Nombre:	Felipe Cabeza Leiva
IES	Islas Filipinas

Curso:
(031) Red Hat System Administration I & II

Familia profesional: Informática

Ciclo Formativo: CFGS - Desarrollo de Aplicaciones Multiplataforma

Módulo: Programación de Servicios y Procesos

INDICE

1 Objetivo

El objetivo de esta actividad es realizar una Unidad Didáctica que pueda ser utilizada en el módulo "Programación de Servicios y Procesos" del ciclo de grado superior Desarrollo de Aplicaciones Multiplataforma.

Los objetivos que persiguen son:

- Repasar conceptos vistos en el primer curso en el módulo de sistemas informáticos
- Complementar la visión de administración de sistemas Linux desde el punto de vista del programador.

Con esta Unidad Didáctica el alumno comprenderá:

- Gestión de usuarios y grupos en Linux
- Instalación de paquetes en Linux ya que se realizará algún programa en C para lo que se tendrá que instalar el entorno de desarrollo:
  - Obtención de estado de un proceso
  - Comunicación entre procesos con pipes
- Procesos y comunicación entre procesos con pipes en Linux
- Monitorización de threads en Linux
- Servicios y demonios en Linux
- Conexiones seguras SSH

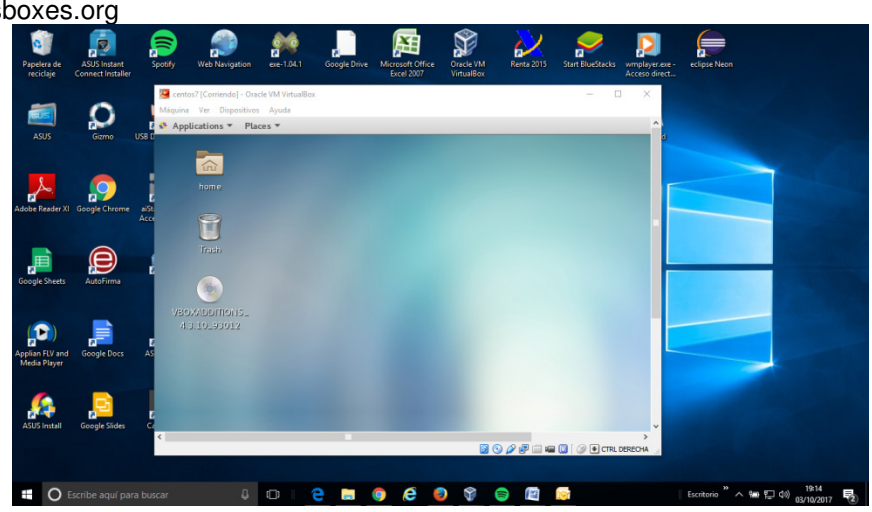
2 Desarrollo de la Unidad Didáctica

2.1 Preparación del entorno

2.1.1 Uso de máquina Virtual Centos 7 con virtual box

Se requiere tener Virtual Box instalado. Para obtener una máquina virtual Centos 7 preparada p<ara Virtual Box se ha descargado una imagen desde la url http://www.osboxes.org/centos/.
Esta máquina viene preparada con los siguientes usuarios:

- root / osboxes.org
- osboxes /osboxes.org



La primera práctica consistirá en definir usuarios nuevos y cambiar la password de los existentes entendiendo el concepto de grupo primario y grupo secundario.

2.1.2 Gestión de usuarios y grupos en Linux

Una vez tenemos un entorno ya funcionando las tareas que debemos realizar son:

2.1.2.1 Creación de un nuevo usuario

- Con el comando **useradd** podemos crear un nuevo usuario. Con useradd –help se puede consultar las opciones disponibles. La versión más básica es indicar únicamente el nombre del usuario

```
[osboxes@osboxes ~]$ su -
Password:
Last login: Tue Oct  3 17:13:31 BST 2017 on pts/0
[root@osboxes ~]# useradd dam2
[root@osboxes ~]#
```

2.1.2.2 Modificar la contraseña de un usuario

- Con el comando **useradd** podríamos haberle indicado contraseña. En este caso lo hacemos con el comando **passwd**, que permitiría cambiar la contraseña a sí mismo

```
[root@osboxes ~]# passwd dam2
Changing password for user dam2.
New password:
BAD PASSWORD: The password is shorter than 7 characters
Retype new password:
passwd: all authentication tokens updated successfully.
[root@osboxes ~]#
```

2.1.2.3 Cambiar de usuario o Switch User

- El comando su permite pasar de un usuario a otro. En las pantallas anteriores se puede observar:
  - **su [-]** : para conectarse como usuario root. Sin él -, se mantienen las variables de entorno del usuario que hace el comando

2.1.2.4 Ejecutar comandos como root

- Se pueden ejecutar comandos como root sin necesidad de cambiar de usuario. Para ello existe el comando **sudo.** Todos los comandos ejecutados usando **sudo** quedan registrados en el fichero /var/log/secure
- Los usuarios que puedan hacer **sudo** vendrá determinado por la configuración indicada en el fichero **/etc/sudoers.**

```
[osboxes@osboxes ~]$ sudo cat /etc/sudoers
[sudo] password for osboxes:
## Sudoers allows particular users to run various commands as
## the root user, without needing the root password.
##
## Examples are provided at the bottom of the file for collections
## of related commands, which can then be delegated out to particular
## users or groups.
##
## This file must be edited with the 'visudo' command.
## Allow root to run any commands anywhere
root    ALL=(ALL)       ALL

## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys ALL = NETWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESS

## Allows people in group wheel to run all commands
%wheel  ALL=(ALL)       ALL
```

- En este contenido se puede ver que los usuarios que pertenezcan al grupo wheel tendrán acceso a ejecutar todos los comandos. Por tanto haremos que nuestro usuario dam2 pertenezca a este grupo, asignándoselo como grupo secundario.

2.1.2.5 Modificar o añadir un grupo a un usuario

- El comando **groupadd** nos permite añadir un nuevo grupo. En nuestro caso se va a añadir a usuario dam2 al grupo wheel
- Con el comando **usermod –aG** podemos añadir un grupo secundario a un usuario. Con el comando **usermod –g** modificaríamos el grupo primario de un usuario

```
[osboxes@osboxes ~]$ usermod -aG wheel dam2
bash: /usr/sbin/usermod: Permission denied
[osboxes@osboxes ~]$ sudo usermod -aG wheel dam2
[sudo] password for osboxes:
Sorry, try again.
[sudo] password for osboxes:
[osboxes@osboxes ~]$
```

- En la pantalla anterior se muestra que se debe ejecutar el comando como root

2.1.2.6 ¿Dónde se almacena la información de los usuarios?

