Servicios Multimedia
Curso 2024/2025
Práctica 1 - VoIP: Asterisk

Índice

Lista de ejercicios

Códigos

1. Objetivo

El objetivo de esta práctica es probar las funcionalidades básicas de Asterisk y comprender cómo se programa un *dialplan*. La práctica se divide en tres partes:

1. Definición y configuración de unos clientes *SIP* de ejemplo.
2. Configuración del buzón de voz de estos clientes.
3. Programación de varias extensiones dentro del *dialplan*, de forma que los clientes obtengan diferentes respuestas según el número que marquen.

Cada una de estas partes se compone de una descripción detallada del objetivo a cumplir y una serie de ejercicios a realizar. En el documento *Notas de programación en Asterisk* se encuentra toda la documentación necesaria para poder hacerlo.

NOTA: En el campus virtual hay un archivo que contiene los códigos de ejemplo que aparecen en esta práctica en modo texto. Se recomienda usar estos ficheros para copiar y pegar, en lugar de hacerlo desde el PDF, ya que a veces se puede copiar algún carácter extraño no visible y después al pegarlo dentro de Asterisk dará un error.

2. Práctica

2.1. Configuración de los clientes SIP

Queremos tener dos usuarios en nuestro sistema: `sip1` y `sip2`. Para ello usaremos el fichero de configuración `pjsip_custom.conf` mostrado a continuación en el código 1.

Código 1: Configuración de clientes SIP en `pjsip_custom.conf`

```
[transport_udp]
type=transport
protocol=udp
bind=0.0.0.0

[sip1]
type=endpoint
aors=sip1
auth=authsip1
context=practica1
mailboxes=001@sm
disallow=all
allow=alaw,g723
transport=transport_udp

[authsip1]
type=auth
auth_type=userpass
password=passwdsip1
username=sip1

[sip1]
type=aor
max_contacts=1
remove_existing=yes

[sip2]
type=endpoint
aors=sip2
auth=authsip2
context=practica1
mailboxes=002@sm
disallow=all
allow=alaw,g723
transport=transport_udp

[authsip2]
type=auth
auth_type=userpass
password=passwdsip2
username=sip2

[sip2]
type=aor
max_contacts=1
remove_existing=yes
```

En el documento *Notas de programación en Asterisk*, el capítulo 1 describe en detalle el significado de las diferentes partes de este fichero. Asegúrate de que entiendes el significado de cada una de estas partes antes de continuar con el ejercicio.

Ejercicio 1: Configurar los clientes `sip1` y `sip2`

Configura los clientes SIP `sip1` y `sip2` en Asterisk usando el fichero de configuración `pjsip_custom.conf` del código 1. Una vez configurados, conecta los clientes y ejecuta en la consola de Asterisk el comando `pjsip show endpoints`. **Copia la salida** y en base a esta responde a las siguientes preguntas (**justifica** las respuestas):

1. ¿Cual es el estado de los *endpoints* `sip1` y `sip2`?
2. ¿En que direcciones IP y puertos se puede contactar con estos *endpoints*?
3. ¿Cuantos contactos podrían registrarse como máximo para cada uno de los *endpoints*? ¿Cómo lo sabes?

2.2. Configuración de los buzones de voz

En Asterisk la configuración de los buzones de voz se realiza en el fichero `voicemail.conf`. En el código 2 se muestra un ejemplo de un fichero de configuración de este tipo. En el documento *Notas de programación en Asterisk*, el capítulo 2 describe en detalle el significado de las diferentes secciones de este código. Asegúrate de que entiendes el significado de cada una de estas partes antes de continuar con el ejercicio.

