Universidad Nacional Autónoma de México

Facultad de Ingeniería

División de Ingeniería Eléctrica

Sistemas Operativos

Grupo 06

2025 - 2

Listeners, eventos y respuestas en lenguajes y sistemas

Alumnos

Eric Ramírez Valdovinos (423095203)

Erick Nava Santiago (320298608)

Profesor

Gunnar Eyal Wolf Iszaevich

11 de mayo de 2025

Contenido

Concepto de listener

En palabras sencillas, podemos definir a un *listener* como un hilo (proceso) se dedica a esperar, a escuchar, como su nombre lo sugiere. Con esto nos podríamos hacer a la idea de que es un mecanismo pasivo que tiene la capacidad de aguardar a que un evento ocurra para entonces realizar otra serie de acciones, también podríamos considerar está acción combustible como una solicitud que nuestro *listener* tendrá que atender.

La función principal de un *listener* es variable, puesto que, depende de la clasificación y el entorno en el cual estemos trabajando, pero en términos generales permite comunicar a la computadora con el usuario y al usuario con la computadora.

Breve historia de los listeners

Para dar contexto acerca de cuándo se empezó a utilizar este concepto, no podemos definir un momento de invención, como si se tratara de un lanzamiento de servicio o algún producto, así que si queremos buscar en el tiempo como estandarizó su uso tendremos que seguir a todas las prácticas que involucren esperar y reaccionar ante eventos.

La idea surgió desde las necesidades que presentaban las primeras computadoras, las cuales trabajaban realizando una tarea a la vez y para utilizar dispositivos externos, estos enviaban una interrupción al proceso principal, saltando a una labor distinta para finalmente volver a la etapa dónde se detectó la interrupción. Posteriormente, esto dio paso a lo que conocemos como manejo de entradas múltiples.

Con el tiempo, los inicios del internet y el desarrollo de redes como ARPANET, introdujo la necesidad de escuchar no solo proceso internos de la computadora, sino también conexiones por parte de otras máquinas; empezaron a surgir estándares para cómo los programas le indicarán al sistema operativo que tenía que estar alerta de alguna conexión de red para cumplir su proceso. Cuando un programa está escuchando un socket, quiere decir que está esperando alguna conexión.

Sería imposible, no hablar de las GUI´s si el tema son listeners, puesto que, para dejar atrás los comandos y volver al cómputo más sencillo para todos fue necesaria la inclusión de movimientos de ratón, tecleos, clics, etc. de esta forma, los *events loops* surgieron para responder a las peticiones del usuario en todo momento, escuchando los eventos de hardware de periféricos, añadiendoles a una cola que la aplicación en cuestión tendrá que manejar para manejar estás interrupciones, pero controladas sin que interrumpa a otros procesos de la máquina.

Ya entrados los años 90´s nos encontramos con el auge del paradigma de programación POO, encontramos al patrón de diseño observador como una certeza de que el concepto ya era un estándar en muchas áreas del cómputo. Por ejemplo, Java (del que hablaremos más adelante) utilizó este patrón como un mecanismo para manejar eventos dentro de sus interfaces gráficas.

Hoy en día se escuchan múltiples entradas y procesos en prácticamente todos los programas de cómputo y para ello estos relacionan con el sistema operativos en muchas ocasiones, ejemplos de prácticas contemporáneas son el internet de las cosas (IoT), utilizando sensores para captar información o el uso de navegadores que escuchan respuestas asíncronas del servidor en todo momento, no se espera una respuesta instantánea.

El concepto como ya lo mencionamos no tuvo un único inventor, por el contrario fue evolucionando conforme el cómputo siguió necesitando de nuevas estrategias y métodos para completar tareas de forma más eficiente, desde el bajo nivel de arquitectura de software y manejo de interrupciones de hardware en modo batch hasta el nivel de programas que manejamos en la actualidad.

