SO PRACTICAS

Ingeniería Infórmatica
Universidad de Granada (UGR)
83 pag.

SO (UGR)	
PRACTICAS RESUELTAS -SISTEMAS OPERATIVOS	
SANCHEZ BUENDIA, MARIA ANGUSTIAS	**16-17**

Universidad de Granada

PORTAFOLIOS SISTEMAS OPERATIVOS

Jesús García Godoy
2º B GII

Indice

Sesion 1

EJERCICIO 3.1

Crea un script de bash que automatice todos los pasos vistos en este punto y que guardarás preferiblemente en tu directorio home. Al entrar de nuevo en el sistema sólo tendrás que ejecutar el script para empezar a trabajar en modo root.

```
#!/bin/bash
#Autor: Jesús García Godoy
#Descripción: Práctica 1 Sistemas Operativos Ejercicio 3.1 Script
#Nombre de archivo: modo_root

cp /fenix/depar/lsi/UML/*.gz /tmp

cd /tmp

gunzip *.gz

./kernel32-3.0.4 ubda=./Fedora14-x86-root_fs mem=1024m
```

EJERCICIO 4.1

Visualiza el contenido de los dos archivos anteriores y comprueba cuales son las opciones por defecto que tendría un usuario que se creara en nuestro sistema. A continuación, crea una cuenta de usuario y visualiza el contenido de los archivos */etc/passwd* y */etc/group*, y el directorio /home para comprobar que los nuevos datos se han rellenado conforme a la especificación tomada de*/etc/default/useradd* y */etc/login.defs*.

```
$ cat /etc/default/useradd

# Default values for useradd(8)
#
# The SHELL variable specifies the default login shell on your
# system.
# Similar to DHSELL in adduser. However, we use "sh" here because
# useradd is a low level utility and should be as general
# as possible
SHELL=/bin/bash
#
```

```
# The default group for users
# 100=users on Debian systems
# Same as USERS_GID in adduser
# This argument is used when the -n flag is specified.
# The default behavior (when -n and -g are not specified) is to create a
# primary user group with the same name as the user being added to the
# system.
# GROUP=100
#
# The default home directory. Same as DHOME for adduser
# HOME=/home
#
# The number of days after a password expires until the account
# is permanently disabled
# INACTIVE=-1
#
# The default expire date
# EXPIRE=
#
# The SKEL variable specifies the directory containing "skeletal" user
# files; in other words, files such as a sample .profile that will be
# copied to the new user's home directory when it is created.
# SKEL=/etc/skel
#
# Defines whether the mail spool should be created while
# creating the account
# CREATE_MAIL_SPOOL=yes
```

```
$ cat /etc/login.defs

# /etc/login.defs - Configuration control definitions for the login package.
#
# Three items must be defined:  MAIL_DIR, ENV_SUPATH, and ENV_PATH.
# If unspecified, some arbitrary (and possibly incorrect) value will
# be assumed.  All other items are optional - if not specified then
# the described action or option will be inhibited.
#
# Comment lines (lines beginning with "#") and blank lines are ignored.
#
# Modified for Linux.  --marekm

# REQUIRED for useradd/userdel/usermod
#   Directory where mailboxes reside, _or_ name of file, relative to the
#   home directory.  If you _do_ define MAIL_DIR and MAIL_FILE,
#   MAIL_DIR takes precedence.
#
#   Essentially:
#      - MAIL_DIR defines the location of users mail spool files
#        (for mbox use) by appending the username to MAIL_DIR as defined
#        below.
#      - MAIL_FILE defines the location of the users mail spool files as the
#        fully-qualified filename obtained by prepending the user home
#        directory before $MAIL_FILE
```

```
#
# NOTE: This is no more used for setting up users MAIL environment variable
#       which is, starting from shadow 4.0.12-1 in Debian, entirely the
#       job of the pam_mail PAM modules
#       See default PAM configuration files provided for
#       login, su, etc.
#
# This is a temporary situation: setting these variables will soon
# move to /etc/default/useradd and the variables will then be
# no more supported
MAIL_DIR        /var/mail
#MAIL_FILE      .mail

#
# Enable logging and display of /var/log/faillog login failure info.
# This option conflicts with the pam_tally PAM module.
#
FAILLOG_ENAB       yes

#
# Enable display of unknown usernames when login failures are recorded.
#
# WARNING: Unknown usernames may become world readable.
# See #290803 and #298773 for details about how this could become a security
# concern
LOG_UNKFAIL_ENAB    no

#
# Enable logging of successful logins
#
LOG_OK_LOGINS             no

#
# Enable "syslog" logging of su activity - in addition to sulog file logging.
# SYSLOG_SG_ENAB does the same for newgrp and sg.
#
SYSLOG_SU_ENAB            yes
SYSLOG_SG_ENAB            yes

#
# If defined, all su activity is logged to this file.
#
#SULOG_FILE  /var/log/sulog

#
# If defined, file which maps tty line to TERM environment parameter.
# Each line of the file is in a format something like "vt100  tty01".
#
#TTYTYPE_FILE      /etc/ttytype

#
# If defined, login failures will be logged here in a utmp format
# last, when invoked as lastb, will read /var/log/btmp, so...
#
FTMP_FILE    /var/log/btmp
```

```
#
# If defined, the command name to display when running "su -".  For
# example, if this is defined as "su" then a "ps" will display the
# command is "-su".  If not defined, then "ps" would display the
# name of the shell actually being run, e.g. something like "-sh".
#
SU_NAME            su

#
# If defined, file which inhibits all the usual chatter during the login
# sequence.  If a full pathname, then hushed mode will be enabled if the
# user's name or shell are found in the file.  If not a full pathname, then
# hushed mode will be enabled if the file exists in the user's home directory.
#
HUSHLOGIN_FILE     .hushlogin
#HUSHLOGIN_FILE    /etc/hushlogins

#
# *REQUIRED*  The default PATH settings, for superuser and normal users.
#
# (they are minimal, add the rest in the shell startup files)
ENV_SUPATH   PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
ENV_PATH     PATH=/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games

[…]
```

```
$ useradd prueba
```

```
$ cat /etc/passwd

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
[…]
prueba:x:1001:1002::/home/prueba:/bin/bash
```

```
$ cat /etc/group

root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:superjes
[…]
prueba:x:1002:
```

EJERCICIO 4.2

1. Utiliza el manual en línea para leer la sintaxis completa de la utilidad para creación de cuentas y crea dos o tres usuarios en tu sistema cambiando alguno de los valores por defecto.

```
$ useradd -d /home/diferente –m -s /bin/sh -g 1002 otro
$ useradd -d /home/pato –m -s /bin/bash pato
```

- Otra manera de hacerlo:

```
$ gedit nuevos_usuarios

      otro:x:1002:1002:otro::/home/diferente:/bin/sh
      pato:x:1003:1003:pato::/home/pato:/bin/bash

$ newusers nuevos_usuarios
```

2. Elimina alguno de ellos y comprueba que "rastro" ha dejado la cuenta recién eliminada en el sistema.

```
$ userdel pato
```

-Ha dejado su carpeta home personal -> **/home/pato**

3. Entra (orden su) en el sistema como uno de estos usuarios que has creado y mira qué archivos tiene en su directorio home.

```
$ su pato
$ ls /home/pato
```

- Solo tiene un archivo -> examples.desktop. Tenemos que ver los archivos ocultos:

```
$ ls –a /home/pato
.  ..  .bash_logout  .bashrc  examples.desktop  .profile
```

- Vemos que tiene los archivos de configuración **.bash_logout, .bashrc** y **.profile.**

EJERCICIO 4.3

Visualiza el archivo */etc/passwd* e indica cual es el formato de cada línea de dicho archivo. Para ello también puedes consultar el man o info de Linux. ¿Quién es el propietario de éste archivo y cuáles son sus permisos?

```
$ cat /etc/passwd

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
pato:x:1003:1003::/home/pato:/bin/bash
superjes:x:1000:1000:SuperJes:/home/superjes:/bin/bash
```

```
[…]
```

El formato de línea es :

- **superjes**: Nombre de la cuenta (Login)
- **x**: Clave de acceso encriptada (password)
- **1000**: UID de esta cuenta
- **1000**: GID del grupo principal al que pertenece la cuenta
- **SuperJes**: Nombre del usuario
- **/home/superjes**: Directorio de trabajo del usuario
- **/bin/bash**: Interprete de comando (shell) de usuario

```
$ ls –lF /etc/|grep passwd

-rw-r--r--  1 root root      1795 2012-10-10 20:31 passwd
```

- El propietario es **root** y tiene permisos de **lectura y escritura** para el propio usuario(root), **lectura** para el grupo al que pertenece y **lectura** para el resto de usuarios.

EJERCICIO 4.4

Visualiza el archivo */etc/shadow* desde un usuario distinto al root ¿Te da algún problema? ¿Sabes por qué? Intenta averiguarlo.

```
$ cat /etc/shadow
```

- No se puede desde otro usuario dado que ese archivo tiene los siguientes permisos:

```
$ ls -lF /etc/shadow

-rw-r----- 1 root shadow 1238 2012-10-11 11:18 /etc/shadow
```

-Tiene solamente permisos de lectura y escritura para **root**, y permiso de lectura para el grupo al que pertenece **root.** Para los demás no tiene ninguno, así que no se puede hacer nada.

EJERCICIO 4.5

1. Crea un par de grupos y asignáselos a algunos de los usuarios de tu sistema.

```
$ groupadd grupo1

$ gpasswd -a pato grupo1
Añadiendo al usuario pato al grupo grupo1

$ groupadd grupo2

$ gpasswd -a otro grupo2
Añadiendo al usuario otro al grupo grupo2
```

2. ¿Qué información devuelve la orden id si estás conectado como root?

```
$ id
```

```
uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm),6(disk),10(wheel)
```

EJERCICIO 5.1

Utilizando la utilidad que ya conoces para la búsqueda de archivos en el sistema de archivos, anota el nombre absoluto del archivo del kernel de Linux que se ha cargado en el sistema operativo que estás usando en el laboratorio de prácticas para acceso modo root.

```
$ whereis Fedora
```

ó

```
$ find / -name Fedora

#-> /temp/Fedora14-x86-root_fs
```

EJERCICIO 5.2

Utilizando la información que tienes disponible responde a las siguientes preguntas:

(a) ¿Dónde podría guardar un programa que se ejecutase en modo root la información temporal de forma que ésta se mantuviese entre arranques del sistema?

- El directorio donde podríamos guardar información y que no se borre entre arranques del sistema es el directorio **/var/tmp.**

(b) ¿En que directorio se guardan los archivos de configuración del *Sistema de Ventanas X*? ¿Cuál es el nombre del archivo que almacena el nombre del programa *Gestor de Ventanas* que utiliza el sistema por defecto? ¿Cuál es el gestor de ventanas que usa por defecto nuestro sistema?

- Los archivos de configuración del Sistema de Ventanas X se encuentra en el directorio **/etc** , concretamente en **/etc/X11**. El nombre del archivo que almacena el nombre del programa Gestor de Ventanas que utiliza el sistema por defecto es /**etc**/**X11**/**default-display-manager**, el cual te indica que el gestor por defecto es /**usr**/**sbin**/**gdm.**

EJERCICIO 5.3

Los archivos /etc/fstab y /etc/mtab muestran información sobre los sistemas de archivos que se encuentran montados en el sistema. ¿Cuál es la diferencia entre la información que muestra cada uno de ellos?