Código 2: Ejemplo de configuración de `voicemail.conf`

```
[general]
format=wav49|wav
serveremail=voicemail@udc.test.es
attach=yes
maxsilence=10
maxlogins=3

[zonemessages]
central=America/Chicago|'vm-received' Q 'digits/at' IMp
european=Europe/Copenhagen|'vm-received' a d b 'digits/at' HM

[mi_contexto]
111 => 1234,Juan,juan@udc.test.es,,tz=central
112 => 0000,Pepe,pepe@udc.test.es,,tz=eastern|maxmsg=5
```

Ejercicio 2: Configurar los buzones de voz para `sip1` y `sip2`

Usando como modelo el código 2, configura los buzones de voz para los clientes SIP `sip1` y `sip2` editando el fichero `voicemail.conf` de Asterisk siguiendo las siguientes instrucciones:

1. Fíjate que en la configuración de los clientes SIP (código 1) se indicaron sus correos en los parámetros `mailboxes` como:
   - `001@sm` para `sip1`
   - `002@sm` para `sip2`

   Por lo tanto, configura los buzones de voz con número `001` y `002` en un contexto con el nombre `sm`.
2. Establece las siguientes opciones para los dos buzones como se indica:
   - `nombre`: pon tu nombre.
   - `email address`: pon el correspondiente email de la UDC.
   - `short email address`: deja esta opción vacía.
3. Una vez configurados los buzones de voz asegúrate de recargar la configuración en Asterisk ejecutando en la consola de Asterisk el comando `voicemail reload`. A continuación ejecuta en la consola de Asterisk el comando `voicemail show users` para mostrar los buzones de voz configurados y **copia la salida obtenida**.

2.3. Programación de un *dialplan*

Una vez que los dos *endpoints* `sip1` y `sip2` y los buzones de voz están correctamente configurados en nuestro sistema, vamos a implementar un *dialplan* de forma que la *PBX* ofrezca las siguientes funcionalidades dependiendo de cual sea la extensión marcada por el usuario:

01 Permite llamar al *endpoint* `sip1`.

02 Permite llamar al *endpoint* `sip2`.

00*X* Con ***X*** = 1 o 2. Permite llamar a `sip1` o `sip2`, y si no se puede entonces activa el buzón de voz para poder dejar un mensaje. La secuencia de acciones sería:

1. Reproducir el mensaje *you-have-dialed*.
2. Decir la extensión marcada.
3. Llamar a `sip1` si ***X*** = 1; llamar a `sip2` si ***X*** = 2.
4. Si no se puede realizar la llamada hacer saltar el buzón de voz. El mensaje de bienvenida debe corresponderse con la causa que provocó que no se pudiese realizar la llamada: *busy* si la llamada finalizó con un estatus de ocupado o *unavailable* en otro caso.

120 El usuario que llama puede consultar su propio buzón de voz.

99*X* Con ***X*** = 1 o 2. Permite llamar a `sip1` o `sip2`, pero solo en unas franjas horarias determinadas. La extensión debe **aceptar** las llamadas solo en el siguiente horario:

- Lunes a Jueves de 15:30 a 20:00, durante los todos los meses del año excepto en Agosto.

Si se llama fuera del horario permitido se reproducirá un mensaje (e.g., *unavailable* & *please-try-again-later*). Si se llama dentro del horario permitido, se llamará a `sip1` si ***X*** = 1, o a `sip2` si ***X*** = 2.

#0 Se reproduce un menú con el siguiente comportamiento:

1. Se le pide al usuario que pulse 0, 1, 2, o 3 (e.g., `Playback(press)` & `SayNumber(1)` & `Playback(or)` ...).
2. Se espera a que marque la extensión (máximo 10 segundos).
3. Se realiza una acción dependiendo de la extensión marcada:
   - **0** Se indica el número de llamadas que no se han podido enviar por estar no disponible, y al acabar se vuelve al punto 1.
   - **1** Se indica el número de llamadas en las que estaba ocupado, y al acabar se vuelve al punto 1.
   - **2** Se indica el número de llamadas que ha contestado, y al acabar se vuelve al punto 1.
   - **3** Se reproduce el mensaje *goodbye* y se cuelga la llamada.
4. Si la extensión marcada no es válida reproducir el mensaje *pbx-invalid* y se vuelve al punto 1.
5. Si se agota el tiempo de espera sin marcar ninguna extensión se reproduce el mensaje *goodbye* y se cuelga la llamada.

A continuación se plantean varios ejercicios que guiarán y facilitarán la implementación del *dialplan*.

Una de las características de Asterisk que usaremos en la implementación del *dialplan* son las subrutinas usando la aplicación `GoSub()`. En la sección 3.4.4 del documento *Notas de programación en Asterisk* se explica en qué consisten las subrutinas y como se usan. Asegurate de entender bien este concepto antes de continuar.