Patrón de diseño

Observer

El patrón de diseño observador, o también llamado *listener* o *event - subscribe* se clasifica dentro de los patrones de diseño de comportamiento y se apega al concepto explicado anteriormente. Este permite definir un mecanismo para identificar un objeto como el observado, de manera que, cuando este cambie de estado o simplemente ocurra algún evento se le notifique a todos los otros objetos de este.

Problema

Este patrón puede ser un problema porque el estar escuchando / observando puede ser un gasto de recursos si se tratan de muchos objetos, además estos objetos que están observando puede que nos siempre debían de estar pendientes a los eventos que le ocurran al objeto observado, así que caemos en un conflicto que si tiene solución

Solución

Una manera de evitar este tipo de estrés innecesario al sistema es crear una matriz para los objetos interesados en escuchar al objeto escuchado dentro de su interfaz, de tal forma que se tiene una lista de los suscriptores en concreto.

Kernel

El concepto de *kernel* de un sistema operativo no podría ser concebido sin tener en cuenta a los *listeners*, siendo que, el núcleo del sistema se basa en distintas formas de observar / escuchar eventos y reaccionar ante ellos. El *kernel* está constantemente escuchando para mantener el sistema funcionando de forma eficiente y con multiprocesos, algunos ejemplos son las interrupciones de hardware, gestión de solicitudes de usuario, sistema de archivos, solicitud de usuario que necesiten un servicio de kernel, etc.

Implementaciones (interrupciones)

A continuación, presentaremos el concepto de interrupciones y llamadas al sistema, sin embargo, esto es modo de símil dado que trabajan de forma muy similar, pero no son los mismo.

Existen dos tipos de interrupciones a nivel de sistema operativo que necesitan del concepto de *listener* para ser manejadas.

1. Hardware: Tiempo crítico
2. Software: No bloquea el sistema, dividiendo el procesamiento

Hardware - Nivel 1

El concepto de *listener* se hace presente en el mecanismo conocido como *hardware interrupts* (aunque un listener y una interrupción no son lo mismo) y es la base para todos los componentes físicos y lógicos que tienen una intervención directa en el funcionamiento del CPU y el sistema operativo.

Las interrupciones de hardware son aquellas señales eléctricas producidas por un dispositivo físico de la computadora. El proceso, a nivel físico, consta de la activación de ciertas entradas que tiene la CPU para ser interrumpida, activando una salida en concreto, para ejecutar un código especial que maneja la interrupción (similar a lo que explicamos en la historia de los *listeners* en la parte de la contextualización histórica).

Las interrupciones producidas por los dispositivos son decodificadas, también se coloca en el bus de datos de la máquina que dispositivo interrumpió y se activa la entrada INT de interrupción del CPU. El controlador de interrupciones tiene un registro de estado para manejar las interrupciones en el sistema; así permitiéndoles o inhibiendoles.

Interrupciones de hardware en Linux

Como ya sabemos, una de las partes más importantes de un sistema operativo es el manejo de interrupciones. Linux maneja un conjunto de estructuras de datos y tablas.

Algo importante es que el sistema de interrupciones es muy dependiente de la arquitectura, por ello aunque Linux tratará de que sea independiente de la máquina en la que esté el sistema, será inevitable su variabilidad en función del procesador. El sistema de interrupciones de Linux utiliza estructuras de datos como:

- irqaction: Almacena la dirección de la función que se está ejecutando (para manejar la itnerrupción).
- irq_chip: Contiene las funciones que utiliza un controlador de interrupciones.
- irq_desc: Vector con entradas para cada una de las interrupciones que se pueden atender.
- irq_action: Apuntador a la dirección de la función que hay que llevar a cabo.

Todas estas estructuras están en lenguaje C.

Software - Nivel 2

Las interrupciones a nivel de software son conocidas como llamadas al sistema, estas se dan cuando un proceso requiere solicitar algún servicio del sistema operativo. Es un mecanismo en el que salimos momentáneamente del modo usuario para que el SO realice alguna acción con recursos que solo él puede utilizar, también llamada como una acción privilegiada.