- El archivo **/etc/fstab** se encuentra comúnmente en sistemas Unix como parte de la configuración del sistema. Lo más destacado de este fichero es la lista de discos y particiones disponibles. En ella se indican cómo montar cada dispositivo y qué configuración utilizar.

- El archivo **/etc/mtab** es un archivo de información del sistema, comúnmente en Unix. Este archivo lista todos actualmente los sistemas de ficheros montados junto con sus opciones de inicialización. mtab tiene mucho en común con fstab, pero la diferencia principal es que este último enumera todos los sistemas de archivos disponibles.

EJERCICIO 5.4

Edita el archivo /**etc**/**fstab** del sistema de archivos que estás utilizando en modo root y anota y describe la información que tiene registrada. Si no conoces alguna opción puedes consultar el manual en línea: **man fstab**.

```
#
# /etc/fstab
#
LABEL=ROOT                              /           auto   noatime1
1
tmpfs                                   /dev/shm    tmpfs  defaults
0 0
tmp                                     /tmp        tmpfs rw,mode=
      1777,fscontext=system_u:object_r:tmp_t:s0    0 0
devpts                                  /dev/pts    devpts gid=5,mode=620     0 0
sysfs                                   /sys        sysfs  defaults0 0
proc                                    /proc       proc   defaults0 0
```

- El primer campo nos indica el archivo de sistema de dispositivo especial remoto a ser montado, que en este caso es LABEL=ROOT.
- El segundo campo nos va a indicar el punto de montaje para el archivo de sistema.
- El tercer campo describe el tipo de archivo del sistema.
- El cuarto nos indica la cantidad de opciones asociados al archivo del sistema.
- El quinto campo es usado por los archivos de sistema por el comando de volcado.
- El sexto campo es usado por el fsck para determinar el orden en que los archivos del sistema son analizados como correctos a la hora de reiniciar.

EJERCICIO 5.5

Compara la información que presentan los cuatro archivos de texto que se han presentado en este apartado y describe en un párrafo para lo que te sirve la información que registra cada archivo.

- El archivo **/etc/fstab** se encuentra comúnmente en sistemas Unix como parte de la configuración del sistema. Lo más destacado de este fichero es la lista de discos y

particiones disponibles. En ella se indican cómo montar cada dispositivo y qué configuración utilizar.

- El archivo **/etc/mtab** es un archivo de información del sistema, comúnmente en Unix. Este archivo lista todos actualmente los sistemas de ficheros montados junto con sus opciones de inicialización. mtab tiene mucho en común con fstab, pero la diferencia principal es que este último enumera todos los sistemas de archivos disponibles.

- El archivo **/proc/filesystems** muestra una lista de los tipos del sistema de archivos soportados actualmente por el kernel.
- El archivo **/proc/mounts** proporciona una lista de todos los montajes en uso por el sistema. La salida de datos que encontramos aquí se parece a /etc/mtab, excepto que /proc/mounts está más actualizada

Sesion 2

EJERCICIO 3.1

En esta actividad utilizaremos un dispositivo USB de tipo "pen drive" para establecer particiones. Vamos a crear una tabla de particiones en las que se van a definir dos particiones primarias que se configuraran para albergar dos sistemas de archivos tipo linux, de manera que la primera albergará un SA ext3 y la segunda un ext4. Ambas serán particiones tipo Linux 0x83. El tamaño de las particiones queda a vuestra libre elección pero por lo menos deberían tener 512 MB.

```
$ fdisk /dev/loop0
```

- El proceso de creación de las particiones es interactivo, pulsando **p** se listan todas las particiones creadas, con **n** creamos una nueva y a continuación nos da a elegir si ha de ser primaria o extendida, y el primer y último cilindro de la misma. Para salir pulsamos **w**. Para que tenga 512 MB tenemos que poner *+512M* en el último cilindro. Hemos de tener en cuenta si la partición que creamos es la segunda de empezar al final de la primera.

EJERCICIO 4.1

El objetivo es simplemente formatear lógicamente las particiones creadas con anterioridad de forma consistente con el tipo de SA que se estableció que iba a ser alojado, e.d. En la primera partición crearemos un SA de tipo ext3 y en la segunda un ext4.

La orden que permite establecer un SA de los reconocidos dentro del sistema Linux sobre una partición de disco es mkfs (consulta el manual en línea para familiarizarte con sus opciones). El resultado de la ejecución de esta orden es el formateo lógico de la partición escogida utilizando el SA que se ha seleccionado.

Utiliza el manual en línea para conocer como ejecutar la orden de creación de SA. mkfs es la orden genérica para creación de sistemas de archivos. Como requisito es necesario que establezcas dos etiquetas de volumen para los SAs: LABEL_ext3 para la primera partición y LABEL_ext4 para la segunda. Debería aparecer un listado en pantalla similar al siguiente.

```
# mkfs.ext3 –L ‘LABEL_ext3’ /dev/loop0

# mkfs.ext4 –L ‘LABEL_ext4’ /dev/loop1
```

EJERCICIO 5.1

Consultando el manual en línea para la orden tune2fs responde a las siguientes preguntas:

(a) ¿Cómo podrías conseguir que en el siguiente arranque del sistema se ejecutara automáticamente e2fsck sin que se haya alcanzado el máximo número de montajes?

- Con el modificador ***–C*** y un poniéndole un número más grande de veces que el sistema de archivos es montado que **max-mount-counts** .

(b) ¿Cómo podrías conseguir reservar para uso exclusivo de un usuario username un número de bloques del sistema de archivos?

```
$ tune2fs /dev/loop0 –r [numero] –u username
```

EJERCICIO 6.1

Utiliza el manual en línea para descubrir la forma de montar nuestros SAs de manera que cumplas los siguientes requisitos:

El SA etiquetado como LABEL_ext3 debe estar montado en el directorio /mnt/SA_ext3 y en modo de solo lectura.

```
$ mount /dev/loop0 –r /mnt/SA_ext3
```

Ó

```
$ mount /dev/loop0 –o ro /mnt/SA_ext3
```

El SA etiquetado como LABEL_ext4 debe estar montado en el directorio /mnt/LABEL_ext4 y debe tener sincronizada sus operaciones de E/S de modificación de directorios.

```
$ mount /dev/loop1 –o dirsync /mnt/LABEL_ext4
```

EJERCICIO 6.2

Escribe las dos líneas necesarios en el archivo /etc/fstab para que se monten automáticamente nuestros dos SA en el arranque del sistema con las mismos requisitos que se han pedido en la actividad 6.1.

```
/dev/loop0 /mnt/SA_ext3 ext3 auto,ro 0 0

/dev/loop1 /mnt/LABEL_ext4 ext4 auto,dirsync 0 0
```

- La opción **auto** se podría obviar, ya que por defecto se monta al arrancar el sistema todo lo que haya en el fichero **/etc/fstab**.

EJERCICIO 7.1

En esta actividad se van a presentar los pasos que necesitas llevar a cabo para establecer el sistema de cuotas de disco en Linux. El objetivo será activar el sistema de cuotas sobre el sistema de archivos tipo ext3 que has creado con anterioridad.

1. Editar el archivo /etc/fstab y activar el sistema de cuotas de usuario para el SA tipo ext3. Busca como se especifica esta opción en el manual en línea. Una ayuda para la búsqueda es que la realices sobre la orden mount y recuerdes que las opciones de montaje bienen especificadas en los apartados: FILESYSTEM INDEPENDENT MOUNT OPTIONS y FILESYSTEM SPECIFIC MOUNT OPTIONS.

```
/dev/loop0 /mnt/SA_ext3 ext3 auto,ro,usrquota 0 0
```

2. Montar de nuevo el SA en el espacio de nombres para que se active la opción previamente establecida. Usa la siguiente orden:

```
# mount -o remount /mnt/SA_ext3
```

3. Crear el archivo que permite llevar el control de cuotas de usuario para el SA. El nombre de este archivo es aquota.user. Para ello utiliza la siguiente orden:

```
# quotacheck -nm /mnt/SA_ext3
```

4. Ahora procedemos a activar el sistema de control de cuotas de usuario. Para ello ejecuta la orden:

```
# quotaon -a
```

5. Ahora solo falta activar la cuota para cada usuario del sistema mediante la siguiente orden. En este caso, implementa un script que automatice la tarea. Puede ser buena idea utilizar el archivo /etc/passwd.

```
#!/bin/bash

for [ linea in `cat /etc/passwd` ]
do
      usuario = `cut –d : -f1`
      setquota -u $usuario
done
```

6. Para finalizar estableceremos el periodo de gracia para el límite soft.

```
# setquota –a –t 7 7
```

EJERCICIO 7.2

Establece los límites de bloques y i-nodos para un par de usuarios del sistema UML sobre el que trabajas en el laboratorio.

```
# setquota –u pato –b nombre 7 10 7 10
```

EJERCICIO 8.1

Accede a los sitios web especializados que ofrecen software para sistemas operativos Linux y enumera las principales características de cada uno de ellos en base, por ejemplo, a si contiene software abierto y/o propietario, tipos de aplicaciones disponibles, tamaño del sitio en cuanto a la cantidad de software que mantiene, y otras características que considere interesantes.

- (Mirar en internet simplemente).

EJERCICIO 8.2

Encuentra los archivos de configuración de YUM y explore las distintas órdenes disponibles en YUM ejecutándolas. En concreto, lista todos los paquetes instalados y disponibles, elimina el paquete instalado que te indique el profesor de prácticas, y a continuación vuelve a instalar el mismo paquete haciendo uso de los paquetes que se encuentran disponibles en /fenix/depar/lsi/so/*paquetes*.