2.3.1. Implementación de la subrutina `sub_recordcallinfo`

En la extensión `#0` necesitaremos un mecanismo para registrar las llamadas que se producen en el sistema y su estado (respondido, ocupado, etc.).

La manera de implementar esta funcionalidad en un sistema real sería usar alguno de los módulos de *call detail recording* (CDR) que proporciona Asterisk. Estos módulos permiten registrar en base de datos la información de cada llamada que entra en el sistema. Posteriormente para recuperar esa información y reproducirla podríamos consultar la base de datos, lo cual es posible en Asterisk mediante el módulo `res_odbc`.

Por simplicidad, en esta práctica no consideraremos el uso de módulos de CDR e implementaremos el registro de información de llamadas en el *dialplan*. Para esto crearemos una subrutina `sub_recordcallinfo` que se encargará de guardar la información de la llamada.

En Asterisk se usa la aplicación `Dial()` para hacer las llamadas, y al terminar se puede obtener su estado mediante la variable `DIALSTATUS`. En la sección 3.4.1 del documento *Notas de programación en Asterisk* se explica en detalle el funcionamiento de `Dial()`. Asegurate de entender bien su funcionamiento antes de continuar.

Para guardar el registro de llamadas, existen dos maneras sencillas de hacerlo directamente en el dialplan de Asterisk:

- Usando variables globales:

  En Asterisk, por defecto, las variables globales se conservan aunque recarguemos el *dialplan* (este comportamiento puede cambiarse con la opción `clearglobalvars=yes` en la sección `[general]` del *dialplan*), pero se pierden al reiniciar Asterisk.

- Usando la base de datos de Asterisk (AstDB):

  Asterisk dispone de una base de datos no relacional llamada AstDB que permite almacenar datos de la forma *familia/clave=valor* de manera persistente. Por lo tanto, en AstDB los datos se agrupan en familias con valores identificados por claves, donde una clave solo puede usarse una vez dentro de una misma familia.

  Para almacenar datos en AstDB se usa la aplicación `Set()` con la función `DB()`. Por ejemplo, para establecer a 1 la clave `count` de la familia `test`:

  ```
  exten => 216,1,Set(DB(test/count)=1)
  ```

  Para recuperar datos de AstDB se usa igualmente la función `DB()`, así por ejemplo, para recuperar el valor del ejemplo anterior:

  ```
  exten => 217,1,Set(MyCount=${DB(test/count)})
  ```

Nosotros consideraremos la implementación de esta subrutina usando AstDB, almacenando el número de llamadas en la familia `calls_`*`usuario`*, siendo *`usuario`* el nombre de usuario indicado en el parámetro 1 de la función, y con las siguientes claves:

- `a`: llamadas terminadas con `DIALSTATUS` = `ANSWER`.

- b: llamadas terminadas con DIALSTATUS = BUSY.
- u: llamadas terminadas con DIALSTATUS = CHANUNAVAIL.
- n: llamadas terminadas con DIALSTATUS = NOANSWER.
- o: llamadas terminadas con otro valor de DIALSTATUS.

Ejercicio 3: Implementar la subrutina sub_recordcallinfo

Implementa la subrutina sub_recordcallinfo a partir del código 3 como sigue:

1. Completa el fragmento resaltado *(a)* con el resto de posibles valores de DIALSTATUS.
2. Completa el fragmento resaltado *(b)* con el código para crear una variable local llamada familykey, que contendrá la cadena *familia/clave* para AstDB, con el valor calls_*usuario*/*clave*, siendo *usuario* el indicado en el parámetro ARG1 de la subrutina y clave la establecida en la variable local key.
3. Completa el fragmento resaltado *(c)* con el código para asignar en AstDB el valor de count + 1 a la familia/clave contenida en la variable local familykey.