- En el fichero /etc/passwd se almacena la información de los usuarios excepto la password. Se pude ver en la siguiente pantalla la información que se almacena:

```
File  Edit  View  Search  Terminal  Help
[dam2@osboxes ~]$ sudo tail -3 /etc/passwd
osboxes:x:1000:1000:osboxes.org:/home/osboxes:/bin/bash
vboxadd:x:988:1::/var/run/vboxadd:/bin/false
dam2:x:1001:1001::/home/dam2:/bin/bash
[dam2@osboxes ~]$
```

- Para el usuario dam2 podemos ver que tiene:
  - Uid: 1001
  - Gid: 1001 (identificador de grupo primario)
  - El directorio home es /home/dam2
  - La Shell o intérprete de comandos que se ejecutará cuando el usuario abra un terminal es /bin/bash. Observar que el usuario vboxadd, creado para virtualBox, no puede abrir un terminal ya que su la shell que ejecutaría es /bin/false

2.1.2.7 Obtener información del usuario

- Comando id para obtener información del usuario con el que estamos conectados o con un usuario que indiquemos. En este caso obtendremos la información del usuario dam2

```
[dam2@osboxes ~]$ id
uid=1001(dam2) gid=1001(dam2) groups=1001(dam2),10(wheel)
nconfined_r:unconfined_t:s0-s0:c0.c1023
[dam2@osboxes ~]$
```

- En la información anterior hay que resaltar:
  - Linux asigna un UID a cada usuario, en este caso 1001 (Linux reserva los rangos 1000+ para usuarios regulares)
  - El grupo primario al que pertenece el usuario es 1001 (dam2) y al grupo secundario 10 (wheel)

2.1.2.8 ¿Dónde se almacena la información de los grupos?

- En el fichero /etc/group se almacena la información de los usuarios. En este caso se va a buscar qué usuarios están asignados al grupo Wheel

```
[dam2@osboxes ~]$ sudo cat /etc/group | grep wheel
wheel:x:10:osboxes,dam2
[dam2@osboxes ~]$
```

2.1.2.9 ¿Dónde se almacenan las passwords?

- En versiones anteriores las passwords se almacenaban en el fichero /etc/passwd, de forma encriptada. Ahora las passwords se almacenan, también de forma encriptada mediante una función hash, en el fichero /etc/shadow.
- Con las passwords también se pueden establecer políticas de caducidad y otras características en este fichero.
- El algoritmo para encriptar las paswords puede ser cambiado por el usuario root con el comando **authconfig –passalgo** y alguno de los argumentos md5, sha256, sha512

```
[dam2@osboxes ~]$ sudo tail -3 /etc/shadow
[sudo] password for dam2:
osboxes:$6$5nSn1HH4/vjaabNj$vrXY1n0Z4aqufQkgH32CXlThozg9CS3IahaEawP4h60yLIj7TbkpXQDeTCGg1KGq2.ah84D3nZWR7XoqqMjc
h/::0:99999:7:::
vboxadd:!!:17442::::::
dam2:$6$2TKu8v62$nz711ZrGGY95BEY4XMCtOjre8GmSof4I0ynqxmCDxWDY26481yRG3Dy6ddK3yKER7rTurU7KARfeyEnvAoPpv/:17442:0:
99999:7:::
[dam2@osboxes ~]$
```

- El $6 al inicio indica que centos 7 utiliza por defecto SHA-512

2.2 Instalación de paquetes

2.2.1 Instalación del compilador C

Para hacer un pequeño programa que ayude al alumno a entender un proceso se necesita instalar un compilador del lenguaje de programación c (gcc)

La diferencia entre usar **yum** o **rpm** para instalar un paquete, es la resolución de dependencias extras, si éstas no se encuentran instaladas en el sistema *yum* **automáticamente** las buscará en los repositorios y las instalará junto con el paquete además de que el mismo será registrado en la base de datos de YUM

2.2.2 Comprobar si está instalado

- Comando rpm : “rpm –q gcc”

```
File  Edit  View  Search  Terminal  Help
[root@osboxes ~]# rpm -q gcc
package gcc is not installed
[root@osboxes ~]#
```

- También podemos comprobar con comando yum
    - Con comando “yum list ‘gcc*’” podemos buscar paquetes instalados o disponibles que contengan gcc

```
[root@osboxes ~]# yum list 'gcc*'
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: mirror.tedra.es
 * extras: mirror.tedra.es
 * updates: mirror.tedra.es
Available Packages
gcc.x86_64                                4.8.5-16.el7                    base
gcc-c++.x86_64                            4.8.5-16.el7                    base
gcc-gfortran.x86_64                       4.8.5-16.el7                    base
gcc-gnat.x86_64                           4.8.5-16.el7                    base
gcc-go.x86_64                             4.8.5-16.el7                    base
gcc-objc.x86_64                           4.8.5-16.el7                    base
gcc-objc++.x86_64                         4.8.5-16.el7                    base
gcc-plugin-devel.x86_64                   4.8.5-16.el7                    base
[root@osboxes ~]#
```

    - Con comando “yum info gcc” podemos saber la información del paquete que lo contiene

```
[root@osboxes ~]# yum info gcc
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: mirror.tedra.es
 * extras: mirror.tedra.es
 * updates: mirror.tedra.es
Available Packages
Name        : gcc
Arch        : x86_64
Version     : 4.8.5
Release     : 16.el7
Size        : 16 M
Repo        : base/7/x86_64
Summary     : Various compilers (C, C++, Objective-C, Java, ...)
URL         : http://gcc.gnu.org
License     : GPLv3+ and GPLv3+ with exceptions and GPLv2+ with exceptions and
            : LGPLv2+ and BSD
Description : The gcc package contains the GNU Compiler Collection version 4.8.
            : You'll need this package in order to compile C code.

[root@osboxes ~]#
```

