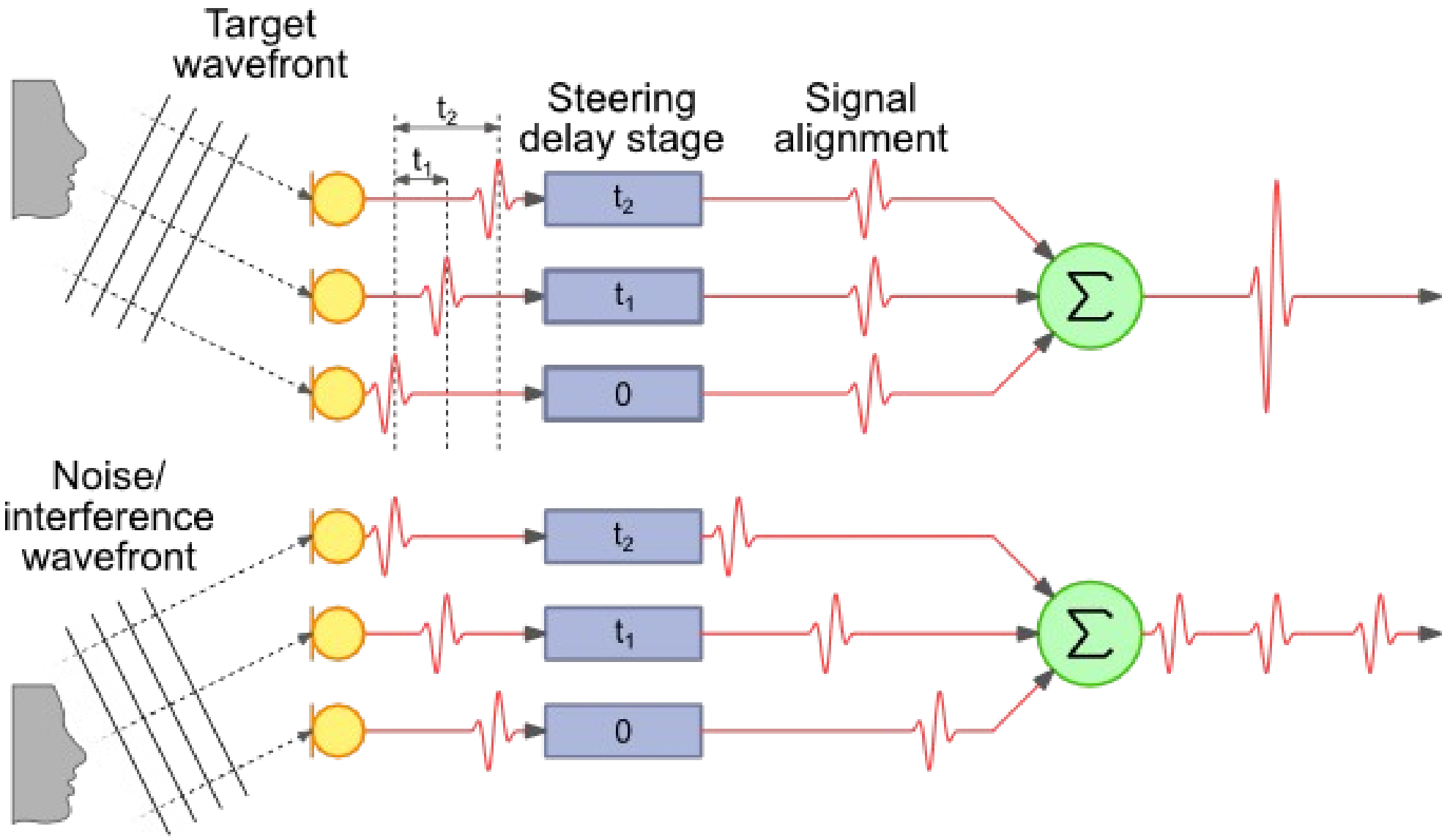
¿Cómo le harían para...?
Tarea 2

- Hacer que el ejemplo registre dos puertos de entrada y uno de salida.
- Conecte apropriadamente a los puertos del agente “system”.
- Sumar punto-a-punto la información de la entrada #1 y la de la entrada #2, y sacar el resultado por la salida.

¡Felicidades!

- Han implementado su primer *beamformer*.
  - También conocidos como filtros espaciales.
  - Reducen la energía de todo lo que viene de una dirección de interés pre-establecida.
- El beamformer que implementaron tiene como dirección de interés a cero grados (enfrente).
  - Todo lo que venga en esa dirección se le duplicará su energía.
  - Lo que no venga de esa dirección, no se le hará nada.