Esto escapa un poco del tema principal del trabajo, pero vale la pena comprender sus agrupaciones para saber que aunque tengamos que bajar hasta niveles de sistemas operativos, el concepto de escuchar solicitudes, conexiones y o eventos no se pasa por alto en ninguna interfaz.

Así pues, las llamadas al sistema se clasifican de la siguiente forma (nótese que en todos se espera / escucha a la realización de algún evento en específico, aunque sean de bajo nivel):

1. Control de procesos: Se realiza la llamada al sistema cuando se crea o espera la finalización de un proceso o cierto tiempo, asignar o liberar memoria, de forma que, el evento que se espera es un proceso o cambio de estado.

2. Manipulación de archivos: Se espera a que un archivo sea modificado, renombrado o eliminado.

3. Manipulación de dispositivos: Los sistemas permiten registrar eventos cuando un dispositivo es conectado o desconectado.

4. Mantenimiento de información: Son llamadas al sistema para obtener o modificar la hora del sistema.

5. Comunicaciones: Son las más comunes cuando se trata de conexiones de puertos o entre computadoras, se utilizan para establecer comunicación con determinado proceso, aceptar solicitudes o intercambiar información.

   Por ejemplo, socket.listen() en Python espera conexiones entrantes. No se trata de una llamada al sistema a nivel de sistema operativo, pero sirve para ejemplificar la agrupación.

6. Protección: Cuando para continuar algún proceso esté consulta o modifica accesos de objetos para poder acceder a ellos.

Las llamadas al sistema son independientes de cada sistema operativo y son expuestas al programador mediante interfaces de aplicación al programador (API). No se les cataloga como listeners formalmente como si en otros casos, pero tienen características asociadas por el hecho de escuchar eventos para completar alguna función.

Si bien un listener y una interrupción son conceptos diferentes, es cierto que, ambos comparten el propósito de manejar eventos, mientras que el *listener* maneja lo hace en aplicaciones de alto nivel, las interrupciones se dedican al bajo nivel, por ejemplo en sistemas operativos o en hardware cuando se tratan de eventos urgentes sin demoras que paran el proceso principal durante microsegundos.

Ahora bien, es normal que no surja la pregunta, ¿es todo lo que implica un *listener?* o ¿acaso hay maneras cotidianas en las que aplique el concepto?

La respuesta a esto es que no, de hecho existen divisiones / ramificaciones, a partir del concepto "puro" del patrón. Tengamos en cuenta que la definición de *listener* sólo se dedica a espera y responder a eventos, sin embargo, es difícil meter todos los huevos en la misma canasta y más cuando hablamos de tantos recursos como los que necesita el sistema, por lo que con el paso del tiempo surgió la necesidad de cambiar desde un paradigma de espera a uno más activo, algo más específica y rápido, como si de una vigilancia se tratara. Es allí dónde surge el concepto de *watcher.*

Watchers

Es imposible no hablar del uso de los *listeners* para supervisar el uso de recursos o cambios de estado, pero cuando nos referimos a un *watcher* podríamos confundir estos conceptos; la diferencia principal

entre ellos es que el patrón *listener* espera a que ocurra un evento en específico, mientras que el *watcher* más bien monitorea activamente un objeto en busca de cambios usando *polling* y su uso típico se da en actualizaciones de archivos de red o memoria cuando existe un solicitante y un proveedor, que podría ser un cliente y un servidor, respectivamente.

Ejemplos de usos de un *watcher* son el uso de las funciones de Linux:

- inotify: Provee de un mecanismo para monitorear eventos del sistema de archivos. Se pueden monitorear archivos sueltos o directorios, de manera que cuando se atienden estos archivos con la función está retorna eventos hacia el directorio o sus archivos.

  El nombre proviene de las palabras *node* y *notify*, por lo que hace sentido con su uso que es notificar cambios sobre archivos y o directorios (nodos).