- Los archivos de configuración del "**yum**" se encuentran en el directorio **/etc/yum/yum.conf .**
- **Yum** es muy potente, y por tanto tiene muchísimas órdenes disponibles. Algunas de ellas son las siguientes:

```
# yum install "paquete"→ Instala la última versión del paquete indicado. Pide por confirmación.

# yum -y install "paquete" → Isntala la última versión del paquete sin pedir confirmación.

# yum -y install "paquete1 paquete2"→ Instala muchos paquetes a la vez. La opción -y es para que no pida confirmación.

# yum -y update → Actualiza todos los paquetes del sistema.

# yum check-update → Muestra una lista de paquetes que necesitan ser actualizados sin instarlos.

# yum info paquete → Descripción completa del paquete indicado.

# yum list→ Lista todos los paquetes disponibles para instalación, actualización o actualizados.
```

- Y así otras muchas mas opciones, todas ellas muy útiles. A continuación, vamos a escribir los comandos exactos que quiere que ejecutemos el ejercicio para sus correspondientes acciones:

```
# yum list installed → Lista todos los paquetes instalados en el sistema.

# yum remove "paquete" → Eliminar un paquete instalado.
```

```
# yum install "paquete" → Instalar un paquete.
```

EJERCICIO 8.3

En primer lugar deseamos mostrar cierta metainformación acerca de uno o más paquetes ya instalados. Para ello debes utilizar la orden rpm con las opciones adecuadas. Utiliza el manual en línea si no sabes ya las opciones que debes utilizar. A continuación vamos a realizar una serie de operaciones: instalar, desinstalar y actualizar sobre un paquete específico que indicará el profesor:

- Para mostrar cierta meta información acerca de uno o más paquetes yainstalados., debemos ejecutar el siguiente comando:

```
# rpm -qi <nombre-paquete-instalado>
```

- A continuación vamos a realizar una serie de operaciones: instalar, desinstalar y actualizar sobre un paquete específico que indicará el profesor (en este caso uno cualquiera):

Muestra la información general (nombre, versión, arquitectura, grupo, descripción, etc.) y lista los archivos que contiene un paquete ya instalado haciendo uso de la orden rpm y un único conjunto de opciones.

```
# rpm -qli <nombre-paquete-instalado>
```

Idem que el anterior pero mostrando únicamente los archivos de configuración que contieneel paquete.

```
# rpm -q -c -i  <nombre-paquete-instalado> // o "-c"?
```

Escribe una orden que muestre los paquetes requeridos por un paquete determinado que se encuentre instalado en el sistema. Escriba la orden que devuelva el mismo resultado pero para un paquete no instalado en el sistema.

```
# rpm -q --whatrequires <nombre-paquete-instalado>
```

El profesor de prácticas te indicará que instales un paquete que se encuentra accesible a través del directorio punto de montaje, tal como se ha indicado en la actividad anterior. Consiga que RPM muestre en pantalla la máxima información posible acerca del proceso de instalación de dicho paquete

```
# rpm -ivh   </fenix/depar/lsi/so/paquetes/"nombre del paquete">
//ya lleva el -v en -ivh para ver la información
```

Desinstala el paquete del punto anterior mostrando en pantalla también la máxima información posible acerca del propio proceso de eliminación del paquete.

```
# rpm -e --nodeps -v <nombre del paquete>
```

Por último, vuelve a realizar el paso 4 y actualiza el paquete instalado a una versión más reciente, localizando el paquete necesario en el directorio punto de montaje comentado anteriormente. El proceso de actualización debe mostrar en pantalla también la máxima información posible acerca del propio proceso.

```
# rpm -F -v </fenix/depar/lsi/so/paquetes/"nombre del paquete">
```

Sesion 3

EJERCICIO 3.1

Responde a las siguientes cuestiones y especifica, para cada una, la opción que has utilizado (para ello utiliza *man* y consulta las opciones de las ordenes anteriormente vistas:

```
# uptime
18:32:07 up 15 min, 3 users, load average: 0.20, 0.26, 0.16
# w
18:33:36 up 17 min, 3 users, load average: 0,04, 0,19, 0,15
USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT
patricia tty7 :0 18:16 ? 20.00s 0.22s x-session-manag
patricia pts/0 :0.0 18:19 11:40 0.30s 4.86s gnome-terminal
patricia pts/1 :0.0 18:22 0.00s 0.18s 0.00s w
```

a) ¿Cuánto tiempo lleva en marcha el sistema?

- **15 minutos**

b) ¿Cuántos usuarios hay trabajando?

- **3 usuarios**

c) ¿Cuál es la carga media del sistema en los últimos 15 minutos?

- **0.16** (según la primera orden *uptime)*

EJERCICIO 3.2

a) Crea un script o guión shell que realice un ciclo de un número variable de iteraciones donde lo único que haga sea un sleep de 1 segundo y a continuación incremente el valor de una variable. Cuando terminen las iteraciones escribirá en pantalla un mensaje indicando el valor actual de la variable. Este guión debe tener un argumento que es el número de iteraciones que va a realizar. Por ejemplo, si el script se llama prueba_procesos, ejecutaríamos:

```
# prueba_procesos 1000
el valor de la variable es 1000
```

prueba.sh:

```
#!/bin/bash

x=0

for i in `seq 1 $1`
do
	sleep 1
	x=`expr $x + 1`
done
```

```
echo “El valor de la variable es “ $x
```

b) Ejecuta el guión anterior varias veces en *background* (segundo plano) y comprueba su prioridad inicial. Cambia la prioridad de dos de ellos, a uno se la aumentas y a otro se la disminuyes, ¿cómo se comporta el sistema para estos procesos?

```
# ./prueba.sh 100 &
[1] 3397

# ./prueba.sh 150 &
[2] 3404

# top

top - 01:22:17 up  4:15,  3 users,  load average: 0.41, 0.19, 0.11
Tasks: 152 total,   1 running, 150 sleeping,   0 stopped,   1 zombie
Cpu(s):  1.0%us, 24.5%sy,  0.0%ni, 73.2%id,  1.3%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3024920k total,   698740k used,  2326180k free,   101404k buffers
Swap:  1046524k total,        0k used,  1046524k free,   400844k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1190 root      20   0 55144  33m 8720 S  7.6  1.1   2:49.05 Xorg
 1649 superjes  20   0 94216  15m  11m S  4.9  0.5   1:10.90 gnome-terminal
 3307 root      20   0  2636 1144  860 R  1.3  0.0   0:01.84 top
 3404 root      20   0  6164 1404 1076 S  1.0  0.0   0:00.47 prueba.sh
 3397 root      20   0  6160 1396 1076 S  0.7  0.0   0:00.48 prueba.sh
```

- La prioridad inicial es de 20 para ambos.

```
# renice -5 4305
4305: prioridad antigua 0, nueva prioridad 5

# renice 25 4311
4311: prioridad antigua 0, nueva prioridad 19

# top

top - 01:22:17 up  4:15,  3 users,  load average: 0.41, 0.19, 0.11
Tasks: 152 total,   1 running, 150 sleeping,   0 stopped,   1 zombie
Cpu(s):  1.0%us, 24.5%sy,  0.0%ni, 73.2%id,  1.3%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3024920k total,   698740k used,  2326180k free,   101404k buffers
Swap:  1046524k total,        0k used,  1046524k free,   400844k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1649 superjes  20   0 94216  15m  11m S  3.6  0.5   1:15.07 gnome-terminal
 3307 root      20   0  2636 1144  860 R  1.0  0.0   0:07.56 top
 4305 superjes  25  -5  6160 1400 1076 S  1.0  0.0   0:00.41 prueba.sh
 4311 superjes  39  19  6164 1400 1076 S  0.7  0.0   0:00.40 prueba.sh
```

- Como podemos ver con la orden *top,* en la columna **NI** se puede ver el añadido o resta de prioridad que hemos hecho en tiempo real, con lo cual podemos ver que se le asigna más CPU en la columna **CPU.**

c) Escribe los tiempos de respuesta de la ejecución de uno de los guiones del apartado anterior.

```
# time ./prueba.sh 10
El valor de la variable es  10

real   0m10.822s
user   0m0.428s
sys    0m17.301s
```