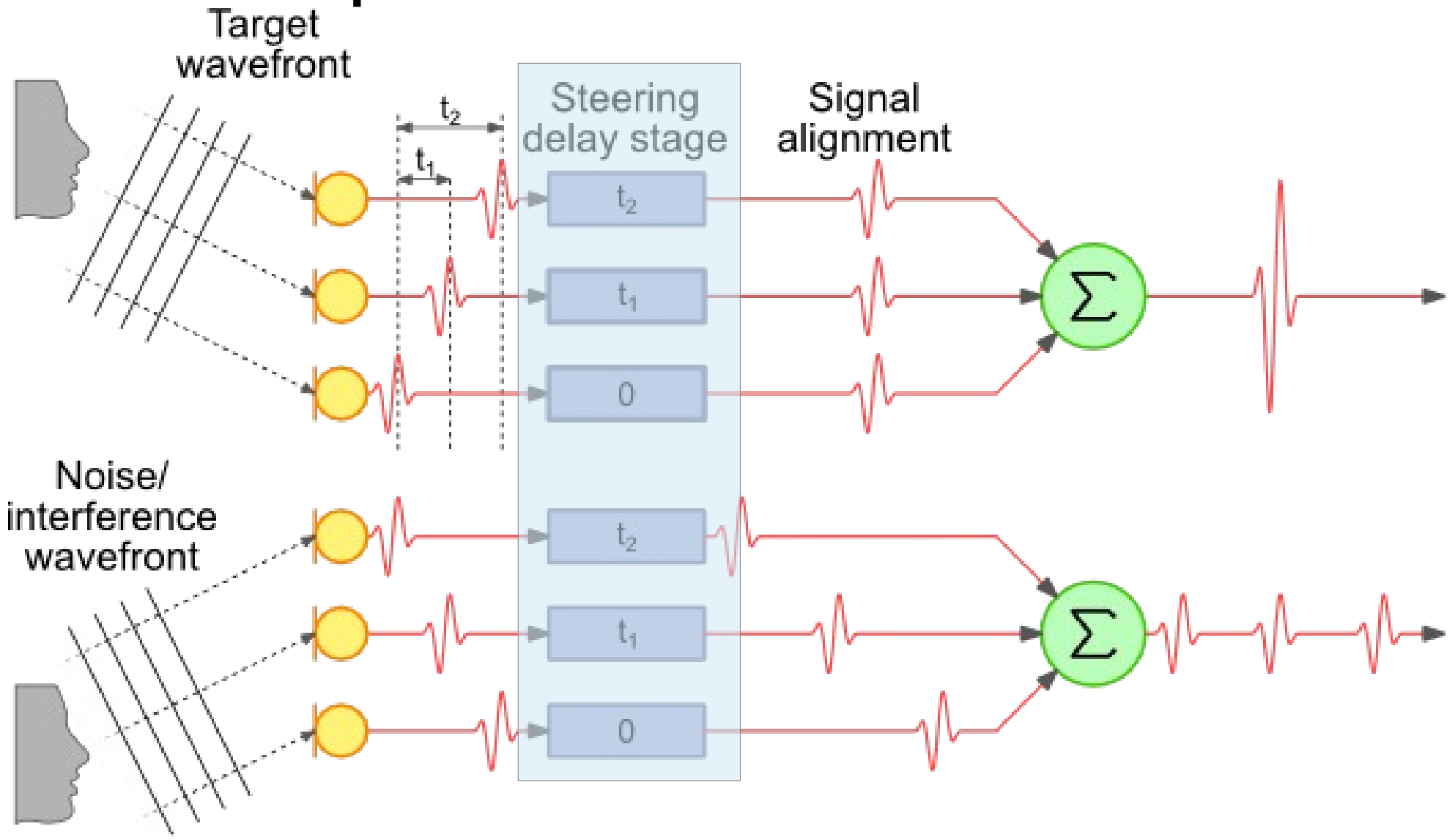
Algo así...


Target wavefront
Steering delay stage
Signal alignment
t_2
t_1
t_2
t_1
0
Σ
Noise/ interference wavefront
t_2
t_1
0
Σ

Pero...

- ¿Cómo cambio la dirección de interés?

Lo vamos a ver después, pero tiene que ver con desfases



Entonces...

- Es muy importante que sepamos como desfasar señales.
- Por lo tanto...

Siguiente Clase

Desfase con JACK y Baudline