- epoll: La utilizamos al trabajar con descriptores de archivos que son identificadores (con números enteros) de recursos abiertos que pueden ser archivos de disco, sockets, *pipes,* etc.
  Un uso típico es cuando servidores concurrentes (múltiples clientes al mismo tiempo) necesitan saber si el socket está listo para actualizar información, así que se recurre a esta para no bloquear el sistema de I/O.

  El nombre proviene de las palabras *event* y *poll*, es decir una encuesta para conocer el estado de identificadores que notifican sobre eventos que están pasando en el manejo del sistema operativo.

- fanotify: Monitorea los accesos a los archivos en todo el sistema. Útil para desarrollo de antivirus o ciberseguridad.

  Surge a partir de la combinación de palabras *file access notify*.

Esta noción es importante porque Linux utiliza el enfoque de que todo es un archivo.

Java

Java es probablemente el lenguaje que más ha adoptado el concepto de *listener* para su desarrollo, tanto así que cuando hacemos desarrollos que involucren GUI su uso es prácticamente un hecho. Para Java un *listener* es un objeto que espera y responde a eventos; utiliza el patrón de diseño *Observer* que ya explicamos anteriormente.

El uso de estos objetos se da implementando alguna interfaz *listener* a la clase que queramos que adquiera estas propiedades de escucha:

public class Beeper1 ... implements ActionListener {...}
public class Beeper2 ... implements MouseListener{...}
public class Beeper3 ... implements PropertyChangeListener{...}

Estas sirven para manejar eventos al escuchar presionar un botón, movimientos o clics del ratón y cambios en las propiedades de un objeto, en ese orden.

Por otra parte, también existe el concepto del *multilistener*, el cual sería una implementación de varios *listeners* que estén observando pasivamente al mismo evento, por lo que dependiendo el contexto este escuchara a múltiples objetos que pudieran ser puertos o clientes y puede notificar a varios oyentes.

La regla más importante acerca de los *listeners* es que estos deben ejecutarse muy rápidamente para que realmente nos sean de utilidad, porque todos los eventos escuchados se ejecutan en el mismo hilo, por lo que un método lento de un objeto *listener* puede tirar por la borda todo la parte gráfica de nuestro programa o alguna de las áreas de nuestro programa, el modo, la vista o el controlador (MVC), o todas si el problema es recurrente. Se recomienda que si se necesita realizar una gran operación como resultado de un evento, mejor se opte por iniciar un proceso en otro hilo para no comprometer la ejecución del hilo principal

Ventajas

- Eficiencia
- Implementación de una interfaz, en lugar de herencia, lo que la hace más flexible
- Facilita interacción con los usuarios

Aplicaciones

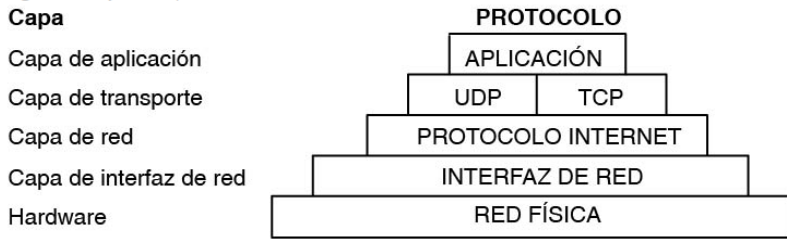
Si bien es innegable que la mayoría de los reflectores para los usos de los *listeners* en Java se los lleva la interfaz gráfica, existen otros tipos de aplicaciones de estos que se relacionan con los punto anteriores de la documentación, como:

- Redes: Escuchar conexiones entrantes en un socket (eventos de red)
- Sistema de archivos: Registrar directorios (eventos de archivos)
- Programación paralela: Detectar cambios de estado generados por hilos en segundo plano. Sirve para lanzar tareas asíncronas.