EJERCICIO 3.3

a) La orden *pstree* muestra el árbol de procesos que hay en ejecución. Comprueba que la jerarquía mostrada es correcta haciendo uso de la orden *ps* y de los valores “PID" y “PPID" de cada proceso.

```
# pstree
init─┬─NetworkManager─┬─dhclient
     │                └─2*[{NetworkManager}]
     ├─4*[VBoxClient───{VBoxClient}]
     ├─VBoxClient───2*[{VBoxClient}]
     ├─VBoxService───7*[{VBoxService}]
     ├─acpid
     ├─atd
     ├─avahi-daemon───avahi-daemon
     ├─bamfdaemon
     ├─bonobo-activati───2*[{bonobo-activat}]
     ├─console-kit-dae───64*[{console-kit-da}]
     ├─cron
     ├─cupsd
     ├─3*[dbus-daemon]
     ├─2*[dbus-launch]
     ├─dconf-service───{dconf-service}
     ├─gconfd-2
     ├─gdm-binary─┬─gdm-simple-slav─┬─Xorg
     │            │                 ├─gdm-session-wor─┬─gnome-session─┬─bluetoo+
     │            │                 │                 │               ├─gdu-not+
     │            │                 │                 │               ├─gnome-p+
     │            │                 │                 │               ├─gnome-p+
     │            │                 │                 │               ├─metacit+
     │            │                 │                 │               ├─nautilu+
     │            │                 │                 │               ├─nm-appl+
     │            │                 │                 │               ├─polkit-+
     │            │                 │                 │               ├─ssh-age+
     │            │                 │                 │               ├─zeitgei+
     │            │                 │                 │               └─2*[{gno+
     │            │                 │                 └─{gdm-session-wo}
     │            │                 └─{gdm-simple-sla}
     │            └─{gdm-binary}
     ├─geoclue-master───{geoclue-master}
     ├─6*[getty]
     ├─gnome-keyring-d───4*[{gnome-keyring-}]
     ├─gnome-screensav
     ├─gnome-settings-───{gnome-settings}
     ├─gnome-terminal─┬─bash───sudo───su───bash───su───bash───sudo───su───bash─+++
     │                ├─bash───sudo───su───bash───su───bash───sudo───su───bash
     │                ├─gnome-pty-helpe
     │                └─2*[{gnome-terminal}]
     ├─gvfs-afc-volume───{gvfs-afc-volum}
     ├─gvfs-fuse-daemo───3*[{gvfs-fuse-daem}]
     ├─gvfs-gdu-volume
```

```
      ├─gvfs-gphoto2-vo
      ├─gvfsd
      ├─gvfsd-burn
      ├─gvfsd-metadata
      ├─gvfsd-trash
      ├─indicator-apple───2*[{indicator-appl}]
      ├─indicator-appli───{indicator-appl}
      ├─indicator-datet───2*[{indicator-date}]
      ├─indicator-me-se───{indicator-me-s}
      ├─indicator-messa───{indicator-mess}
      ├─indicator-sessi───{indicator-sess}
      ├─indicator-sound───2*[{indicator-soun}]
      ├─modem-manager
      ├─notification-ar───{notification-a}
      ├─notify-osd───{notify-osd}
      ├─polkitd───{polkitd}
      ├─pulseaudio─┬─gconf-helper───{gconf-helper}
      │            └─2*[{pulseaudio}]
      ├─rsyslogd───2*[{rsyslogd}]
      ├─rtkit-daemon───2*[{rtkit-daemon}]
      ├─trashapplet───{trashapplet}
      ├─udevd───2*[udevd]
      ├─udisks-daemon─┬─udisks-daemon
      │               └─{udisks-daemon}
      ├─upowerd───{upowerd}
      ├─upstart-socket-
      ├─upstart-udev-br
      ├─wnck-applet───{wnck-applet}
      ├─wpa_supplicant
      └─zeitgeist-daemo─┬─cat
                        ├─zeitgeist-datah
                        └─{zeitgeist-daem}
```

```
# ps -Al
F S   UID   PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
4 S     0     1     0  0  80   0 -   764 poll_s ?        00:00:02 init
1 S     0     2     0  0  80   0 -     0 kthrea ?        00:00:00 kthreadd
1 S     0    19     2  0  60 -20 -     0 rescue ?        00:00:00 md
1 S     0    22     2  0  80   0 -     0 watchd ?        00:00:00 khungtaskd
1 S     0    23     2  0  80   0 -     0 kswapd ?        00:00:00 kswapd0
1 S     0    24     2  0  85   5 -     0 ksm_sc ?        00:00:00 ksmd
1 S     0    25     2  0  80   0 -     0 fsnoti ?        00:00:00 fsnotify_mark
1 S     0    26     2  0  60 -20 -     0 rescue ?        00:00:00 aio
1 S     0    27     2  0  80   0 -     0 ecrypt ?        00:00:00 ecryptfs-kthre
1 S     0    28     2  0  60 -20 -     0 rescue ?        00:00:00 crypto
1 S     0    32     2  0  60 -20 -     0 rescue ?        00:00:00 kthrotld
1 S     0    34     2  0  80   0 -     0 scsi_e ?        00:00:00 scsi_eh_0
1 S     0    35     2  0  80   0 -     0 scsi_e ?        00:00:07 scsi_eh_1
5 S     0    36     2  0  80   0 -     0 worker ?        00:00:00 kworker/u:3
1 S     0    39     2  0  60 -20 -     0 rescue ?        00:00:00 kmpathd
1 S     0    40     2  0  60 -20 -     0 rescue ?        00:00:00 kmpath_handler
1 S     0    41     2  0  60 -20 -     0 rescue ?        00:00:00 kondemand
1 S     0    42     2  0  60 -20 -     0 rescue ?        00:00:00 kconservative
1 S     0   159     2  0  80   0 -     0 scsi_e ?        00:00:00 scsi_eh_2
1 S     0   175     2  0  80   0 -     0 kjourn ?        00:00:01 jbd2/sda1-8
1 S     0   176     2  0  60 -20 -     0 rescue ?        00:00:00 ext4-dio-unwri
```

```
1 S     0   193     2  0  80   0 -     0 bdi_wr ?        00:00:00 flush-8:0
1 S     0   226     1  0  80   0 -   604 poll_s ?        00:00:00 upstart-udev-b
5 S     0   235     1  0  76  -4 -   721 poll_s ?        00:00:00 udevd
5 S   101   320     1  0  80   0 -  6869 poll_s ?        00:00:00 rsyslogd
5 S   102   332     1  0  80   0 -   875 poll_s ?        00:00:03 dbus-daemon
5 S   104   357     1  0  80   0 -   755 poll_s ?        00:00:01 avahi-daemon
1 S   104   360   357  0  80   0 -   755 unix_s ?        00:00:00 avahi-daemon
5 S     0   364     1  0  80   0 -  6467 poll_s ?        00:00:01 NetworkManager
4 S     0   378     1  0  80   0 -  1166 poll_s ?        00:00:00 modem-manager
4 S     0   382     1  0  80   0 -  5941 poll_s ?        00:00:02 polkitd
1 S     0   417     2  0  60 -20 -     0 rescue ?        00:00:00 kpsmoused
4 S     0   432     1  0  80   0 -  1294 poll_s ?        00:00:00 wpa_supplicant
1 S     0   658     1  0  80   0 -   604 poll_s ?        00:00:00 upstart-socket
0 S     0   786     1  0  80   0 -   469 n_tty_ tty4     00:00:00 getty
0 S     0   791     1  0  80   0 -   469 n_tty_ tty5     00:00:00 getty
0 S     0   802     1  0  80   0 -   469 n_tty_ tty2     00:00:00 getty
0 S     0   803     1  0  80   0 -   469 n_tty_ tty3     00:00:00 getty
0 S     0   806     1  0  80   0 -   469 n_tty_ tty6     00:00:00 getty
1 S     0   815     1  0  80   0 -   467 poll_s ?        00:00:00 acpid
1 S     1   816     1  0  80   0 -   534 hrtime ?        00:00:00 atd
1 S     0   817     1  0  80   0 -   568 hrtime ?        00:00:00 cron
1 S     0   982     1  0  80   0 -  1751 rt_sig ?        00:00:18 VBoxService
4 S     0  1074     1  0  80   0 -  4940 poll_s ?        00:00:02 gdm-binary
4 S     0  1081     1  0  80   0 -  1699 ep_pol ?        00:00:00 cupsd
4 S     0  1087     1  0  80   0 -  6543 poll_s ?        00:00:01 console-kit-da
4 S     0  1162  1074  0  80   0 -  5283 poll_s ?        00:00:00 gdm-simple-sla
4 R     0  1190  1162  1  80   0 - 10999 -      tty7     00:03:47 Xorg
4 S     0  1231  1162  0  80   0 -  4870 poll_s ?        00:00:00 gdm-session-wo
4 S  1000  1257  1231  0  80   0 -  9285 poll_s ?        00:00:00 gnome-session
0 S     0  1258     1  0  80   0 -   469 n_tty_ tty1     00:00:00 getty
1 S  1000  1314     1  0  80   0 -  2215 rtR0Se ?        00:00:00 VBoxClient
1 S  1000  1321     1  0  80   0 -  2404 poll_s ?        00:00:01 VBoxClient
1 S  1000  1324  1257  0  80   0 -   843 poll_s ?        00:00:00 ssh-agent
1 S  1000  1329     1  0  80   0 -   865 poll_s ?        00:00:00 dbus-launch
1 S  1000  1330     1  0  80   0 -  1134 poll_s ?        00:00:03 dbus-daemon
0 S  1000  1335     1  0  80   0 -  2821 poll_s ?        00:00:00 gconfd-2
1 S  1000  1348     1  0  80   0 - 10926 poll_s ?        00:00:00 gnome-keyring-
1 S  1000  1352     1  0  80   0 - 25568 poll_s ?        00:00:12 gnome-settings
0 S  1000  1356     1  0  80   0 -  2260 poll_s ?        00:00:00 gvfsd
1 S  1000  1362     1  0  80   0 -  7772 futex_ ?        00:00:00 gvfs-fuse-daem
0 S  1000  1365  1257  0  80   0 - 27033 poll_s ?        00:00:18 metacity
1 S  1000  1367     1  0  69 -11 - 24227 poll_s ?        00:00:14 pulseaudio
4 S   110  1370     1  0  81   1 -  4723 poll_s ?        00:00:01 rtkit-daemon
0 S  1000  1374  1257  0  80   0 - 31211 poll_s ?        00:00:05 nautilus
0 S  1000  1382  1367  0  80   0 -  5198 poll_s ?        00:00:00 gconf-helper
0 S  1000  1384  1257  0  80   0 - 12017 poll_s ?        00:00:00 zeitgeist-data
0 S  1000  1387  1257  0  80   0 - 11720 poll_s ?        00:00:00 bluetooth-appl
0 S  1000  1389  1257  0  80   0 - 22849 poll_s ?        00:00:06 gnome-panel
0 S  1000  1390  1257  0  80   0 -  7049 poll_s ?        00:00:00 polkit-gnome-a
0 S  1000  1391  1257  0  80   0 - 32058 poll_s ?        00:00:02 nm-applet
0 S  1000  1398     1  0  80   0 -  8337 poll_s ?        00:00:01 zeitgeist-daem
0 S  1000  1400  1257  0  80   0 -  9605 poll_s ?        00:00:02 gnome-power-ma
0 S  1000  1409  1398  0  80   0 -  1266 unix_s ?        00:00:00 cat
0 Z  1000  1414  1398  0  80   0 -     0 exit   ?        00:00:00 zeit <defunct>
0 S  1000  1426     1  0  80   0 -  2635 poll_s ?        00:00:00 gvfs-gdu-volum
4 S     0  1429     1  0  80   0 -  3526 poll_s ?        00:00:00 udisks-daemon
```

```
1 S     0  1430  1429  0  80   0 -  1392 poll_s ?        00:00:07 udisks-daemon
0 S  1000  1435     1  0  80   0 -  2416 poll_s ?        00:00:00 gvfs-gphoto2-v
0 S  1000  1440     1  0  80   0 -  4839 poll_s ?        00:00:00 gvfs-afc-volum
0 S  1000  1447     1  0  80   0 -  2406 poll_s ?        00:00:00 gvfsd-trash
4 S     0  1463     1  0  80   0 -  4175 poll_s ?        00:00:00 upowerd
0 S  1000  1465  1257  0  80   0 -  5364 poll_s ?        00:00:00 gdu-notificati
0 S  1000  1478     1  0  80   0 - 18907 poll_s ?        00:00:10 notify-osd
1 S  1000  1493     1  0  80   0 -  7240 poll_s ?        00:00:00 gnome-screensa
0 S  1000  1514     1  0  80   0 -  2295 poll_s ?        00:00:00 gvfsd-burn
0 S  1000  1529     1  0  80   0 -  8921 poll_s ?        00:00:00 bonobo-activat
1 S  1000  1537     1  0  80   0 -  2210 rtR0Se ?        00:00:00 VBoxClient
0 S  1000  1545     1  0  80   0 - 20577 poll_s ?        00:00:11 wnck-applet
0 S  1000  1548     1  0  80   0 - 20304 poll_s ?        00:00:00 trashapplet
0 S  1000  1555     1  0  80   0 - 25235 poll_s ?        00:00:06 indicator-appl
0 S  1000  1559     1  0  80   0 - 19838 poll_s ?        00:00:00 notification-a
1 S  1000  1579     1  0  80   0 -  2220 rtR0Se ?        00:00:00 VBoxClient
1 S  1000  1589     1  0  80   0 -  2369 poll_s ?        00:00:03 VBoxClient
0 S  1000  1592     1  0  80   0 -  2216 poll_s ?        00:00:00 gvfsd-metadata
0 S  1000  1595     1  0  80   0 -  5343 poll_s ?        00:00:00 dconf-service
0 S  1000  1600     1  0  80   0 - 15369 poll_s ?        00:00:00 indicator-date
0 S  1000  1602     1  0  80   0 - 10125 poll_s ?        00:00:01 indicator-appl
0 S  1000  1607     1  0  80   0 - 30532 poll_s ?        00:00:00 indicator-soun
0 S  1000  1609     1  0  80   0 - 15358 poll_s ?        00:00:00 indicator-mess
0 S  1000  1611     1  0  80   0 - 13114 poll_s ?        00:00:00 indicator-sess
0 S  1000  1613     1  0  80   0 - 15618 poll_s ?        00:00:00 indicator-me-s
0 S  1000  1643     1  0  80   0 -  4519 poll_s ?        00:00:00 geoclue-master
0 S  1000  1649     1  0  80   0 - 23563 poll_s ?        00:01:36 gnome-terminal
0 S  1000  1651     1  0  80   0 -  5347 poll_s ?        00:00:06 bamfdaemon
0 S  1000  1655  1649  0  80   0 -   518 unix_s ?        00:00:00 gnome-pty-help
0 S  1000  1656  1649  0  80   0 -  2174 wait   pts/0    00:00:00 bash
4 S     0  1709  1656  0  80   0 -  1635 poll_s pts/0    00:00:00 sudo
4 S     0  1710  1709  0  80   0 -  1578 wait   pts/0    00:00:00 su
0 S     0  1718  1710  0  80   0 -  1630 wait   pts/0    00:00:00 bash
1 S     0  1735     2  0  60 -20 -     0 loop_t ?        00:00:00 loop0
1 S     0  1739     2  0  60 -20 -     0 loop_t ?        00:00:00 loop1
4 S  1000  2199  1718  0  80   0 -  1578 wait   pts/0    00:00:00 su
0 S  1000  2208  2199  0  80   0 -  2181 wait   pts/0    00:00:00 bash
4 S     0  2260  2208  0  80   0 -  1635 poll_s pts/0    00:00:00 sudo
4 S     0  2261  2260  0  80   0 -  1578 wait   pts/0    00:00:00 su
0 S     0  2269  2261  0  80   0 -  1669 wait   pts/0    00:00:04 bash
5 S     0  2410     1  0  80   0 -   865 poll_s pts/0    00:00:00 dbus-launch
5 S     0  2411     1  0  80   0 -   696 poll_s ?        00:00:00 dbus-daemon
1 S     0  2717     2  0  80   0 -     0 bdi_wr ?        00:00:00 flush-7:0
1 S     0  2830     2  0  80   0 -     0 kjourn ?        00:00:00 kjournald
5 S     0  2926   235  0  78  -2 -   720 poll_s ?        00:00:00 udevd
5 S     0  2927   235  0  78  -2 -   720 poll_s ?        00:00:00 udevd
4 S     0  2943   364  0  80   0 -   639 poll_s ?        00:00:00 dhclient
0 S  1000  3314  1649  0  80   0 -  2183 wait   pts/1    00:00:01 bash
4 S     0  3367  3314  0  80   0 -  1635 poll_s pts/1    00:00:00 sudo
4 S     0  3370  3367  0  80   0 -  1578 wait   pts/1    00:00:00 su
0 S     0  3378  3370  0  80   0 -  1704 wait   pts/1    00:00:00 bash
4 S  1000  4243  3378  0  80   0 -  1578 wait   pts/1    00:00:00 su
0 S  1000  4252  4243  0  80   0 -  2181 wait   pts/1    00:00:01 bash
4 S     0  4368  4252  0  80   0 -  1373 poll_s pts/1    00:00:00 sudo
4 S     0  4369  4368  0  80   0 -  1578 wait   pts/1    00:00:00 su
0 S     0  4381  4369  0  80   0 -  1664 n_tty_ pts/1    00:00:00 bash
```

```
1 S     0  5655     2  0  80   0 -     0 worker ?        00:00:00 kworker/0:0
1 S     0  5989     2  0  80   0 -     0 worker ?        00:00:00 kworker/0:2
1 S     0  6029     2  0  80   0 -     0 worker ?        00:00:00 kworker/0:1
1 S     0  6051     2  0  80   0 -     0 worker ?        00:00:00 kworker/0:3
4 R     0  6053  2269  0  80   0 -  1411 -      pts/0    00:00:00 ps
```