- Con yum también podemos ver "conjuntos de software" para un propósito específico. Con el comando "yum group list" se mostrará la lista de agrupaciones de software instaladas y disponibles. Con este comando se puede encontrar que el grupo "Development Tools" está disponible.

```
[root@osboxes ~]# yum group list
Loaded plugins: fastestmirror, langpacks
There is no installed groups file.
Maybe run: yum groups mark convert (see man yum)
Loading mirror speeds from cached hostfile
 * base: mirror.tedra.es
 * extras: mirror.tedra.es
 * updates: mirror.tedra.es
Available Environment Groups:
   Minimal Install
   Compute Node
   Infrastructure Server
   File and Print Server
   Basic Web Server
   Virtualization Host
   Server with GUI
   GNOME Desktop
   KDE Plasma Workspaces
   Development and Creative Workstation
Available Groups:
   Compatibility Libraries
   Console Internet Tools
   Development Tools
   Graphical Administration Tools
   Legacy UNIX Compatibility
   Scientific Support
   Security Tools
   Smart Card Support
   System Administration Tools
   System Management
Done
```

2.2.3 Instalación de compilador

- Se elije la opción de instalar el grupo de software llamado "Development Tools". El coste es que vamos a instalar un entorno de desarrollo que, entre otros compiladores, tendrá el gcc.

```
[root@osboxes ~]# yum groupinstall "Development Tools" > salida
There is no installed groups file.
Maybe run: yum groups mark convert (see man yum)
```

```
Loaded plugins: fastestmirror, langpacks
Loading mirror speeds from cached hostfile
 * base: mirror.tedra.es
 * extras: mirror.tedra.es
 * updates: mirror.tedra.es
Resolving Dependencies
--> Running transaction check
---> Package autoconf.noarch 0:2.69-11.el7 will be installed
--> Processing Dependency: m4 >= 1.4.14 for package: autoconf-2.69-11.el7.noarch
--> Processing Dependency: perl(Data::Dumper) for package: autoconf-2.69-11.el7.
noarch
```

....

```
Dependencies Resolved

================================================================================
 Package                 Arch       Version                   Repository   Size
================================================================================
Installing for group install "Development Tools":
 autoconf                noarch     2.69-11.el7               base        701 k
 automake                noarch     1.13.4-3.el7              base        679 k
 bison                   x86_64     3.0.4-1.el7               base        674 k
 byacc                   x86_64     1.9.20130304-3.el7        base         65 k
 cscope                  x86_64     15.8-10.el7               base        203 k
 ctags                   x86_64     5.8-13.el7                base        155 k
 diffstat                x86_64     1.57-4.el7                base         35 k
 doxygen                 x86_64     1:1.8.5-3.el7             base        3.6 M
 flex                    x86_64     2.5.37-3.el7              base        292 k
 gcc                     x86_64     4.8.5-16.el7              base         16 M
 gcc-c++                 x86_64     4.8.5-16.el7              base        7.2 M
```

...

```
 systemtap-runtime       x86_64     3.1-3.el7                 base        394 k

Transaction Summary
================================================================================
Install  25 Packages (+29 Dependent packages)
Upgrade              ( 13 Dependent packages)

Total download size: 106 M
Is this ok [y/d/N]: Exiting on user command
Your transaction was saved, rerun it with:
 yum load-transaction /tmp/yum_save_tx.2017-10-04.00-10.hl3s64.yumtx
[root@osboxes ~]#
```

- Con el comando “yum install –y gcc” sólo se habría instalado el gcc resolviendo dependencias.

```
Installed:
  gcc.x86_64 0:4.8.5-16.el7

Dependency Installed:
  cpp.x86_64 0:4.8.5-16.el7
  glibc-devel.x86_64 0:2.17-196.el7
  glibc-headers.x86_64 0:2.17-196.el7
  kernel-headers.x86_64 0:3.10.0-693.2.2.el7
  libmpc.x86_64 0:1.0.1-3.el7

Dependency Updated:
  glibc.x86_64 0:2.17-196.el7          glibc-common.x86_64 0:2.17-196.el7
  libgcc.x86_64 0:4.8.5-16.el7         libgomp.x86_64 0:4.8.5-16.el7

        Complete!

root@osboxes ~]#
```

2.3 Procesos y Threads en Linux

2.3.1 Introducción a procesos y threads

En programación concurrente hay dos unidades básicas de ejecución: procesos y threads.
Podemos resumir la diferencia entre procesos y threads de una forma sencilla:

- 1 proceso es una unidad independiente de otro proceso. Cada proceso tiene su propio espacio de memoria. Los procesos se suelen ver como un programa en ejecución, pero hay que tener cuidado con esta afirmación ya que 1 programa puede desencadenar varios procesos mediante instrucciones fork().
- 1 hilo se ejecuta dentro de un proceso. Podemos decir que un proceso tiene al menos un hilo. Varios hilos o threads que se ejecutan dentro de un proceso tienen una zona de memoria compartida
- El cambio de contexto de un proceso (con todos sus datos almacenados en lo que se llama PCB o Process Control Block) es más costoso para el sistema operativo que el cambio de contexto de thread (con su datos almacenados en lo que se llama TCB o Thread Control Block)

Referencia: https://docs.oracle.com/javase/tutorial/essential/concurrency/procthread.html

2.3.2 Monitorización de procesos.

2.3.2.1 Comando ps

El comando ps soporta tres opciones de formato. Consultar **"man ps"** en un terminal**:**

- Unix (POSIX). Opciones adicionales que deben estar precedidas por un -. Ej.
  - ps –ef : muestra todos los procesos existentes (e) con detalle(f)

```
[dam2@osboxes ~]$ ps -ef | head
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 18:08 ?        00:00:02 /usr/lib/systemd/systemd --switc
hed-root --system --deserialize 21
root         2     0  0 18:08 ?        00:00:00 [kthreadd]
root         3     2  0 18:08 ?        00:00:00 [ksoftirqd/0]
root         5     2  0 18:08 ?        00:00:00 [kworker/0:0H]
root         7     2  0 18:08 ?        00:00:00 [migration/0]
root         8     2  0 18:08 ?        00:00:00 [rcu_bh]
root         9     2  0 18:08 ?        00:00:01 [rcu_sched]
root        10     2  0 18:08 ?        00:00:00 [watchdog/0]
root        12     2  0 18:08 ?        00:00:00 [khelper]
[dam2@osboxes ~]$
```