Código 3: Código de la subrutina sub_recordcallinfo (a completar)

```
[sub_recordcallinfo]
; Record call information using AstDB
; Parameters:
; ARG1: User
; ARG2: Dialstatus
exten => start,1,Verbose(3,Recording call info: ${ARG1} (${ARG2}))
same => n,ExecIf($[${ARG2}=ANSWER]?Set(LOCAL(key)=a))
same => n,                              (a)
same => n,                              (a)
same => n,                              (a)
same => n,ExecIf(${ISNULL(${LOCAL(key)})}?Set(LOCAL(key)=o))
same => n,                              (b)
same => n,Set(LOCAL(count)=${DB(${LOCAL(familykey)})})
same => n,ExecIf($[${ISNULL(${LOCAL(count)})} = 1]?Set(LOCAL(count)=0))
same => n,                              (c)
same => n,Return()
```

2.3.2. Implementación de la subrutina sub_sipdial

Una vez implementada la subrutina sub_recordcallinfo, implementaremos la subrutina sub_sipdial, que realizará una llamada usando la aplicación Dial() y luego ejecutará sub_recordcallinfo para guardar los datos de la llamada.

Ejercicio 4: Implementar la subrutina sub_sipdial

Implementa la subrutina sub_sipdial a partir del código 4 como sigue:

1. En el fragmento resaltado *(a)* usa la aplicación Dial() para llamar al *endpoint*

indicado en el parámetro ARG1, con el *timeout* indicado en el parámetro ARG2, y las opciones indicadas en el parámetro ARG3.

2. Si la llamada realizada por la aplicación Dial() definida en el fragmento *(a)* no se responde (por timeout o por alguna otra razón) se continuará con el *dialplan* en la extensión actual. En el fragmento resaltado *(b)*, llama a la subrutina sub_recordcallinfo, pasándole como parámetros el *endpoint* indicado en el parámetro ARG1 y el estado de la llamada indicado en DIALSTATUS.

3. Si la llamada se contesta, al finalizarla se saltará a la extensión especial *h*. Completa el fragmento *(c)* para que, cuando esa extensión se ejecute, se salte a la prioridad etiquetada como *record*, y que así se guarde en la base de datos también las veces que se contestó la llamada.

Código 4: Código de la subrutina sub_sipdial (a completar)

```
[sub_sipdial]
; Dial a user and save the call information
; Parameters:
; ARG1: PJSIP endpoint (e.g., sip1 or sip2)
; ARG2: Timeout
; ARG3: Options
exten => start,1,Verbose(1,Dialing SIP user ${ARG1})
same => n,                          (a)
same => n(record),                          (b)
same => n,Return()

exten => h,1,                          (c)
```

2.3.3. Implementación de las extensiones 01 y 02

Una vez que hemos implementado las subrutinas anteriores ya estamos listos para implementar las extensiones 01 y 02 para llamar respectivamente a los usuarios sip1 y sip2.

Ejercicio 5: Implementar las extensiones 01 y 02

Implementa las extensiones 01 y 02 a partir del código 5. Para ello completa los fragmentos resaltados *(a)* y *(b)* con llamadas a la subrutina sub_sipdial, pasándole como parámetro el nombre del *endpoint PJSIP* correspondiente (sip1 o sip2).

Una vez implementadas las extensiones y aplicados los cambios en Asterisk realiza algunas llamadas de prueba y comprueba que la información de las llamadas se registra en AstDB como se indicó en el apartado 2.3.1. Para ello ejecuta en la consola de Asterisk los comandos database show calls_sip1 y database show calls_sip2 y copia su salida. ¿Se registra correctamente la información de las llamadas?

Código 5: Código de las extensiones 01 y 02 (a completar)

```
exten => 01,1,Verbose(1,Dial user sip1)
same => n,                          (a)
same => n,HangUp()

exten => 02,1,Verbose(1,Dial user sip2)
```

```
same => n,                                (b)
same => n,HangUp()
```

2.3.4. Implementación de la extensión 00*X*

Queremos ahora implementar la extensión 00*X*, siendo *X* = 1 o 2. En esta extensión, si no se pudo realizar la llamada se debe hacer saltar el buzón de voz con el mensaje apropiado. Para ello en primer lugar implementaremos una subrutina `sub_voicemail` que se encargue de esto. Esta subrutina recibirá 3 parámetros: el valor de `DIALSTATUS`, el buzón de voz, y opciones adicionales.