- Como podemos comprobar en los procesos marcados con **negrita** como ejemplo, podemos ver fácilmente que se sus **PPID** se corresponden con los **PID** de los procesos padres, tal y como muestra el árbol de la orden *pstree.*

b) Ejecuta la orden *ps* con la opción -A, ¿qué significa que un proceso tenga un carácter "?" en la columna etiquetada como TTY?

- Significa que no tiene asociado ningún terminal en concreto.

EJERCICIO 3.4

Responde a las siguientes cuestiones y especifica, para cada una, la orden que has utilizado:

a) ¿Qué porcentaje de tiempo de CPU se ha usado para atender interrupciones hardware?

```
# mpstat
Linux 2.6.38-8-generic (superjes-VirtualBox) 21/10/12     _i686_ (1 CPU)

02:12:22     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest
%idle
02:12:22     all    0,33    0,38    4,49    0,49    0,00    0,04    0,00    0,00
94,27
```

- Tal y como indica la columna **%irq**, correspondiente al porcentaje de CPU dedicado a interrupciones hardware, se ha dedicado un **0,00%.**

b) ¿Y qué porcentaje en tratar interrupciones software?

```
# mpstat
Linux 2.6.38-8-generic (superjes-VirtualBox) 21/10/12     _i686_ (1 CPU)

02:12:22     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest
%idle
02:12:22     all    0,33    0,38    4,49    0,49    0,00    0,04    0,00    0,00
94,27
```

- Tal y como indica la columna **%soft**, correspondiente al porcentaje de CPU dedicado a interrupciones software, se ha dedicado un **0,04%.**

c) ¿Cuánto espacio de *swap* está libre y cuánto ocupado?

```
# top

top - 02:18:10 up  5:11,  3 users,  load average: 0.00, 0.05, 0.10
Tasks: 153 total,   1 running, 151 sleeping,   0 stopped,   1 zombie
Cpu(s):  0.3%us,  4.5%sy,  0.4%ni, 94.3%id,  0.5%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3024920k total,   712296k used,  2312624k free,   103564k buffers
Swap:  1046524k total,        0k used,  1046524k free,   407128k cached
```

```
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 6387 root      20   0  2632 1132  848 R  8.6  0.0   0:00.18 top
    1 root      20   0  3056 1836 1272 S  0.0  0.1   0:02.37 init
    2 root      20   0     0    0    0 S  0.0  0.0   0:00.02 kthreadd
    […]
```

- Como podemos ver con la orden *top*, el total de memoria SWAP es de **1046524k**, de los cuales en uso hay **0k**, así que toda está libre: **1046524k.**

EJERCICIO 4.1

Explore las opciones de las que consta la orden **free** prestando especial atención a las diferentes unidades de medida según las que puede informar acerca de memoria. Adicionalmente, compare entre la posibilidad que tiene la propia orden para monitorizar periódicamente la información acerca de la memoria o haciendo esto mismo con ayuda de la orden **watch**.

- Las diferentes opciones del comando **free** son las siguientes:

```
OPTIONS
       -b     Display the amount of memory in bytes.

       -c count
              Display the result count times.  Requires the -s option.

       -g     Display the amount of memory in gigabytes.

       -k     Display the amount of memory in kilobytes. This is the default.

       -l     Show detailed low and high memory statistics.

       -m     Display the amount of memory in megabytes.

       -o     Display the output in old format, the only difference being this
              option will disable the display of the "buffer adjusted" line.

       -s     Continuously display the result delay  seconds  apart.  You  may
              actually  specify any floating point number for delay, usleep(3)
              is used for microsecond resolution delay times.

       -t     Display a line showing the column totals.

       -V     Display version information.
```

-Por lo tanto, si queremos ver cada x segundos el resultado de la orden **free** (en este ejemplo cada 4 segundos) debemos poner:

```
# free –s 4
```

Ó

```
# watch –n 4 free
```

EJERCICIO 4.2

Intente reproducir el escenario justo descrito anteriormente supervisando la actividad del sistema mediante la ejecución periódica de **vmstat** tal cual se ha descrito, y proporcione como muestra la salida almacenada en un archivo de texto.

- Para ello he creado el siguiente script, donde creo el fichero de texto, lo leo y voy cogiendo cada dato de la columna correspondiente , para luego mostrarlo en formato texto para que resulte más práctico, el archivo se llama **script_4-2.sh**:

```
#!/bin/bash

vmstat >> monitorizacion.txt

num_procs=`cat "monitorizacion.txt"|tail -1|tr -s " " " "|cut -d " " -f2`
num_procs_swap=`cat "monitorizacion.txt"|tail -1|tr -s " " " "|cut -d " " -f17`
incremento_swap=`cat "monitorizacion.txt"|tail -1|tr -s " " " "|cut -d " " -f9`
mem_libre=`cat "monitorizacion.txt"|tail -1|tr -s " " " "|cut -d " " -f5`


echo "Hay " $num_procs " procesos en cola de ejecución"
echo "Hay " $num_procs_swap " procesos ejecutándose en el área de intercambio"

if [ $incremento_swap -eq 0 ]
then
      echo "No hay incremento de procesos pasándose a la memoria de intercambio"
else
      echo "Se ha incrementado el numero de procesos que van a la memoria de
intercambio en " $incremento_swap
fi

echo "La memoria libre es de " $mem_libre "KB"
```

- Para realizar una revisión periódica(en este ejemplo que se actualice cada 4 segundos) , podemos ejecutar en un terminal el siguiente comando:

```
# watch -n 4 ./script_4-2.sh
```

- Lo que nos da la siguiente salida:

```
Every 4,0s: ./script_4-2.sh                              Sun Oct 21 12:41:39 2012

Hay  0  procesos en cola de ejecución
Hay  0  procesos ejecutándose en el área de intercambio
No hay incremento de procesos pasándose a la memoria de intercambio
La memoria libre es de  2298828 KB
```

EJERCICIO 5.1

Anota al menos dos nombres de archivo de dispositivo de bloques y dos nombres de dispositivo de caracteres de tu sistema UML. Anota los nombres de los archivos ocultos de tu directorio de inicio como usuario root que tienen relación con el intérprete de órdenes que tienes asignado por defecto. Ahora efectúa la misma tarea pero en una consola de terminal del sistema Ubuntu 8.10 que arrancas inicialmente en el laboratorio de prácticas. ¿Coinciden los nombres anotados en modo root y en modo usuario de laboratorios de informática?

- Como **root:**

```
# ls -a /root
.             archivo_SA30   .config   .gconfd   .pulse
..            .bash_history  .dbus     .gnome2   .pulse-cookie
archivo_SA20  .bashrc        .gconf    .profile  .synaptic
```

- Como **usuario**:

```
superjes@superjes-VirtualBox:~/SO/Practica_1$ ls -a $HOME
.                .esd_auth         .netbeans-registration
..               .fontconfig       Plantillas
Archivos FS      .gconf            .profile
.bash_history    .gconfd           pruebagii
.bash_logout     .gksu.lock        pruebamake
.bashrc          .gnome2           Público
.cache           .gnome2_private   .pulse
.codeblocks      .gtk-bookmarks    .pulse-cookie
.config          .gvfs             SCD
CopiasSeguridad  .ICEauthority     SO
.dbus            .icons            .sudo_as_admin_successful
.ddd             Imágenes          .themes
Descargas        .libreoffice      .thumbnails
.dmrc            .local            .vboxclient-display.pid
Documentos       .mission-control  .vboxclient-seamless.pid
.dropbox         modulos           Vídeos
Dropbox          .mozilla          .wxMaxima
.dropbox-dist    MP                .xchat2
EC               Música            .xsession-errors
ED               .nautilus         .xsession-errors.old
.emacs.d         .netbeans
Escritorio       NetBeansProjects
```

- Sí, coinciden pero el **usuario** además tiene el archivo **.bash_logout.**

EJERCICIO 5.2

Conocemos la sintaxis de la orden para obtener un listado en formato largo (*"long listing format"*). Manteniendo la opción de listado largo añade las opciones que sean necesarias para obtener un listado largo con las siguientes especificaciones:

(a) Que contenga el campo "*access time*" de los archivos del directorio especificado y que esté ordenado por dicho campo.

```
# ls –ltu $HOME
```

(b) Que contenga el campo "*ctime*" de los archivos del directorio especificado y que esté ordenado por dicho campo.

```
# ls –ltc $HOME
```

EJERCICIO 5.3

Comprueba cuantos bloques de datos está usando la partición raíz del sistema UML del laboratorio. Ahora obtén la misma información pero expresada en "human readable format": Megabytes o Gigabytes. Para ello consulta "inteligentemente" el manual en línea.

```
# du /usr
157728 /usr/bin
444    /usr/games
4      /usr/local/bin
4      /usr/local/games
```

```
4       /usr/local/src
4       /usr/local/share/fonts
4       /usr/local/share/ppd
{...}
2636116       /usr
```

- El UML de mi usuario está usando **2636116 bloques.**
- En formato "**human readable format**":

```
# du -h /usr

{…}

2,6G   /usr
```

EJERCICIO 5.4

¿Cuántos inodos se están usando en la partición raíz? ¿Cuántos nuevos archivos se podrían crear en esta partición?

```
# df -i /
S.ficheros            Inodos   IUsado   ILibre IUso% Montado en
/dev/sda1             595680  188386  407294   32% /
```

- Se están usando **188386** inodos en la partición raíz.
- Se podrían crear tantos archivos como inodos libres haya, en este caso **407294.**

EJERCICIO 5.5

¿Cuál es el tamaño del directorio /etc? ¿Y el del directorio /var? Compara estos tamaños con los de los directorios /bin, /usr y /lib. Anota brevemente tus conclusiones.

```
# du -h /etc

4,0K   /etc/ifplugd/action.d
8,0K   /etc/ifplugd
4,0K   /etc/openoffice
8,0K   /etc/kernel/header_postinst.d
12K    /etc/kernel/postrm.d
20K    /etc/kernel/postinst.d
44K    /etc/kernel
8,0K   /etc/profile.d
{…}
14M    /etc
```

- El tamaño del directorio **/etc** es de **14 MB.**

```
# du –h /var

{…}
299M   /var
```

- El tamaño del directorio **/var** es de **299 MB.**

```
# du -h /bin
```

```
6,6M   /bin
```

- El tamaño del directorio **/bin** es de **6,6 MB.**

```
# du –h /usr

2,6G   /usr
```

- El tamaño del directorio **/usr** es de **2,6 GB.**

```
#du –h /lib

152M   /lib
```

- El tamaño del directorio **/lib** es de **152 MB.**

- Como podemos ver, el directorio que más ocupa es **/usr**, dado que los programas están instalados ahí. El más pequeño es **/bin**, dado que sólo contiene ejecutables.