  - Se añade |head para ver sólo los primeros procesos de la lista

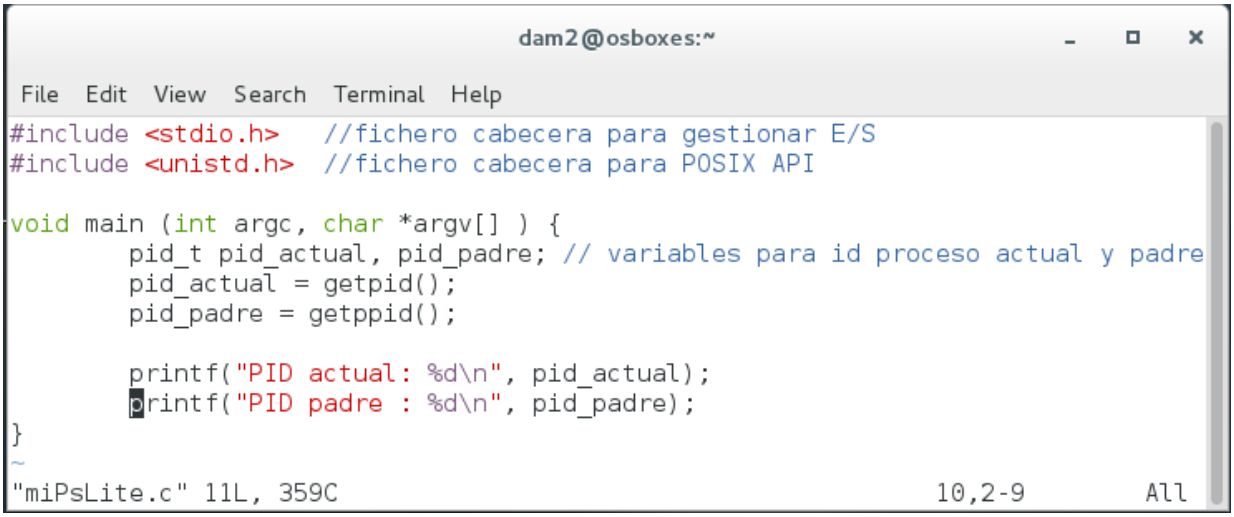
- BSD ("Berkeley Software Distribution". Así se llamó a las distribuciones de código fuente que se hicieron en la Universidad de Berkeley en California y que en origen eran extensiones del sistema operativo UNIX® de AT&T Research) Opciones adicionales que no van precedidas por -
  - ps aux : igual que el comando anterior
    - ax: muestra los procesos de todos los usuarios, incluso los que no estén asociados a ningún terminal.
    - u: muestra columnas adicionales, como el usuario al que pertenece el proceso, tiempo de ejecución, etcétera.
  - Nota: ps –aux no es lo mismo que ps aux

- GNU long options: opciones precedidas por 2 '-'

2.3.2.2 Ejemplo programa C que muestra información de proceso

A continuación se propone al alumno que realice un programa c que pueda obtener información de un proceso, concretamente el identificador de proceso y el identificador de proceso del padre

- Con el editor vi crear un fichero llamado miPsLite.c en el directorio home del usuario dam2



- Compilar el fichero con el compilador "gcc". Se genera un fichero mipsLite

```
[dam2@osboxes ~]$ vi miPsLite.c
[dam2@osboxes ~]$ gcc miPsLite.c -o miPsLite
[dam2@osboxes ~]$
```

- Comprobar fichero generado y qué permisos tiene. Se puede comprobar que tiene permisos de ejecución para el usuario dam2 y para el grupo dam2 y para el resto de usuarios.

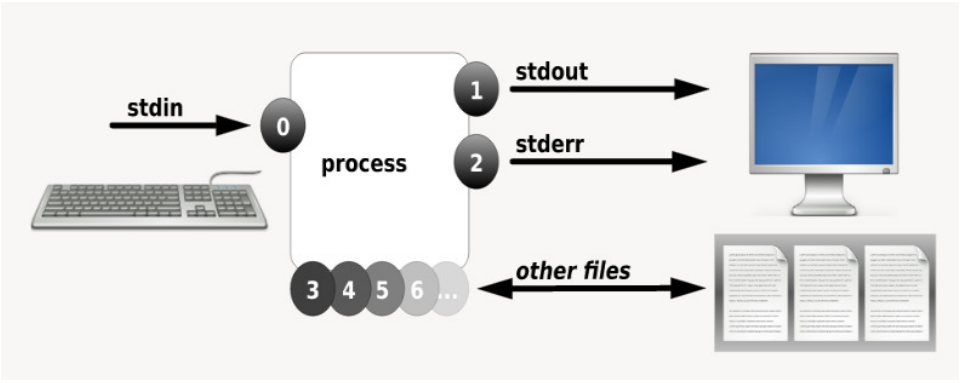
```
[dam2@osboxes ~]$ ls -al | grep miP
-rwxrwxr-x. 1 dam2 dam2 8624 Oct  4 00:22 miPsLite
-rw-rw-r--. 1 dam2 dam2  355 Oct  4 00:22 miPsLite.c
[dam2@osboxes ~]$
```

- Ejecución del programa creado: ./miPsLite

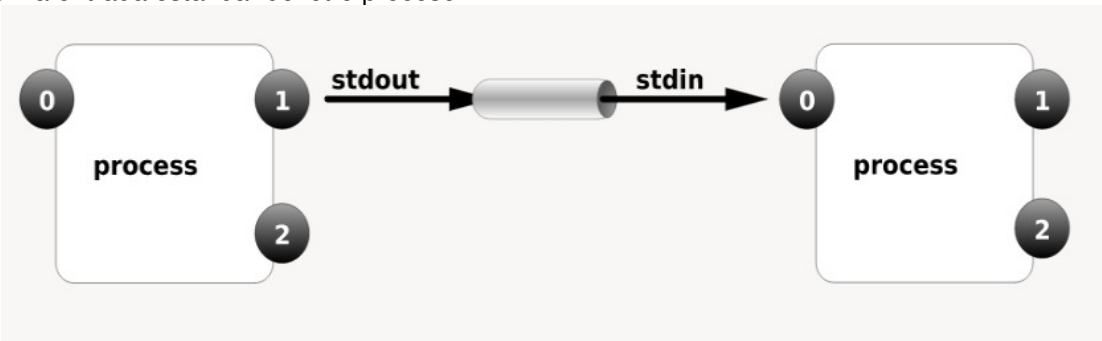
```
[dam2@osboxes ~]$ ./miPsLite
PID actual: 8713
PID padre : 8521
[dam2@osboxes ~]$
```

2.3.3 Comunicación entre procesos con pipes

En un proceso se cuenta con entrada estándar (stdin), salida estándar (stdout) y salida de error (stderr)



Se puede comunicar 2 procesos haciendo uso de tuberías o pipes. Consiste en conectar la salida estándar de un proceso con la entrada estándar del otro proceso



2.3.3.1 Ejemplo de tubería sin nombre

Es habitual que se creen tuberías para comunicar procesos en comandos rutinarios.