Por simplicidad, en la implementación del esta extension definiremos en el *dialplan* las variables globales mostradas en el código 7 para definir las direcciones de los buzones de voz de los usuarios.

Para hacer uso del buzón de voz, usaremos la aplicación `VoiceMail()`. En la sección 3.4.6 del documento *Notas de programación en Asterisk* se explica en detalle el funcionamiento de `Voicemail()`. Asegurate de entender bien su funcionamiento antes de continuar.

Por defecto los sonidos que se reproducen en Asterisk lo hacen en inglés. Para hacer que se use el español, puedes consultar el apéndice C.

Ejercicio 6: Implementar la subrutina `sub_voicemail`

Implementa la subrutina `sub_voicemail` a partir del código 6 completando los fragmentos resaltados *(a)* y *(b)* como sigue:

1. En el fragmento resaltado *(a)* usa la función `IF` para establecer la variable local `message` a `b` si el parámetro `ARG2` (el valor de `DIALSTATUS`) es igual a la cadena `BUSY`, o establecerla a `u` en caso contrario.

2. En el fragmento resaltado *(b)* usa la aplicación `VoiceMail()` para lanzar el buzón de voz, pasandole como primer parámetro el valor de `ARG1`, y como segundo parámetro la concatenación de `message` y `ARG3`.

Código 6: Código de la subrutina `sub_voicemail` (a completar)

```
[sub_voicemail]
; Call voicemail after a Dial
; Parameters:
; ARG1: mailbox@context
; ARG2: value of DIALSTATUS
; ARG3: additional options to voicemail
exten => start,1,Verbose(3,Executing sub_voicemail for ${ARG1})
same => n,GotoIf($[ ${ISNULL(${ARG2})} | ${ARG2} = ANSWER ]?exit)
same => n,Set(LOCAL(message)=                    (a)                    )
same => n,                                (b)
same => n(exit),Return()
```

Ejercicio 7: Implementar la extensión 00*X*

Implementa la extensión 00*X*. Para ello en primer lugar copia al *dialplan* las variables globales indicadas en el código 7, las cuales serán necesarias para el código de la

extensión. A continuación implementa la extensión a partir del código 8 completando los fragmentos resaltados *(a)*, *(b)*, y *(c)* como sigue:

1. El el fragmento resaltado *(a)* usa un patrón de extensión que coincida con las extensiones 001 o 002.

2. En el fragmento resaltado *(b)* usa la variable de Asterisk que proporciona el número de la extensión actual para que este se reproduzca.

3. En el fragmento resaltado *(c)* inserta la expresión adecuada para que a la variable `user` se le asigne la cadena `sip1` si $X = 1$, o `sip2` si $X = 2$. Para esto simplemente concatena la cadena `sip` a la cadena obtenida de manipular la variable de Asterisk de la extensión actual para extraer su último dígito.

Una vez configurada la extensión prueba a dejar algún mensaje en el buzón de voz, luego ejecuta en la consola de Asterisk el comando `voicemail show users` y copia su salida. En la salida de este comando se indica para cada usuario en la columna «NewMsg» el número de mensajes nuevos en el buzón de voz. ¿Se están registrando correctamente los mensajes en el buzón de voz?

Código 7: Variables globales para la extensión 00*X*

```
MAILBOX_sip1=001@sm
MAILBOX_sip2=002@sm
```

Código 8: Código de la extension 00*X* (a completar)

```
exten => (a),1,Verbose(1,Say extension and dial user)
same => n,Playback(you-have-dialed)
same => n,SayDigits( (b) )
same => n,Set(user= (c) )
same => n,GoSub(sub_sipdial,start,1(${user}))
same => n,GoSub(sub_voicemail,start,1(${MAILBOX_${user}},${DIALSTATUS}))
same => n,HangUp()
```

2.3.5. Implementación de la extensión 120

En la extensión 120 se nos pide que el usuario que llama pueda consultar su buzón de voz. Para implementarla, necesitaremos dos cosas: conocer la dirección del buzón de voz de la persona y después usar la aplicación `VoiceMailMain()` para que pueda consultarlo. En la sección 3.4.6 del documento *Notas de programación en Asterisk* se explica en detalle el funcionamiento de `VoiceMailMain()`. Asegurate de entender bien su funcionamiento antes de continuar.