EJERCICIO 5.6

Obtén el número de bloques de tamaño 4 KB que utiliza la rama de la estructura jerárquica de directorios que comienza en el directorio /etc. En otras palabras, los bloques de tamaño 4 KB del subárbol cuya raíz es /etc. ¿Cuál es el tamaño de bloque, por omisión, utilizado en el SA?

```
# du -B 4 /etc

{…}
3632128      /etc
```

- El directorio **/etc** tiene **3632128 bloques** de tamaño 4 KB.
- El tamaño de bloque por omisión utilizado en el SA es de **4 KB**:

```
# tune2fs -l /dev/sda1 | grep -i "Block size"
Block size:               4096
```

EJERCICIO 5.7

Construye los mismos enlaces, duros y simbólico, que muestra la salida por pantalla anterior. Para ello crea los archivos archivo.txt y target_hardLink2.txt y, utilizando el manual en línea para ln, construye los enlaces softLink, hardLink y hardLink2. Anota las órdenes que has utilizado.

```
# ln -s archivo.txt ./softLink

# ln archivo.txt hardlink

# ln target_hardLink2.txt hardlink2

# mkdir D1/

# ls -lai
total 161788
 6853 drwxr-xr-x 4 superjes superjes      4096 2012-10-21 17:40 .
 6706 drwxr-xr-x 3 superjes superjes      4096 2012-10-08 12:58 ..
```

```
26010 -rw-r--r-- 2 root     root            0 2012-10-21 17:34 archivo.txt
35462 drwxr-xr-x 2 root     root         4096 2012-10-21 17:40 D1
26010 -rw-r--r-- 2 root     root            0 2012-10-21 17:34 hardlink
35461 -rw-r--r-- 2 root     root            0 2012-10-21 17:35 hardlink2
35460 lrwxrwxrwx 1 root     root           11 2012-10-21 17:37 softLink ->
archivo.txt
35461 -rw-r--r-- 2 root     root            0 2012-10-21 17:35
target_hardLink2.txt

# cd D1

# ls -lai
total 8
35462 drwxr-xr-x 2 root     root     4096 2012-10-21 17:40 .
 6853 drwxr-xr-x 4 superjes superjes 4096 2012-10-21 17:40 ..
```

¿Por qué el contador de enlaces del archivo archivo.txt vale 2 si sobre el existen un enlace duro hardLink y un enlace simbólico softLink?

- Porque los enlaces blandos no cuentan en el contador de enlaces, tan sólo los duros, que son realmente referencias reales al metadato, los blandos tan sólo al nombre del archivo.

EJERCICIO 5.8

A continuación se especifica el resultado que se pretende conseguir con la ejecución de la orden ls, junto con una especificación de opciones concreta que debes descubrir. Anota la orden completa que permite obtener los resultados especificados. Para resolver la actividad haz uso del manual en línea para la orden ls.

(a) Lista la información de metadatos de archivo para los archivos del directorio especificado. En el caso de que haya archivos de tipo enlace simbólico, la orden debe mostrar la información del archivo al que enlaza cada enlace simbólico y no la del propio archivo de tipo enlace simbólico).

```
# ls -laiL
total 161788
 6853 drwxr-xr-x 4 superjes superjes      4096 2012-10-21 17:45 .
 6706 drwxr-xr-x 3 superjes superjes      4096 2012-10-08 12:58 ..
26010 -rw-r--r-- 2 root     root            0 2012-10-21 17:34 archivo.txt
35462 drwxr-xr-x 2 root     root         4096 2012-10-21 17:40 D1
26010 -rw-r--r-- 2 root     root            0 2012-10-21 17:34 hardlink
35461 -rw-r--r-- 2 root     root            0 2012-10-21 17:35 hardlink2
34803 -rwxr-xr-x 1 root     root          766 2012-10-21 12:37 script_4-2.sh
26010 -rw-r--r-- 2 root     root            0 2012-10-21 17:34 softLink
35461 -rw-r--r-- 2 root     root            0 2012-10-21 17:35
target_hardLink2.txt
```

- Con el parámetro **–L** es con el que conseguimos lo que dice el enunciado.

(b) Lista la información de metadatos de archivo para los archivos del directorio especificado. En el caso de directorios y enlaces simbólicos debe mostrar la información del propio tipo de archivo. En el caso de directorios no debe mostrar el contenido de los directorios contenidos en el especificado, sino los metadatos de los archivos tipo directorio.

```
# ls -laid
6853 drwxr-xr-x 4 superjes superjes 4096 2012-10-21 17:45 .
```

- Con el parámetro **–d** es con el que conseguimos hacer lo que dice el enunciado en concreto.

EJERCICIO 5.9

Consulta el manual en línea para la orden mknod y crea un dispositivo de bloques y otro de caracteres. Anota las órdenes que has utilizado y la salida que proporciona un ls -li de los dos archivos de dispositivo recién creados. Puedes utilizar las salidas por pantalla mostradas en esta sección del guión para ver el aspecto que debe presentar la información de un archivo de dispositivo.

```
# mknod disp_bloques b 10 5

# mknod disp_caracteres c 10 5

# ls -li
total 161780
26010 -rw-r--r-- 2 root     root            0 2012-10-21 17:34 archivo.txt
35462 drwxr-xr-x 2 root     root         4096 2012-10-21 17:40 D1
35510 brw-r--r-- 1 root     root        10, 5 2012-10-22 12:14 disp_bloques
35572 crw-r--r-- 1 root     root        10, 5 2012-10-22 12:14 disp_caracteres
```

Sesion 4

EJERCICIO 3.1

A partir de la información proporcionada por la orden ***ps*** encuentre los datos asociados a los demonios ***atd*** y ***cron***, en concreto: quién es su padre, qué terminal tienen asociado y cuál es su usuario.

```
# ps -aux|grep atd

daemon     816  0.0  0.0   2136   344 ?        Ss   Oct28   0:00 atd
root     13800  0.0  0.0   5324   872 pts/0    S+   10:34   0:00 grep --color=auto
atd

# ps -p 816 -f
UID        PID  PPID  C STIME TTY          TIME CMD
daemon     816     1  0 Oct28 ?        00:00:00 atd
```

- Como podemos ver, **atd** no tiene asociado ningún terminal(**?**), su usuario es **daemon** y su padre es **1(init)**

```
# ps -aux|grep cron

root       817  0.0  0.0   2272   908 ?        Ss   Oct28   0:01 cron
root     13802  0.0  0.0   5324   872 pts/0    S+   10:34   0:00 grep --color=auto
cron

# ps -p 817 -f
UID        PID  PPID  C STIME TTY          TIME CMD
root       817     1  0 Oct28 ?        00:00:01 cron
```

- Como podemos ver, **cron** no tiene asociado ningún terminal(**?**), su usuario es **root** y su padre es **1(init).**

EJERCICIO 4.1

Cree un archivo **genera-apunte** que escriba la lista de hijos del directorio home en un archivo de nombre **listahome-`date +%Y-%j-%T-$$`,** es decir, la yuxtaposición del literal "listahome" y el año, día dentro del año, la hora actual y pid (consulte la ayuda de date).

```
#!/bin/bash

ls ~ > listahome-`date +%Y-%j-%T-$$`
```

Lance la ejecución del archivo **genera-apunte** un minuto más tarde de la hora actual.

```
# at –f ./genera-apunte.sh 11:03
warning: commands will be executed using /bin/sh
job 2 at Mon Oct 29 11:03:00 2012
```

¿En qué directorio se crea el archivo de salida?

- En la carpeta donde lo ejecutamos.

EJERCICIO 4.2

Lance varias órdenes at utilizando distintas formas de especificar el tiempo como las siguientes:

(será de utilidad la opción -v):

a) a medianoche de hoy

```
# at -f ./genera-apunte.sh midnight
warning: commands will be executed using /bin/sh
job 5 at Tue Oct 30 00:00:00 2012
```

b) un minuto después de la medianoche de hoy

```
# at -f ./genera-apunte.sh midnight+1 minute
warning: commands will be executed using /bin/sh
job 7 at Tue Oct 30 00:01:00 2012
```

c) a las 17 horas y 30 minutos de mañana

```
# at -f ./genera-apunte.sh 17:30 tomorrow
warning: commands will be executed using /bin/sh
job 8 at Tue Oct 30 17:30:00 2012
```

d) a la misma hora en que estemos ahora pero del día 25 de diciembre de 2013

```
# at -f ./genera-apunte.sh Dec 25 2013
warning: commands will be executed using /bin/sh
job 11 at Wed Dec 25 11:24:00 2013
```

e) a las 00:00 del 1 de enero de 2012

```
at -f ./genera-apunte.sh midnight Jan 01 2013
warning: commands will be executed using /bin/sh
job 11 at Wed Dec 25 11:24:00 2013
```

Utilice las órdenes atq y atrm para familiarizarse con su funcionamiento (consulte la ayuda de estas órdenes).

EJERCICIO 4.3

El proceso nuevo que se lanza al cumplirse el tiempo que se especificó en la orden at....

1. ¿Qué directorio de trabajo tiene inicialmente? ¿hereda el que tenía el proceso que invocó a at o bien es el home, directorio inicial por omisión?

- El directorio de trabajo se mantiene desde el momento de la invocación (según dice el manual de **at**), así que hereda el que tenía el proceso que invocó a **at.**

2. ¿Qué máscara de creación de archivos umask tiene? ¿es la heredada del padre o la que se usa por omisión?

- La máscara es **0022**, como dicho anteriormente, se mantiene desde el momento de la invocación, la hereda.

3. ¿Hereda las variables locales del proceso padre?