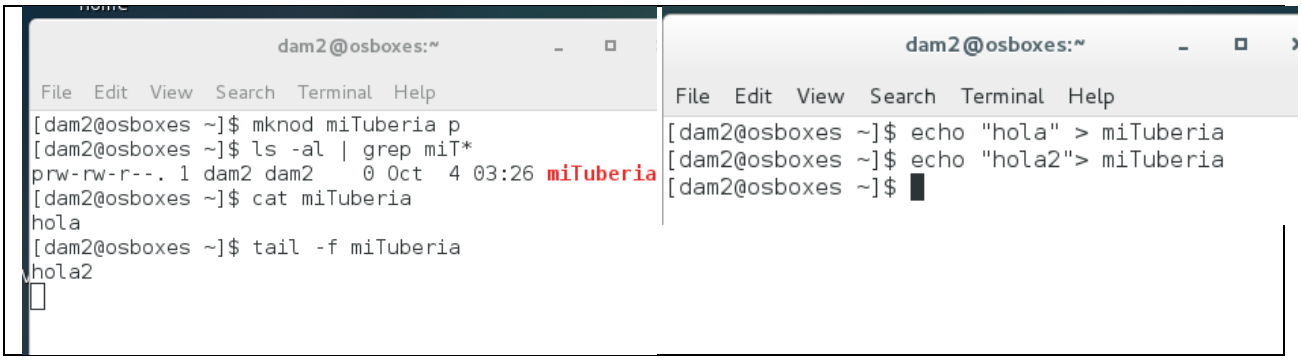
- Ejemplo: listar el contenido de un directorio y mostrar solo los ficheros que empiecen por miP. En este ejemplo se crea una tubería en la que la salida del comando ls se conecta con la entrada del comando grep. Es una tubería sin nombre

```
[dam2@osboxes ~]$ ls -al | grep miP.*
-rwxrwxr-x. 1 dam2 dam2 8624 Oct  4 00:27 miPsLite
-rw-rw-r--. 1 dam2 dam2  359 Oct  4 00:27 miPsLite.c
[dam2@osboxes ~]$
```

2.3.3.2 Ejemplo de tubería con nombre

Seguir los siguientes pasos:

- Conectarse con usuario dam2
- En el home del usuario crear una tubería llamada miTuberia con el comando "**mknod miTuberia p**"
- Visualizar con ls –al el contenido del directorio. Se podrá ver el tipo "p" en la entrada.
- Visualizar el contenido de miTuberia con el comando cat o tail
- En otra ventana volcar sobre miTuberia y comprobar que se muestra en la otra terminal el contenido.
- Finalmente, con el comando rm, eliminar la tubería



2.3.3.3 Ejemplo de programa C para comunicar procesos con Pipes

Se muestra a continuación un ejemplo en C que comunica 2 procesos con tuberías.

Hay que fijarse que la creación de la tubería siempre es anterior a la creación del proceso hijo. Tras la llamada fork, el proceso hijo, que es una copia del padre, se lleva también una copia de la tabla de descriptores de fichero. Por tanto, padre e hijo disponen de acceso a los descriptores que permiten operar con la tubería. Dado que las tuberías son un mecanismo unidireccional, es necesario que únicamente uno de los procesos escribe, y que el otro únicamente lea.

En este código, la tubería "p" se hereda al hacer el fork() que da lugar al proceso hijo, pero es necesario que el padre haga un close() de p[0] (el lado de lectura de la tubería), y el hijo haga un close() de p[1] (el lado de escritura de la tubería). Una vez hecho esto, los dos procesos pueden emplear la tubería para comunicarse (siempre unidireccionalmente), haciendo write() en p[1] y read() en p[0], respectivamente.

```
#include <sys/types.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>

#define SIZE 512

int main( int argc, char **argv )
{
  pid_t pid;
  int p[2], readbytes;
  char buffer[SIZE];

  pipe( p );

  if ( (pid=fork()) == 0 )
  { // hijo
    close( p[1] ); /* cerramos el lado de escritura del pipe */

    while( (readbytes=read( p[0], buffer, SIZE )) > 0)
      write( 1, buffer, readbytes );

    close( p[0] );
  }
  else
  { // padre
    close( p[0] ); /* cerramos el lado de lectura del pipe */

    strcpy( buffer, "Esto llega a traves de la tuberia\n" );
    write( p[1], buffer, strlen( buffer ) );

    close( p[1] );
  }
  waitpid( pid, NULL, 0 );
  exit( 0 );
}
```

2.3.4 Control de jobs

Job control es una característica de la shell que permite ejecutar y administrar múltiples comandos desde una sola instancia de Shell. Un job está asociado con cada tubería ingresada en un shell prompt. Todos los procesos de esa tubería son parte del job y son miembros del mismo grupo de procesos. Lo mínimo para esta tubería es teclear un solo comando.

Cada terminal es su propia sesión y sólo puede tener un proceso en primer plano y procesos en segundo plano independientes.

2.3.4.1 Ejecutar jobs en segundo plano:

Son procesos que se ejecutan asociados a un terminal (se ve en la columna TTY del comando ps) y que no pueden leer entradas ni recibir interrupciones generadas desde el teclado desde el terminal, pero sí pueden escribir en el terminal. . Un job en segundo plano o proceso puede suspenderse o puede estar ejecutándose.