Ejercicio 8: Implementar la extensión 120

Implementa la extensión 120 a partir del código 9 completando los fragmentos resaltados *(a)*, *(b)*, y *(c)* como sigue:

1. En primer lugar, necesitamos obtener el nombre del *endpoint* que ha realizado la llamada, que en nuestro caso será `sip1` o `sip2`. Para esto usaremos la función `CHANNEL()`, con la que podemos obtener información del canal. En la sección 3.5

del documento *Notas de programación en Asterisk* tienes mas información acerca de esta función.

2. Una vez almacenado el nombre del *endpoint* en la variable `user`, en la siguiente prioridad de la extensión se guarda la dirección de buzón de voz en la variable `mailbox` a partir de la variable global correspondiente definida en el código 7. Sin embargo, si definimos un nuevo usuario SIP y no definimos una nueva variable global con su dirección de buzón de voz, el valor de la variable `mailbox` será la cadena vacía. Completa el fragmento resaltado *(b)* usando la aplicación `GotoIf()` de forma que si `mailbox` es una cadena vacía se salte a la prioridad con la etiqueta `end`. Para esto revisa el apéndice A.

3. Finalmente, Completa el fragmento resaltado *(c)* para consultar los mensajes del buzón de voz indicado en en la variable `mailbox`.

Una vez configurada la extensión el usuario podrá llamar y escuchar los mensajes que se le han dejado en su buzón. Prueba a llamar a la extensión, ¿Puedes escuchar los mensajes enviados al usuario?

Código 9: Código de la extensión 120 (a completar)

```
exten => 120,1,Verbose(1,Check voicemail messages)
same => n,Set(user=                (a)                )
same => n,Set(mailbox=${MAILBOX_${user}})
same => n,                    (b)
same => n,                    (c)
same => n,HangUp()
same => n(end),Verbose(1,Mailbox for user ${user} not defined)
same => n,Playback(feature-not-avail-line)
same => n,HangUp()
```

2.3.6. Implementación de la extensión 99*X*

En la extensión 99*X*, siendo $X = 1$ o 2, tenemos que permitir la llamada al usuario `sip1` o `sip2` en unos determinados horarios, rechazando la llamada en el resto. Para implementar esta funcionalidad en Asterisk se usará la aplicación `GoToIfTime()`. En la sección 3.4.3 del documento *Notas de programación en Asterisk* se explica en detalle el funcionamiento de `GoToIfTime()`.

Un aspecto importante a considerar al usar `GoToIfTime()` o cualquier otra aplicación que se base en fechas (p.e., `IfTime()`) es la zona horaria. Por defecto, en la distribución FreePBX no hay ninguna zona horaria configurada en el sistema, por lo que estas aplicaciones usarán directamente la fecha en tiempo universal coordinado (UTC). Por lo tanto, deberemos establecer la zona horaria en el sistema o indicarla cuando llamemos a estas aplicaciones. Esta segunda opción será la que consideremos, para ello definimos la variable global `TIMEZONE` establecida a la zona horaria «Europe/Madrid» como se muestra en el código 10.

Para probar el funcionamiento correcto de esta extensión se puede cambiar manualmente la fecha del sistema. Para esto, en la distribución FreePBX primero deberemos de desactivar el servicio `chronyd`, que es el encargado de la sincronización de la fecha mediante el network time protocol (NTP). Para desactivar este servicio ejecutaremos el siguiente comando en la terminal de FreePBX:

```
systemctl stop chronyd
```

Una vez hecho esto podemos cambiar la fecha del sistema con ejecutando el comando `date` en la terminal del sistema de la siguiente forma:

```
date MMDDhhmm[[CC]YY]
```

donde

- `MM` es el número de mes.
- `DD` es el día del mes.
- hh es la hora del día.
- `mm` son los minutos.
- `[[CC]YY]` es el año, el cual es opcional y puede indicarse mediante un número de 2 o 4 cifras.