- El entorno también se mantiene desde el momento de la invocación, excepto las variables **BASH_VERSINFO, DISPLAY, EUID,GROUPS, SHELLOPTS, TERM, UID,** y _

Experimente con la orden ***at*** lanzando las órdenes adecuadas para encontrar las respuestas. (Puede encontrar información en la ayuda de ***at***)

EJERCICIO 4.4

El proceso nuevo que se lanza al cumplirse el tiempo que se especificó en la orden at.... ¿de quién es hijo? Investigue lanzando la ejecución retardada de un script que muestre la información completa sobre los procesos existentes y el pid del proceso actual; el script podría contener lo que sigue:

```
#!/bin/bash
#Nombre: script_4.4.sh

nombrearchivo=`date +%Y-%j-%T`
ps -ef > $nombrearchivo
echo Mi pid = $$ >> $nombrearchivo
```

Ejecuto el programa:

```
# at -f ./script_4.4.sh now+1 minute
warning: commands will be executed using /bin/sh
job 17 at Mon Oct 29 11:57:00 2012
```

Vemos el resultado:

```
# cat 2012-303-11\:57\:00
daemon   14192   816  0 11:56 ?        00:00:00 atd
superjes 14194 14192  0 11:56 ?        00:00:00 sh
superjes 14196 14194  0 11:56 ?        00:00:00 ps -ef
Mi pid = 14194
```

- Como podemos ver, nuestro proceso con **PID 14194** es hijo del proceso **atd**, con **PID 14192.**

EJERCICIO 4.5

Construya un script que utilice la orden find para generar en la salida estándar los archivos modificados en las últimas 24 horas (partiendo del directorio home y recorriéndolo en profundidad), la salida deberá escribirse el archivo de nombre "modificados:" seguido por año, día dentro del año y hora (dentro del directorio home). Con la orden **at** provoque que se ejecute dentro de un día a partir de este momento.

```
#!/bin/bash
#Nombre: script_4.5.sh

find ~ -mtime 1 > modificados:`date +%Y-%j-%T`
```

Ejecutamos con **at**:

```
# at -f ./script_4.5.sh now+1 day
warning: commands will be executed using /bin/sh
job 19 at Tue Oct 30 12:40:00 2012
```

EJERCICIO 4.6

Lance los procesos que sean necesarios para conseguir que exista una gran carga de trabajo para el sistema de modo que los trabajos lanzados con la orden ***batch*** no se estén ejecutando (puede simplemente construir un script que esté en un ciclo infinito y lanzarla varias veces en segundo plano). Utilice las órdenes oportunas para manejar este conjunto de procesos (la orden **jobs** para ver los trabajos lanzados, **kill** para finalizar un trabajo, …y tal vez también las órdenes **fg**, **bg** para pasar de segundo a primer plano y viceversa, <Ctrl-Z> para suspender el proceso en primer plano actual, etc). Experimente para comprobar cómo al ir disminuyendo la carga de trabajos habrá un momento en que se ejecuten los trabajos lanzados a la cola batch.

EJERCICIO 4.7

Construya tres script que deberá lanzar a las colas **c**, **d** y **e** especificando una hora constante que esté unos pocos minutos más adelante (no muchos para ser operativos). Idee qué actuación deben tener dichos script de forma que se ponga de manifiesto que de esas colas la más prioritaria es la **c** y la menos es la **e**. Visualice en algún momento los trabajos asignados a las distintas colas.

```
#!/bin/bash
#Nombre: script_4.7.sh

at -q c -f ./script_4.4.sh now+1 minute
at -q d -f ./script_4.4.sh now+1 minute
at -q e -f ./script_4.4.sh now+1 minute
```

- Vemos las prioridades:

```
$ atq -q c
29    Mon Oct 29 13:11:00 2012 c superjes
$ atq -q d
30    Mon Oct 29 13:11:00 2012 d superjes
$ atq -q e
31    Mon Oct 29 13:11:00 2012 e superjes
```

- Efectivamente (también al hacer el ls al directorio) podemos ver que se ha ejecutado antes el trabajo de la cola **c**, luego el de la cola **d** y por último el de la cola **e.**

EJERCICIO 5.1

Al igual que se investigó en la Actividad 4.4 sobre quién es el proceso padre del nuestro, lance el script construido en dicha actividad con una periodicidad de un minuto y analice los resultados.

```
#Nombre: cron_5.1
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin


#minuto hora día-del-mes mes , día-de-la-semana orden
* * * * * /home/superjes/SO/Practica_1/script_4.4.sh
```

Ejecutamos con **cron**:

```
$ crontab cron_5.1
```

EJERCICIO 5.2

Construya un script que sea lanzado con una periodicidad de un minuto y que borre los nombres de los archivos que cuelguen del directorio /tmp/varios y que comiencen por "core" (cree ese directorio y algunos archivos para poder realizar esta actividad). Utilice la opción -v de la orden rm para generar como salida una frase de confirmación de los archivos borrados; queremos que el conjunto de estas salidas se añadan al archivo /tmp/listacores.

- Creamos el script que borra los archivos:

```
#!/bin/bash
#Nombre: script_5.2.sh

rm -v /tmp/varios/core* >> /tmp/listacores
```

- Creamos el archivo que determina las opciones de crontab:

```
#Nombre: cron_5.2

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin


#minuto hora día-del-mes mes , día-de-la-semana orden
* * * * * /home/superjes/SO/Practica_1/script_5.2.sh
```

- Ejecutamos con crontab:

```
$ crontab cron_5.2
```

Pruebe la orden crontab -l para ver la lista actual de trabajos (consulte la ayuda para ver las restantes posibilidades de esta orden para gestionar la lista actual de trabajos).

```
$ crontab -l
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin


#minuto hora día-del-mes mes , día-de-la-semana orden
* * * * * /home/superjes/SO/Practica_1/script_5.2.sh
```

EJERCICIO5.3

Para asegurar que el contenido del archivo /**tmp**/**listacores** no crezca demasiado, queremos que periódicamente se deje dicho archivo solo con sus X primeras líneas (elija usted el valor de X, para que sea un número manejable pensemos en 10) (puede ser de utilidad la orden **tail**). Construya un script llamado **reducelista** (dentro del directorio ~/**SO**) que realice la función anterior y lance su ejecución con periodicidad de un minuto.

- Script **reducelista:**

```
#!/bin/bash
```

```
#Nombre: reducelista

tail -n 10 /tmp/listacores > /tmp/temporal
rm /tmp/listacores
mv /tmp/temporal /tmp/listacores
```

- Creamos el archivo al que llamará crontab:

```
#Nombre: cron_5.3
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin


#minuto hora día-del-mes mes , día-de-la-semana orden
* * * * * /home/superjes/SO/Practica_1/reducelista
```

- Ejecutamos la orden **crontab:**

```
$ crontab cron_5.3
```

EJERCICIO 5.4

Construye un sencillo script que escriba en el archivo ~/**SO**/**listabusqueda** una nueva línea con la fecha y hora actual y después el valor de la lista de búsqueda, por ejemplo:
...
2011-297-12:39:10 - /usr/local/bin:/usr/local/bin:/usr/bin...
…
Ejecute este script desde el lenguaje de órdenes y también láncelo como trabajo crontab y compare los resultados, ¿se tiene en ambos casos la misma lista de búsqueda?

- Creamos el script:

```
#Nombre: script_5.4.sh
#!/bin/bash

fecha=`date +%Y-%j-%T`

echo $fecha - $PATH >> /home/superjes/SO/Practica_1/listabusqueda
```

- Creamos el archivo que ejecutará crontab posteriormente(cada minuto):

```
#Nombre: cron_5.4
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin


#minuto hora día-del-mes mes , día-de-la-semana orden
* * * * * /home/superjes/SO/Practica_1/script_5.4.sh
```

- Probamos la ejecución directamente:

```
$ cat listabusqueda
2012-303-19:57:16 -
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
```

- Probamos la ejecutación con **crontab**:

```
$ crontab cron_5.4

$ cat listabusqueda
2012-303-20:01:01 - /usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
```

- Como podemos ver, la variable **PATH** es diferente según si lo ejecutamos directamente o con **crontab**, dado que la hemos redefinido dentro del fichero en formato crontab.

EJERCICIO 5.5

Practicamos ahora lo que acabamos de explicar situándonos en lo que hemos realizado en la actividad 5.3. Construye un script que escribirá un archivo crontab llamado crontab-reducelista que deberá contener...

…. como primera linea la asignación a la variable PATH de la lista de búsqueda actual y además el directorio $HOME/SO

…. después la indicación a cron de la ejecución con periodicidad de 1 minuto del script reducelista

Una vez construido crontab-reducelista láncelo con la orden crontab. Compruebe que con esta nueva lista de búsqueda podremos hacer alusión a reducelista especificando únicamente su nombre independientemente del directorio de trabajo en que nos situemos (no como ocurría en la Actividad 5.2 en que el directorio $HOME/SO no estaba en la lista de búsqueda).

- Script que simplemente crea el archivo **crontab-reducelista** con las líneas especificadas:

```
#!/bin/bash
#Nombre: script_5.5.sh

echo "SHELL=/bin/sh" > crontab-reducelista
echo "PATH="`pwd`"/:$HOME/SO:"$PATH >> crontab-reducelista
echo "* * * * * reducelista" >> crontab-reducelista
```

- El archive **crontab-reducelista** se crea pues con este contenido:

```
SHELL=/bin/sh
PATH=/home/superjes/SO/Practica_1/:/home/superjes/SO:/usr/local/sbin:/usr/local/bi
n:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games
* * * * * reducelista
```

- Lanzamos con **crontab:**

```
$ crontab crontab-reducelista
```

EJERCICIO 5.6

Vamos a lanzar un archivo crontab cuyo propietario es otro usuario. Visualiza el contenido del archivo /fenix/depar/lsi/so/ver-entorno y /fenix/depar/lsi/so/crontabver. Compruebe con ls –l que el propietario es el usuario lsi. Sin copiarlos, úselos para lanzar la ejecución cada minuto del script /fenix/depar/lsi/so/ver-entorno. Analice el archivo de salida: ¿de qué línea del archivo /etc/passwd se toman LOGNAME y HOME, de la línea del propietario del archivo crontab o de la línea del usuario que lanza el archivo crontab?

- (No dispongo del ordenador del aula, así que lo he hecho con un script propio para probar)

- Creo el script:

```
#!/bin/bash
#Nombre: script_5.6.sh

echo $LOGNAME > /tmp/fichero
echo $HOME >> /tmp/fichero
```

- Creo el fichero crontab:

```
#Nombre: cron_5.6
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games

* * * * * /home/superjes/SO/Practica_1/script_5.6.sh
```

- Con otro usuario(en este caso uno llamado **pato**), lanzo con la orden **crontab** el fichero anterior:

```
$ crontab /home/superjes/SO/Practica_1/cron_5.6
```

- Compruebo el fichero creado:

```
$ cat /tmp/fichero
pato
/home/pato
```

- Como vemos, los valores se toman del usuario que lanza el comando crontab.

EJERCICIO 5.7

El objetivo es ejecutar todos los días a las 0 horas 0 minutos una copia de los archivos que cuelguen de $HOME que se hayan modificado en las últimas 24 horas. Vamos a programar este salvado incremental utilizando la orden **find** que usábamos en la actividad 4.5; ahora queremos que se copien los archivos encontrados por find utilizando la orden **cpio:**
<orden find de la actividad 4.5> | cpio -pmduv /tmp/salvado$HOME

- Creo el script:

```
#!/bin/bash
#Nombre: script_5.7.sh

find ~ -mtime 1|cpio -pmduv /tmp/salvado$HOME
```

- Creo el archivo crontab:

```
#Nombre: cron_5.7
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin


0 0 * * * /home/superjes/SO/Practica_1/script_5.7.sh
```

- Ejecutamso con **crontab**:

```
$ crontab cron_5.7
```

EJERCICIO 5.8

Como usuario root, deshabilite/habilite a un determinado usuario para utilizar el servicio cron; compruebe que efectivamente funciona.