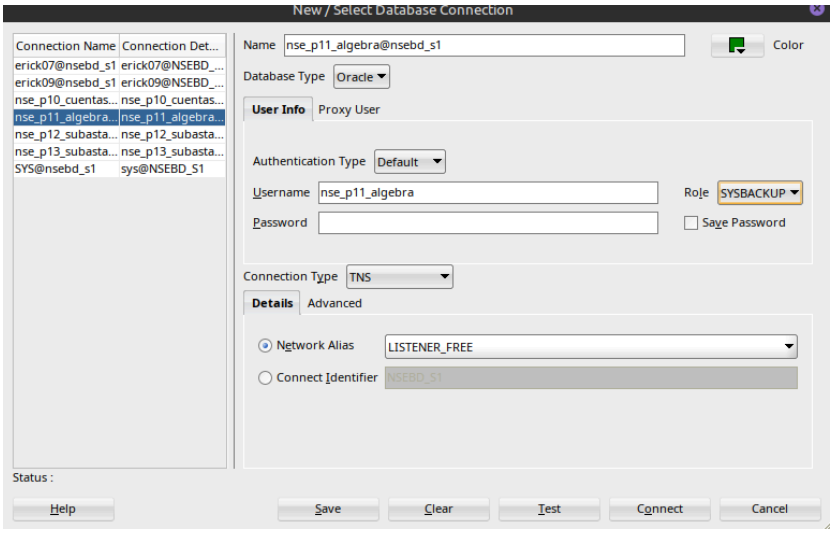
Oracle

Oracle cuenta con su propio listener, dicho listener es un proceso crucial para la comunicación de la base de datos. El listener de oracle ‘escucha’ las solicitudes de los usuarios dirigiéndose a la base de datos correctas. Las características del listener son las siguientes

- Gestiona el tráfico de solicitudes
- Gestiona las conexiones de los puertos
- Funciona con un protocolo TCP/IP



Esto hace que soporte conexiones con clientes, como lo es sqldeveloper



- Utiliza archivos para su configuración
    - listener.ora
    - tnsnames.ora
    - sqlnet.ora

Dentro de estos archivos se logra encontrar la siguiente información

```
# tnsnames.ora Network Configuration File: /opt/oracle/product/23ai/dbhomeFree/network/admin/tnsnames.ora
# Generated by Oracle configuration tools.

FREE =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = d2-bda-ens.fi.unam)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVER = DEDICATED)
      (SERVICE_NAME = free.fi.unam)
    )
  )

LISTENER_FREE =
  (ADDRESS = (PROTOCOL = TCP)(HOST = d2-bda-ens.fi.unam)(PORT = 1521))


ENSBDA_S2 =
  (DESCRIPTION =
    (ADDRESS_LIST =
      (ADDRESS = (PROTOCOL = TCP)(HOST = d2-bda-ens.fi.unam)(PORT = 1521))
    )
    (CONNECT_DATA =
      (SERVICE_NAME = ensbda_s2.fi.unam)
    )
  )
```

En el archivo se logran ver las conexiones a los servicios. El protocolo utilizado, en este caso TCP para una conexión local, el HOST Y el puerto utilizado, nótese que múltiples bases utilizan el mismo puerto.

```
  GNU nano 5.6.1
# listener.ora Network Configuration File: /opt/oracle/product/23ai/dbhomeFree/network/admin/listener.ora
# Generated by Oracle configuration tools.

LISTENER =
  (DESCRIPTION_LIST =
    (DESCRIPTION =
      (ADDRESS = (PROTOCOL = TCP)(HOST = d2-bda-ens.fi.unam)(PORT = 1521))
      (ADDRESS = (PROTOCOL = IPC)(KEY = EXTPROC1521))
    )
  )
```

[contenido archivo listener.ora]