- Ejemplo: **Comando watch ps**. El comando watch arrancará el proceso ps cada 2 segundos. Si finalizamos con &, se ejecutará en segundo plano. Nos devuelve un id de proceso (10222 y 10226) y orden [1] [2] dentro de los jobs que hay en segundo plano

```
[osboxes@osboxes ~]$ ps
  PID TTY          TIME CMD
10163 pts/0    00:00:00 bash
10218 pts/0    00:00:00 ps
[osboxes@osboxes ~]$ watch ps &
[1] 10222

[1]+  Stopped                 watch ps
[osboxes@osboxes ~]$ watch ps &
[2] 10226

[2]+  Stopped                 watch ps
[osboxes@osboxes ~]$ ps
  PID TTY          TIME CMD
10163 pts/0    00:00:00 bash
10222 pts/0    00:00:00 watch
10226 pts/0    00:00:00 watch
10238 pts/0    00:00:00 ps
[osboxes@osboxes ~]$
```

2.3.4.2 Devolver un job a primer plano:

Con el comando **jobs** se pueden mostrar los trabajos que están ejecutándose en segundo plano y con el comando **fg %x** se puede recuperar un job para que se ejecute en primero plano

```
[osboxes@osboxes ~]$ jobs
[1]-  Stopped                 watch ps
[2]+  Stopped                 watch ps
[osboxes@osboxes ~]$ fg %1
```

2.3.4.3 Enviar un proceso en primer plano a segundo plano:

Con la petición Ctrl+z sobre el terminal (petición de suspensión) un proceso se pasa a segundo plano y es suspendido (estado T en columna S)

```
[dam2@osboxes ~]$ sleep 1000
^Z
[1]+  Stopped                 sleep 1000
[dam2@osboxes ~]$ ps -al
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0 10492 10163  0  80   0 - 53044 wait   pts/0    00:00:00 su
4 S  1001 10508 10492  0  80   0 - 29013 wait   pts/0    00:00:00 bash
0 T  1001 10552 10508  0  80   0 - 26977 signal pts/0    00:00:00 sleep
0 R  1001 10553 10508  0  80   0 - 37236 -      pts/0    00:00:00 ps
[dam2@osboxes ~]$
```

Con el comando **ps j** también podemos mostrar información sobre los jobs. El PGID es el PID del proceso líder, normalmente el primer proceso en la tubería de la shell

```
[dam2@osboxes ~]$ ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
10492 10508 10508 10163 pts/0    10586 S     1001   0:00 -bash
10508 10552 10552 10163 pts/0    10586 T     1001   0:00 sleep 1000
10508 10586 10586 10163 pts/0    10586 R+    1001   0:00 ps j
[dam2@osboxes ~]$
```

2.3.4.4 Reanudar un proceso suspendido en segundo plano

Para reanudar un proceso que tengamos suspendido en segundo plano, se puede usar el comando **bg** con el mismo id de job. Se verá que después de reanudarlo deja de estar en estado T

```
[dam2@osboxes ~]$ jobs
[1]+  Stopped                 sleep 1000
[dam2@osboxes ~]$ bg %1
[1]+ sleep 1000 &
[dam2@osboxes ~]$ ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
10492 10508 10508 10163 pts/0    10604 S     1001   0:00 -bash
10508 10552 10552 10163 pts/0    10604 S     1001   0:00 sleep 1000
10508 10604 10604 10163 pts/0    10604 R+    1001   0:00 ps j
[dam2@osboxes ~]$
```

2.3.5 Monitorización de threads en Linux

Se puede afirmar que un proceso siempre tiene al menos un thread y que, en general los procesos no comparten memoria.

Sin embargo un proceso puede tener más de un thread y sí existe una memoria compartida por todos los threads de un proceso. Si el proceso finaliza todos los threads finalizan también.

Para comprobar que un proceso genera varias hebras se puede hacer con el comando

- ps –eO nlwp (nlwp atiende a number of lightweight processes)

Como ejemplo se va a crear un programa en java, que sólo haga “hola mundo” y se va a mostrar cuántas hebras son generadas en el proceso correspondiente. Como nota, de momento, resaltar que una de esas hebras será el main del programa y otra de ellas será el “garbage collector”

2.3.5.1 Instalación de java jdk

- Comando : sudo yum install java-1.8.0-openjdk*

2.3.5.2 Edición y compilación de programa java

Introducimos un sleep para que el proceso no finalice inmediatamente

```
public class HolaMundo {
        public static void main(String[] args)
                throws InterruptedException {

                System.out.println("Hola Mundo");
                Thread.sleep(10000);
        }
}
~
```

```
[dam2@osboxes ~]$ javac HolaMundo.java
[dam2@osboxes ~]$
```

2.3.5.3 Ejecución y monitorización de programa java

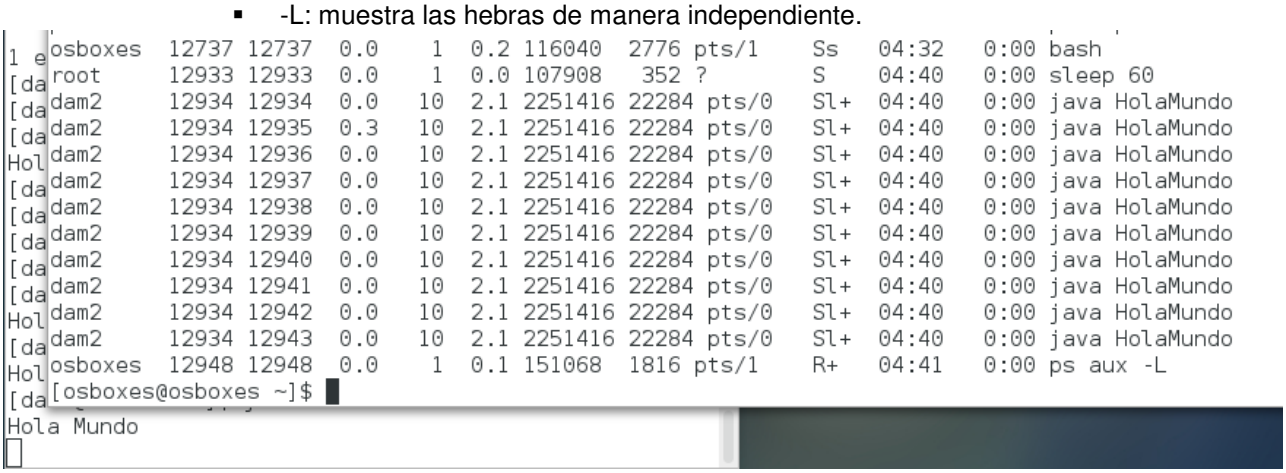
- Con comando java HolaMundo se ejecuta el programa. Nos mostrará “Hola Mundo” en el terminal y se quedará un tiempo “dormido”
- En otra ventana con el comando ps
  - Comprobamos el número de proceso

```
[osboxes@osboxes ~]$ ps -ef | grep java
dam2     12787 12095  1 04:33 pts/0    00:00:00 java HolaMundo
osboxes  12806 12737  0 04:33 pts/1    00:00:00 grep --color=auto java
[osboxes@osboxes ~]$
```