Al finalizar las pruebas iniciaremos otra vez el servicio `chronyd` mediante el siguiente comando:

```
systemctl start chronyd
```

Ejercicio 9: Implementar la extensión 99*X*

Implementa la extensión 99*X*. Para ello primero copia al dialplan la variable global `TIMEZONE` indicada en el código 10. A continuación implementa la extensión a partir del código 11 completando los fragmentos resaltados *(a)*, *(b)*, y *(c)* como sigue:

1. El el fragmento resaltado *(a)* usa un patrón de extensión que coincida con las extensiones 991 o 992.
2. El el fragmento resaltado *(b)* usa la aplicación `GoToIfTime()` para comprobar si la llamada se produce en el horario permitido definido en la extensión 99*X*. En caso de que sea así, haz que se acepte saltando a la prioridad con la etiqueta `accept`.
3. Finalmente, en el fragmento resaltado *(c)* llama a la subrutina `sub_sipdial` pasandole como usuario `sip1` si $X = 1$, o `sip2` si $X = 2$.

Código 10: Variable global `TIMEZONE` para la extensión 99*X*

```
TIMEZONE=Europe/Madrid
```

Código 11: Código de la extensión 99*X* (a completar)

```
exten => (a),1,Verbose(1,Accept calls only in the given times)
same => n,Verbose(1,System time is: ${STRFTIME()})
same => n,Verbose(1,Local time is: ${STRFTIME(,${TIMEZONE})})
(b)
same => n,Playback(unavailable)
same => n,Playback(please-try-again-later)
same => n,Goto(exit)
same => n(accept),(c)
same => n(exit), HangUp()
```

2.3.7. Implementación de la extensión #0

La extensión **#0** permite escuchar el número de llamadas perdidas y respondidas por el usuario. Para implementarla es necesario crear antes una subrutina que permita obtener ese número desde la base de datos AstDB que se usó en la subrutina `sub_recordcallinfo`. Esta nueva subrutina se puede ver en código 12. Recibe como parámetros el usuario del que se quieren consultar las llamadas y el tipo de llamada. Devuelve un valor que se corresponde con el número de llamadas de ese tipo registradas en la base de datos.

Código 12: Código de la subrutina `sub_getcallscount` (a completar)

```
[sub_getcallscount]
; Gets the count of calls using AstDB
; ARG1: Endpoint (sip1 or sip2)
; ARG2: Type: a(answer), b(BUSY), u(CHANUNAVAIL), n(NOANSWER), o(OTHER)
exten => start,1,Verbose(3,Getting call numer for: User ${ARG1}, type: ${ARG2})
same => n, Set(LOCAL(familykey)=    (a)
same => n, Set(LOCAL(count)=    (b)
same => n, ExecIf($[${ISNULL(${LOCAL(count)})} = 1]?Set(LOCAL(count)=0))
same =>     (c)
```

Ejercicio 10: Implementar la subrutina `sub_getcallscount`

Implementa la subrutina `sub_getcallscount` a partir del código 12 como sigue:

1. En el fragmento resaltado *(a)* guarda en `LOCAL(familykey)` la ruta de la base de datos que almacena el valor pedido.
2. En el fragmento resaltado *(b)*, usa la función `DB` para extraer el dato almacenado en la ruta `LOCAL(familykey)`.
3. Por último, en el fragmento resaltado *(c)* haz que la subrutina devuelva el dato. En la sección 3.4.4 del documento *Notas de programación en Asterisk* se explica el mecanismo por el que una subrutina puede devolver un resultado y como se obtiene este desde la extensión que hizo la llamada.

Ejercicio 11: Implementar la extensión #0

Implementa todos los casos descritos en la definición de la extensión #0. Revisa que la extensión ejecute las acciones correspondientes a cada situación (extensión inválida, se agota el tiempo de espera, etc.). Para ayudarte con el ejercicio, revisa el apéndice B en donde se explica el manejo de los menús. Puedes también guiarte por el menú de ejemplo que se ve en el código código 13, pero recuerda adaptarlo a lo que se pide para #0.

3. Entrega

La práctica se puede realizar individualmente o por parejas. **En caso de que se haga por parejas solo debe hacer la entrega uno de los dos autores en el campus virtual, pero en el fichero de respuestas se debe poner al principio el nombre de los dos**. La nota será la misma para ambos.