El archivo sqlnet.ora puede no ser necesario. Sin embargo, un ejemplo sería:

```
# sqlnet.ora
#bloqueando conexion de ip especificas
TCP.VALIDNODE_CHECKING = YES
TCP.INVITED_NODES = (192.168.0.10, 192.168.0.11)
#CONEXIONES A BUSCAR
NAMES.DIRECTORY_PATH = (TNSNAMES, EZCONNECT)
#BLOQUEAR CONEXIONES SIN AUTENTICACION
SQLNET.AUTHENTICATION_SERVICES = (NONE)
#TIEMPOS MAXIMO DE CONEXION Y TIEMPO DE ESPERA
SQLNET.INBOUND_CONNECT_TIMEOUT = 60
SQLNET.RECV_TIMEOUT = 120
TRACE_LEVEL_CLIENT = OFF
```

Administración

The Listener Controller se encarga de administrar el listener. Este administrador provee varios comandos con distintos fines, tales como start, show, help, etc.

En práctica el comando start funciona para iniciar el listener, es aquí la forma para poder utilizar una base de datos usando el listener.

Se inicia el listener

```
erick@pc-bda-ens:~$ dockerBda2
c2-bda-ens
[root@d2-bda-ens /]# su oracle
[oracle@d2-bda-ens /]$ lsnrctl start

LSNRCTL for Linux: Version 23.0.0.0.0 - Production on 01-MAY-2025 18:55:07

Copyright (c) 1991, 2025, Oracle.  All rights reserved.

Starting /opt/oracle/product/23ai/dbhomeFree/bin/tnslsnr: please wait...

TNSLSNR for Linux: Version 23.0.0.0.0 - Production
System parameter file is /opt/oracle/product/23ai/dbhomeFree/network/admin/listener.ora
Log messages written to /opt/oracle/diag/tnslsnr/d2-bda-ens/listener/alert/log.xml
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=d2-bda-ens.fi.unam)(PORT=1521)))
Listening on: (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=EXTPROC1521)))

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=d2-bda-ens.fi.unam)(PORT=1521)))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for Linux: Version 23.0.0.0.0 - Production
Start Date                01-MAY-2025 18:55:09
Uptime                    0 days 0 hr. 0 min. 0 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File   /opt/oracle/product/23ai/dbhomeFree/network/admin/listener.ora
Listener Log File         /opt/oracle/diag/tnslsnr/d2-bda-ens/listener/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=d2-bda-ens.fi.unam)(PORT=1521)))
  (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=EXTPROC1521)))
The listener supports no services
The command completed successfully
[oracle@d2-bda-ens /]$
```

Visualizando los servicios registrados:

```
[oracle@d2-bda-ens admin]$ lsnrctl services

LSNRCTL for Linux: Version 23.0.0.0.0 - Production on 01-MAY-2025 20:44:15

Copyright (c) 1991, 2025, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=d2-bda-ens.fi.unam)(PORT=1521)))
Services Summary...
Service "2cf5fcb8b4ef1dc4e0631fc45e647325.fi.unam" has 1 instance(s).
  Instance "free", status READY, has 1 handler(s) for this service...
    Handler(s):
      "DEDICATED" established:0 refused:0 state:ready
         LOCAL SERVER
Service "3014e98b6eee0c4ee063020011ac0603.fi.unam" has 1 instance(s).
  Instance "free", status READY, has 1 handler(s) for this service...
    Handler(s):
      "DEDICATED" established:0 refused:0 state:ready
         LOCAL SERVER
Service "ensbda_s2.fi.unam" has 1 instance(s).
  Instance "free", status READY, has 1 handler(s) for this service...
    Handler(s):
      "DEDICATED" established:0 refused:0 state:ready
         LOCAL SERVER
Service "free.fi.unam" has 1 instance(s).
  Instance "free", status READY, has 1 handler(s) for this service...
    Handler(s):
      "DEDICATED" established:0 refused:0 state:ready
         LOCAL SERVER
Service "freeXDB.fi.unam" has 1 instance(s).
  Instance "free", status READY, has 1 handler(s) for this service...
    Handler(s):
      "D000" established:0 refused:0 current:0 max:1022 state:ready
         DISPATCHER <machine: d2-bda-ens.fi.unam, pid: 198>
         (ADDRESS=(PROTOCOL=tcp)(HOST=d2-bda-ens.fi.unam)(PORT=45513))
The command completed successfully
[oracle@d2-bda-ens admin]$
```