- Comprobamos el número de threads del proceso. Se puede observar que hay 10 hebras para el proceso "java HolaMundo"

```
  PID NLWP S TTY          TIME COMMAND
    1    1 S ?        00:00:05 /usr/lib/systemd/systemd --system --deserializ
    2    1 S ?        00:00:00 [kthreadd]
12845    1 S ?         00:00:00 sleep 60
12865   10 S pts/0     00:00:00 java HolaMundo
12875    1 R pts/1     00:00:00 ps -eO nlwp
[osboxes@osboxes ~]$
```

- El comando "ps aux –L" muestra las hebras de manera independiente
  - ax: muestra los procesos de todos los usuarios, incluso los que no estén asociados a ningún terminal.
  - u: muestra columnas adicionales, como el usuario al que pertenece el proceso, tiempo de ejecución, etcétera.
  - -L: muestra las hebras de manera independiente.



- En la columna STAT se ve estado de cada hebra. Algunas de las letras que pueden aparecer para indicar el estado son:
  - D: uninterruptible sleep (normalmente por E/S)
  - R: en ejecución, o preparado a la espera de procesador.
  - S: interruptible sleep (esperando a que ocurra algún evento).
  - Z: zombie.
- También pueden aparecer otras con información extra:
  - s: lider de sesión.
  - l: multihebra.
  - +: proceso en primer plano (en su terminal).
  - <: proceso con prioridad alta.
  - N: proceso con prioridad baja.
  - L: proceso con páginas bloqueadas en memoria.

2.4 Servicios y demonios en Linux

Un demonio o servicio es un programa que se ejecuta en segundo plano, fuera del control interactivo de los usuarios del sistema ya que carecen de interfaz con estos. El término demonio se usa fundamentalmente en sistemas UNIX y basados en UNIX, como GNU/Linux o Mac OS X. En Windows y otros sistemas operativos se denominan servicios porque fundamentalmente son programas que ofrecen servicios al resto del sistema.

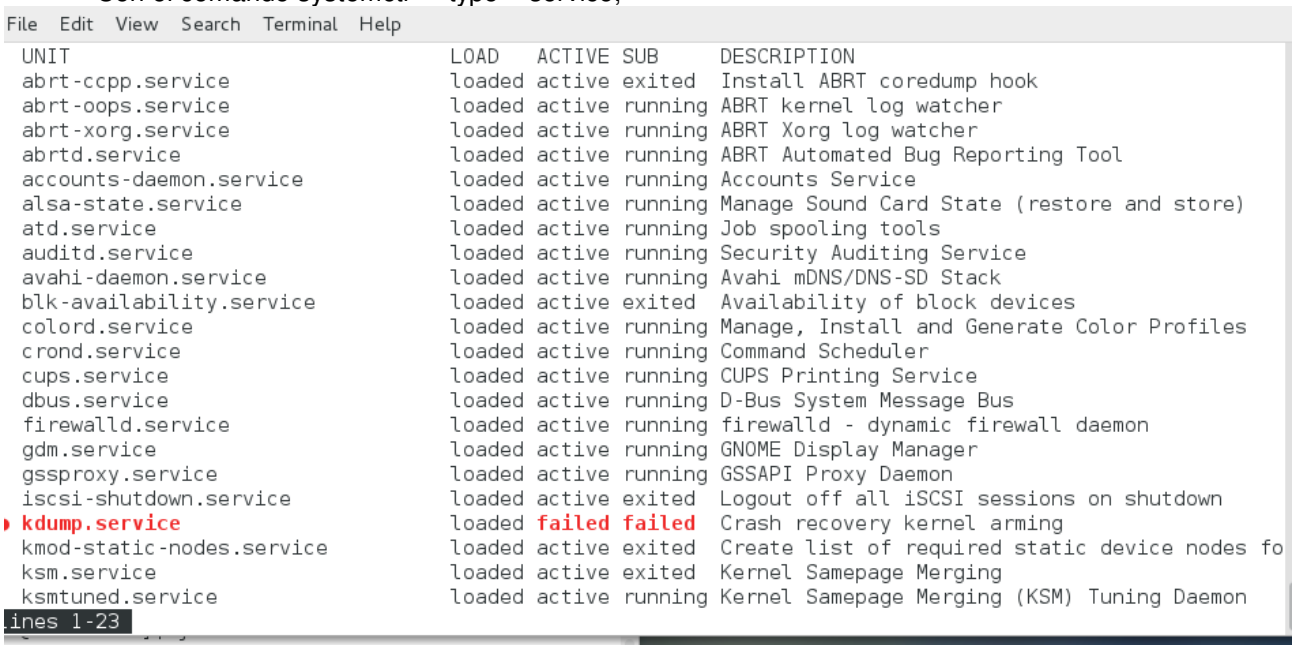
El sistema generalmente inicia los demonios durante el arranque, siendo las funciones más comunes de estos las de ofrecer servicios a otros programas, ya sea respondiendo a las peticiones que llegan a través de la red o atendiendo a procesos que se ejecutan en el mismo sistema, así como responder ante cierta actividad del hardware

Tradicionalmente en sistemas UNIX y derivados los nombres de los demonios terminan con la letra d. Por ejemplo syslogd es el demonio que implementa el registro de eventos del sistema, mientras que sshd es el que sirve a las conexiones SSH entrantes.