Para esta práctica se deberá entregar un archivo comprimido *zip* a través del campus virtual que contenga los siguientes ficheros:

1. Un documento de respuestas con nombre de fichero `respuestas`:
   - Indicar el nombre, apellidos, y email del autor al principio. Si la práctica se hizo en pareja, indicar los datos de los dos autores.
   - El fichero debe contener las respuestas a cada uno de los ejercicios pedidos. En los casos en los que se pida completar un código solo hace falta poner en la respuesta el fragmento de código añadido.
   - El archivo de respuestas puede tener formato *txt*, *odt*, *docx*, o *pdf*.
2. Los ficheros de configuración de Asterisk creados durante la práctica:
   - `extensions_custom.conf`
   - `pjsip_custom.conf`
   - `voicemail.conf`

Apéndices

A. Comprobar si una variable está vacía

Existen varias maneras de determinar si el valor de una variable es la cadena vacía, en el siguiente ejemplo se muestran las más habituales:

```
; Function LEN
exten => s,1,Set(S=)
exten => s,2,Verbose(Is S empty?: $[${LEN(${S}) = 0]) ; Is S empty?: 1

; Function ISNULL
exten => s,1,Set(S=)
exten => s,2,Verbose(Is S empty?: ${ISNULL(${S})} ; Is S empty?: 1

; String comparison
exten => s,1,Set(S=)
exten => s,2,Verbose(Is S empty?: $["${S}" = ""]) ; Is S empty?: 1
```

B. Implementación de menús y manejo de extensiones inválidas y *timeouts*.

Para implementar un menú se deberá crear un contexto propio en el que se llamará primero a las aplicaciones `Background()` y `WaitExten()`. Estas aplicaciones esperan a que el usuario introduzca una extensión y luego saltan a la extensión introducida. Además, para manejar los casos de extensiones inválidas y *timeouts* estas aplicaciones pueden saltar a dos extensiones especiales:

- `i`: la extensión que se marcó es inválida.
- `t`: el usuario esperó demasiado sin pulsar ningún dígito.

Por ejemplo, un menú simple que nos permite llamar a sip1, sip2, o a otra extensión se puede implementar como se muestra a continuación en el código 13.

Código 13: Ejemplo de código de menú

```
[menu1]
exten => start,1,Background(press)
same => n,Background(digits/1)
same => n,Background(or-press)
same => n,Background(digits/2)
same => n,WaitExten(10)

exten => 1,1,Dial(pjsip/sip1)
exten => 2,1,Dial(pjsip/sip2)

exten => i,1,Playback(pbx-invalid)
same => n,GoTo(menu1,start,1)

exten => t,1,Playback(please-try-again)
same => n,GoTo(menu1,start,1)
```

C. Usar sonidos en español

Por defecto Asterisk utiliza los sonidos en inglés cuando se usa alguna aplicación como `Background()` o `Playback()`. Los sonidos utilizados se encuentran en `/var/lib/asterisk/sounds`, con una carpeta para cada idioma. Por ejemplo para español, debería haber una carpeta `/var/lib/asterisk/sounds/es`.

Para hacer que Asterisk use otro idioma en lugar del inglés, debe editarse el fichero `/etc/asterisk/asterisk.conf` y dentro de la sección `[options]` añadir la entrada:

```
defaultlanguage=es
```

Al relanzar Asterisk, los mensajes deberían reproducirse en español. En caso de que los sonidos en español no estén instalados, se pueden instalar desde la aplicación web de configuración de FreePBX. Para ello hay que ir a **Admin ->Sound Languages**, donde se encuentra un panel desde el que se pueden descargar directamente los sonidos en el idioma deseado.

Hay que tener en cuenta que existen diferentes paquetes de sonidos para cada idioma, siendo los sonidos más básicos los del paquete `core`. Sin embargo hay otros paquetes adicionales con mas sonidos, como por ejemplo el sonido de *feature-not-avail-line* que se usa en la práctica. Desde FreePBX no se descarga ese paquete, por lo que para hacerlo se puede visitar `https://www.asterisksounds.org/es-es/node/25635` y descargar el archivo zip con estos sonidos. Posteriormente, se debe extraer su contenido dentro de `/var/lib/asterisk/sounds/es`.