usando el comando de versión:

```
[oracle@d2-bda-ens admin]$ lsnrctl version

LSNRCTL for Linux: Version 23.0.0.0.0 - Production on 01-MAY-2025 20:44:11

Copyright (c) 1991, 2025, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=d2-bda-ens.fi.unam)(PORT=1521)))
TNSLSNR for Linux: Version 23.0.0.0.0 - Production
        TNS for Linux: Version 23.0.0.0.0 - Production
        Unix Domain Socket IPC NT Protocol Adaptor for Linux: Version 23.0.0.0.0 - Production
        Oracle Bequeath NT Protocol Adapter for Linux: Version 23.0.0.0.0 - Production
        TCP/IP NT Protocol Adapter for Linux: Version 23.0.0.0.0 - Production,,
The command completed successfully
```

Se levanta la instancia:

```
[oracle@d2-bda-ens /]$ lsnrctl status

LSNRCTL for Linux: Version 23.0.0.0.0 - Production on 01-MAY-2025 19:03:36

Copyright (c) 1991, 2025, Oracle.  All rights reserved.

Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)(HOST=d2-bda-ens.fi.unam)(PORT=1521)))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for Linux: Version 23.0.0.0.0 - Production
Start Date                01-MAY-2025 18:55:09
Uptime                    0 days 0 hr. 8 min. 27 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File   /opt/oracle/product/23ai/dbhomeFree/network/admin/listener.ora
Listener Log File         /opt/oracle/diag/tnslsnr/d2-bda-ens/listener/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=d2-bda-ens.fi.unam)(PORT=1521)))
  (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=EXTPROC1521)))
The listener supports no services
The command completed successfully
[oracle@d2-bda-ens /]$ su erick
Password:
[erick@d2-bda-ens /]$ sqlplus sys as sysdba

SQL*Plus: Release 23.0.0.0.0 - Production on Thu May 1 19:04:37 2025
Version 23.7.0.25.01

Copyright (c) 1982, 2025, Oracle.  All rights reserved.

Enter password:
Connected to an idle instance.

idle> startup
ORACLE instance started.

Total System Global Area  802346584 bytes
Fixed Size                  5428824 bytes
Variable Size             390070272 bytes
Database Buffers          402653184 bytes
Redo Buffers                4194304 bytes
Database mounted.
Database opened.
```

Una vez iniciado el listener y teniendo la instancia levantada se puede consultar el estado

```
[erick@d2-bda-ens /]$ lsnrctl status

LSNRCTL for Linux: Version 23.0.0.0.0 - Production on 01-MAY-2025 19:06:21

Copyright (c) 1991, 2025, Oracle.  All rights reserved.

Connecting to (ADDRESS=(PROTOCOL=tcp)(HOST=)(PORT=1521))
STATUS of the LISTENER
------------------------
Alias                     LISTENER
Version                   TNSLSNR for Linux: Version 23.0.0.0.0 - Production
Start Date                01-MAY-2025 18:55:09
Uptime                    0 days 0 hr. 11 min. 12 sec
Trace Level               off
Security                  ON: Local OS Authentication
SNMP                      OFF
Listener Parameter File   /opt/oracle/product/23ai/dbhomeFree/network/admin/listener.ora
Listener Log File         /opt/oracle/diag/tnslsnr/d2-bda-ens/listener/alert/log.xml
Listening Endpoints Summary...
  (DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=d2-bda-ens.fi.unam)(PORT=1521)))
  (DESCRIPTION=(ADDRESS=(PROTOCOL=ipc)(KEY=EXTPROC1521)))
Services Summary...
Service "2cf5fcb8b4ef1dc4e0631fc45e647325.fi.unam" has 1 instance(s).
  Instance "free", status READY, has 1 handler(s) for this service...
Service "3014e98b6eee0c4ee063020011ac0603.fi.unam" has 1 instance(s).
  Instance "free", status READY, has 1 handler(s) for this service...
Service "ensbda_s2.fi.unam" has 1 instance(s).
  Instance "free", status READY, has 1 handler(s) for this service...
Service "free.fi.unam" has 1 instance(s).
  Instance "free", status READY, has 1 handler(s) for this service...
Service "freeXDB.fi.unam" has 1 instance(s).
  Instance "free", status READY, has 1 handler(s) for this service...
The command completed successfully
[erick@d2-bda-ens /]$
```