En Red Hat Linux 7 y Centos 7 el primer proceso de arranque del sistema es **systemd** que sustituye al antiguo init. Durante años, **init** ha sido el proceso de arranque de sistemas Linux y era el encargado de activar otros servicios en el sistema

2.4.1 Lista de servicios

- Con el comando systemctl - - type = service;



2.4.2 Estado de un servicio

- Con el comando systemctl status sshd.service

```
[dam2@osboxes ~]$ systemctl status sshd.service
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: active (running) since Tue 2017-10-03 18:09:29 BST; 10h ago
     Docs: man:sshd(8)
           man:sshd_config(5)
 Main PID: 950 (sshd)
   CGroup: /system.slice/sshd.service
           └─950 /usr/sbin/sshd

Oct 03 18:09:27 osboxes systemd[1]: Starting OpenSSH server daemon...
Oct 03 18:09:29 osboxes sshd[950]: Server listening on 0.0.0.0 port 22.
Oct 03 18:09:29 osboxes sshd[950]: Server listening on :: port 22.
Oct 03 18:09:29 osboxes systemd[1]: PID file /var/run/sshd.pid not readable (yet?) after start.
Oct 03 18:09:29 osboxes systemd[1]: Started OpenSSH server daemon.
[dam2@osboxes ~]$
```

2.4.3 Inicio, arranque y reload de servicio

- Con el comando systemctl, indicando start o stop. O Bien con un solo comando restart
- Parada servicio sshd

```
[dam2@osboxes ~]$ systemctl stop sshd.service
[dam2@osboxes ~]$ systemctl status sshad.service
Unit sshad.service could not be found.
[dam2@osboxes ~]$ systemctl status sshd.service
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: inactive (dead) since Wed 2017-10-04 05:11:03 BST; 16s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
 Main PID: 950 (code=exited, status=0/SUCCESS)

Oct 03 18:09:27 osboxes systemd[1]: Starting OpenSSH server daemon...
Oct 03 18:09:29 osboxes sshd[950]: Server listening on 0.0.0.0 port 22.
Oct 03 18:09:29 osboxes sshd[950]: Server listening on :: port 22.
Oct 03 18:09:29 osboxes systemd[1]: PID file /var/run/sshd.pid not readable (yet?) after start.
Oct 03 18:09:29 osboxes systemd[1]: Started OpenSSH server daemon.
Oct 04 05:11:02 osboxes systemd[1]: Stopping OpenSSH server daemon...
Oct 04 05:11:03 osboxes sshd[950]: Received signal 15; terminating.
Oct 04 05:11:03 osboxes systemd[1]: Stopped OpenSSH server daemon.
[dam2@osboxes ~]$
```

- Arranque servicio sshd

```
[dam2@osboxes ~]$ systemctl start sshd.service
[dam2@osboxes ~]$ systemctl status sshd.service
● sshd.service - OpenSSH server daemon
   Loaded: loaded (/usr/lib/systemd/system/sshd.service; enabled; vendor preset: enabled)
   Active: active (running) since Wed 2017-10-04 05:12:50 BST; 3s ago
     Docs: man:sshd(8)
           man:sshd_config(5)
  Process: 13568 ExecStart=/usr/sbin/sshd $OPTIONS (code=exited, status=0/SUCCESS)
 Main PID: 13571 (sshd)
   CGroup: /system.slice/sshd.service
           └─13571 /usr/sbin/sshd

Oct 04 05:12:50 osboxes systemd[1]: Starting OpenSSH server daemon...
Oct 04 05:12:50 osboxes systemd[1]: PID file /var/run/sshd.pid not readable (yet?) after start.
Oct 04 05:12:50 osboxes sshd[13571]: Server listening on 0.0.0.0 port 22.
Oct 04 05:12:50 osboxes sshd[13571]: Server listening on :: port 22.
Oct 04 05:12:50 osboxes systemd[1]: Started OpenSSH server daemon.
[dam2@osboxes ~]$
```

- Con la opción reload se puede conseguir que el servicio vuelva a leer el fichero de configuración si ha habido cambios para no tener que detenerlo

2.5 Servicio OpenSSH

El término OpenSSH hace referencia a la implementación del software Secure Shell (SSH) que se utiliza en el sistema OpenSSH Secure Shell.
SSH trabaja de forma similar a como se hace con telnet. La diferencia principal es que SSH usa técnicas de cifrado que hacen que la información que viaja por el medio de comunicación vaya de manera no legible.

El comando **ssh** es el comando Linux que se usa normalmente para iniciar sesión de manera segura en un sistema remoto. También se puede utilizar **ssh** para ejecutar un comando individual en un sistema remoto.

En esta tarea se pretende que el alumno conozca cómo se puede configurar la autenticación de ssh usando un esquema de claves privada y pública.

2.5.1.1 Generación de las claves (ssh-keygen)

Los usuarios pueden autenticarse en una sesión de ssh sin una contraseña si utilizan autenticación mediante llave pública.
El comando ssh-keygen genera la clave privada y la clave pública

- Fichero por defecto de clave privada: ~/.ssh/id_rsa
- Fichero por defecto de clave pública: ~/.ssh/id_rsa.pub

```
[dam2@osboxes ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/dam2/.ssh/id_rsa):
Created directory '/home/dam2/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/dam2/.ssh/id_rsa.
Your public key has been saved in /home/dam2/.ssh/id_rsa.pub.
The key fingerprint is:
ab:6a:bf:21:30:55:ff:bd:ad:04:22:4d:de:6e:24:d0 dam2@osboxes
The key's randomart image is:
+--[ RSA 2048]----+
|      .          |
|     . o         |
|    . . E        |
|   .   = o .     |
|  o   . S = .    |
|   o   . * . o   |
|    . . . o o .  |
|     .. o . . .  |
|    ..o+.    .   |
+-----------------+
[dam2@osboxes ~]$
```

2.5.2 Transferencia de clave a sistema destino (ssh-copy-id)

Para poder usar la autenticación a través de claves, la clave pública ha de ser transferida al sistema destino. Esto puede hacerse con ssh-copy-id

- Comando : ssh-copy-id –i ~/.ssh/id_rsa.pub root@ubuntu.com
- Una vez transferido se podrá hacer logon remoto sin password

2.5.3 Personalización de la configuración del servicio SSH

El archivo de configuración está en /etc/ssh/sshd_config. Por cada cambio que se haga habrá que reanudar el servicio con:

- systemctl restart sshd

Ejemplos de parámetros:

- Se puede prohibir el inicio en el sistema como root
  - PermitRootLogin no
- Impedir acceso a través de contraseña
  - PasswordAuthentication yes

3 Temporalización

La temporalización del módulo de Programación de Servicios y Procesos es como sigue:

UNIDADES DE TRABAJO	Trimestre	Sesiones
UT1. Programación multiproceso	I	8
UT2. Programación multihilo	I	24
UT3. Programación de comunicaciones en red	I/II	24
UT4. Generación de servicios en red	II	26
UT5. Utilización de técnicas de programación seguras	II	8

Según este contenido se considera adecuado que esta actividad forme parte de la UT1, excepto el apartado relacionado con Servicio OpenSSH que tiene cabida en la UT5 como ejemplo de utilización de autenticación mediante clave privada-clave pública.