Bibliografía

De Roer, D. D., & De Roer, D. D. (2022, Junio). Eventos y Listeners en Java. Disco Duro de Roer -. https://www.discoduroderoer.es/eventos-y-listeners-en-java/

El listener de Oracle | Dataprix. (2008). https://www.dataprix.com/es/forum/oracle-database/el-listener-de-oracle

Introduction to event listeners (The JavaTM Tutorials > Creating a GUI with Swing > Writing Event listeners). https://docs.oracle.com/javase/tutorial/uiswing/events/intro.html

Observer. (n.d.). https://refactoring.guru/es/design-patterns/observer

Grey, R. (2024, Octubre 13). ¿Qué es ARPANET? - NinjaOne. NinjaOne. https://www.ninjaone.com/es/it-hub/it-service-management/que-es-arpanet/

Mallón, X. (2025). Sockets: Qué son, cómo funcionan y tipos - Guía 2025. KeepCoding Bootcamps. https://keepcoding.io/blog/que-es-un-socket/

Universidad de Las Palmas de Gran Canaria, LECCIÓN 3: INTERRUPCIONES HARDWARE. https://sopa.dis.ulpgc.es/ii-dso/leclinux/interrupciones/int_hard/LEC3_INT_HARD.pdf

Wolf, G., Ruíz, E., Bergero, F., & Meza, E. (2015). *FUNDAMENTOS DE SISTEMAS OPERATIVOS*, Universidad Nacional Autónoma de México.

inotify(7) - Linux manual page. https://man7.org/linux/man-pages/man7/inotify.7.html

epoll(7) - Linux manual page. https://man7.org/linux/man-pages/man7/epoll.7.html

Introduction to event listeners (The JavaTM Tutorials > Creating a GUI with Swing > Writing Event listeners). (n.d.). https://docs.oracle.com/javase/tutorial/uiswing/events/intro.html

AIX 7.2. (s. f.). https://www.ibm.com/docs/es/aix/7.2?topic=protocol-tcpip-protocols
August2024. (2024, 24 octubre). Service data provider. Oracle Help Center. https://docs.oracle.com/en/cloud/paas/integration-cloud/visual-developer/service-data-provider.html

Database net Services reference. (s. f.). https://docs.oracle.com/cd/E11882_01/network.112/e10835/lsnrctl.htm#CHDBDHHF

Despliegue de una topología de DR híbrida para una Oracle Exadata local. (s. f.). Oracle Help Center. https://docs.oracle.com/es/solutions/hybrid-dr-for-exadata/configure-listeners-and-sqlnet-ora.html#GUID-8E4A6551-2A7D-4A50-9A42-C201420012A3

Starting the Oracle Listener (Sun Java System Mobile Enterprise Platform 1.0 Installation Guide). (s. f.). https://docs.oracle.com/cd/E19957-01/820-3750/ggrgc/index.html

Tivoli NetCool Performance Manager 1.4.4. (s. f.). https://www.ibm.com/docs/en/tnpm/1.4.4?topic=bit-configure-oracle-listener

Williams, D., & Jashnani, P. (2024, 5 septiembre). Configuring and Administering Oracle Net Listener. Oracle Help Center. https://docs.oracle.com/en/database/oracle/oracle-database/19/netag/configuring-and-administering-oracle-net-listener.html#GUID-09139A19-5265-4216-9054-3237B225E09D