- Como root añado (o creo si no existe) al archivo **/etc/cron.deny** una nueva línea con el nombre del usuario (en este caso el usuario **pato**) al que quiero prohibir el uso de **crontab**:

```
# echo pato >> /etc/cron.deny
```

- Nos metemos como el usuario que acabamosde indicar y probamos a ejecutar crontab:

```
# su pato
Contraseña:

pato@superjes-VirtualBox:/home/superjes/SO/Practica_1$ crontab cron_5.6
You (pato) are not allowed to use this program (crontab)
See crontab(1) for more information
```

- Como podemos ver, efectivamente no tenemos permisos para ejecutar crontab.

EJERCICIO 5.9

Iniciar y terminar el servicio cron. Pruebe las siguientes órdenes para iniciar y terminar este servicio:
Iniciar el servicio cron: /sbin/service crond start
Terminar el servicio cron: /sbin/service crond stop

- En mi Ubuntu se puede hacer directamente de este modo:

```
# service cron stop
cron stop/waiting

# service cron start
cron start/running, process 21796
```

Practica 2

Sesion 1

EJERCICIO 1

¿Qué hace el siguiente programa? Probad tras la ejecución del programa las siguientes órdenes del shell: $>cat archivo y $> od -c archivo

```
/*
tarea1.c
Trabajo con llamadas al sistema del Sistema de Archivos ''POSIX 2.10 compliant''
Probar tras la ejecución del programa: $>cat archivo y $> od -c archivo
*/

#include<sys/types.h>     //Primitive system data types for abstraction of
implementation-dependent data types.
                                        //POSIX Standard: 2.6 Primitive System Data
Types <sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdlib.h>
#include<stdio.h>
#include<errno.h>

char buf1[]="abcdefghij";
char buf2[]="ABCDEFGHIJ";

int main(int argc, char *argv[])
{
int fd;

if( (fd=open("archivo",O_CREAT|O_TRUNC|O_WRONLY,S_IRUSR|S_IWUSR))<0) {
      printf("\nError %d en open",errno);
      perror("\nError en open");
      exit(-1);
}
if(write(fd,buf1,10) != 10) {
      perror("\nError en primer write");
      exit(-1);
}

if(lseek(fd,40,SEEK_SET) < 0) {
      perror("\nError en lseek");
      exit(-1);
}

if(write(fd,buf2,10) != 10) {
      perror("\nError en segundo write");
      exit(-1);
}
```

```
return 0;
}
```

- Lo que hace el siguiente programa es crear dos arrays: *buf1* y *buf2* que contienen 10 caracteres cada uno. Crea un entero *fd* al cual asigna el valor de la llamada al sistema *open* del archivo "archivo", el resultado del cual da error si es menor que cero. Con **O_CREAT** se crea si no existe, con **O_TRUNC** si existe el fichero y tiene habilitada la escritura, lo sobreescribe a tamaño 0, con **O_WRONLY** decimos que solo se permite escritura, con **S_IRUSR** comprobamos que el usuario tiene permiso de lectura, por último con **S_IWUSR** comprobamos que el usuario tiene permiso de escritura.
- Después comprueba que si, después de hacer la orden *write* del primer búfer, el resultado asociado a esta operación es distinto de 10 se muestra error, dado que hemos escrito los 10 caracteres del primer búfer en el archivo.
- Después con *lseek* ponemos el puntero del archivo en la posición 40(en bytes) desde SEEK_SET (inicio del fichero), y da error si el resultado de la operación es menor que 0. Con esto conseguimos que el puntero se sitúe justo después de los 40 bytes que hemos escrito previamente, al final de los 10 caracteres de *buf1.*
- Por último llamamos a *write* para el segundo búfer igual que hemos hecho antes con el primero. Esto imprime los 10 caracteres de *buf2*.
- Hacemos el cat del archivo y vemos que está bien impreso:

```
$ cat archivo
abcdefghijABCDEFGHIJ
```

- Hacemos el od del archivo:

```
$ od -c archivo
0000000   a   b   c   d   e   f   g   h   i   j  \0  \0  \0  \0  \0  \0
0000020  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000040  \0  \0  \0  \0  \0  \0  \0  \0   A   B   C   D   E   F   G   H
0000060   I   J
0000062
```

EJERCICIO 2

Implementa un programa que acepte como argumento un "pathname", abra el archivo correspondiente y utilizando un tamaño de partición de los bytes del archivo igual a 80 Bytes cree un archivo de salida en el que debe aparecer lo siguiente:

```
Bloque1
//los primeros 80 Bytes
Bloque2
//los siguientes 80 Bytes
...
```

Si no se pasa un argumento al programa se debe utilizar la entrada estándar como archivo de entrada.

Modificación adicional. ¿Cómo tendrías que modificar el programa para que una vez finalizada la escritura en el archivo de salida y antes de cerrarlo, pudiésemos indicar en su primera línea el número de etiquetas "bloque i" escritas de forma que tuviese la siguiente apariencia:

```
El número de bloques es <nº_bloques>
Bloque1
//los primeros 80 Bytes
Bloque2
//los siguientes 80 Bytes
…
```

Ejercicio2.c

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
#include<errno.h>
#include<unistd.h>


int main(int argc, char *argv[])
{
	int cont=1,leidos;
	int filein,fileout;
	char cadena[30];
	char cad_bloque[40];
	char salto_linea[2]="\n";
	char caracter[1];
	int num_char = 1;

	if (argc==2)
	{
		filein=open(argv[1], O_RDONLY);
	}
	else{
		filein=STDIN_FILENO;
	}

	fileout=open("archivo_salida", O_CREAT|O_TRUNC|O_WRONLY,S_IRUSR|S_IWUSR);

	if (fileout < 0){
		 printf("El fichero de salida no se pudo abrir correctamente\n");
		 exit(-1);
	}

	while((leidos=read(filein,caracter,1))!=0){
		if (num_char == 1 || num_char%80 == 0){
			if (num_char != 1)
			    write(fileout, salto_linea, strlen(salto_linea));
		else{
		     sprintf(cad_bloque, "El numero de bloques es <%d>\n",cont);
			   write(fileout, cad_bloque, strlen(cad_bloque));
		}
```

```
            sprintf(cad_bloque, "%s%d\n", "Bloque", cont);
             write(fileout, cad_bloque, strlen(cad_bloque));
             cont++;
           }
           write(fileout, caracter, 1);
           num_char++;


      }

      sprintf(cad_bloque, "El numero de bloques es <%d>\n", cont);
      lseek(fileout,0,SEEK_SET);
      write(fileout, cad_bloque, strlen(cad_bloque));

      close(filein);
      close(fileout);


      return 0;
}
```

EJERCICIO 3

¿Qué hace el siguiente programa?

```
/*
tarea2.c
Trabajo con llamadas al sistema del Sistema de Archivos ''POSIX 2.10
compliant''
*/
#include<sys/types.h>
#include<unistd.h>
#include<sys/stat.h>
#include<stdio.h>
#include<errno.h>
#include<string.h>
int main(int argc, char *argv[])
{
int i;
struct stat atributos;
char tipoArchivo[30];
if(argc<2) {
printf("\nSintaxis de ejecucion: tarea2 [<nombre_archivo>]+\n\n");
exit(-1);
}
for(i=1;i<argc;i++) {
printf("%s: ", argv[i]);
if(lstat(argv[i],&atributos) < 0) {
printf("\nError al intentar acceder a los atributos de %s",argv[i]);
perror("\nError en lstat");
}
else {
if(S_ISREG(atributos.st_mode)) strcpy(tipoArchivo,"Regular");
else if(S_ISDIR(atributos.st_mode)) strcpy(tipoArchivo,"Directorio");
```

```
else if(S_ISCHR(atributos.st_mode)) strcpy(tipoArchivo,"Especial de
caracteres");
else if(S_ISBLK(atributos.st_mode)) strcpy(tipoArchivo,"Especial de
bloques");
else if(S_ISFIFO(atributos.st_mode)) strcpy(tipoArchivo,"Cauce con nombre
(FIFO)");
else if(S_ISLNK(atributos.st_mode)) strcpy(tipoArchivo,"Enlace relativo
(soft)");
else if(S_ISSOCK(atributos.st_mode)) strcpy(tipoArchivo,"Socket");
else strcpy(tipoArchivo,"Tipo de archivo desconocido");
printf("%s\n",tipoArchivo);
}
}
return 0;
}
```

- Una vez que hemos compilado el ejercicio y lo hemos ejecutado pasándole como argumento <nombre_archivo> , lo que hace es decirnos que tipo de archivo es, ya sea un archivo regular, un directorio, un dispositivo de bloques, etc...

- Internamente, el programa comprueba cada flag correspondiente a un archivo determinado con el archivo que hemos introducido como parámetro, y si se activa nos mostrará el tipo de archivo en el cual se ha activado.

EJERCICIO 4

Define una macro en lenguaje C que implemente la macro S_ISREG(mode) usando para ello los flags definidos en <sys/stat.h> para el campo st_mode de la struct stat. y comprueba que funciona en un programa simple.

```
#define S_ISREG2(mode) (mode & S_IFMT == S_IFREG)
```

Sesion 2

EJERCICIO 1

¿Qué hace el siguiente programa?

```
/*
tarea3.c
Trabajo con llamadas al sistema del Sistema de Archivos ''POSIX 2.10
compliant''
Este programa fuente está pensado para que se cree primero un programa con la
parte de CREACION DE ARCHIVOS y se haga un ls -l para fijarnos en los permisos
y entender la llamada umask.
En segundo lugar (una vez creados los archivos) hay que crear un segundo
programa con la parte de CAMBIO DE PERMISOS para comprender el cambio de
permisos relativo a los permisos que actualmente tiene un archivo frente a un
establecimiento de permisos absoluto.
*/
#include<sys/types.h>
#include<unistd.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<stdio.h>
#include<errno.h>
int main(int argc, char *argv[])
{
int fd1,fd2;
struct stat atributos;
//CREACION DE ARCHIVOS
if( (fd1=open("archivo1",O_CREAT|O_TRUNC|O_WRONLY,S_IRGRP|S_IWGRP|S_IXGRP))<0)
{
printf("\nError %d en open(archivo1,...)",errno);
Sistemas Operativos II Guión de Prácticas, pág.12
perror("\nError en open");
exit(-1);
}
umask(0);
if( (fd2=open("archivo2",O_CREAT|O_TRUNC|O_WRONLY,S_IRGRP|S_IWGRP|S_IXGRP))<0)
{
printf("\nError %d en open(archivo2,...)",errno);
perror("\nError en open");
exit(-1);
}
//CAMBIO DE PERMISOS
if(stat("archivo1",&atributos) < 0) {
printf("\nError al intentar acceder a los atributos de archivo1");
perror("\nError en lstat");
exit(-1);
}
if(chmod("archivo1", (atributos.st_mode & ~S_IXGRP) | S_ISGID) < 0) {
perror("\nError en chmod para archivo1");
exit(-1);
}
if(chmod("archivo2",S_IRWXU | S_IRGRP | S_IWGRP | S_IROTH) < 0) {
perror("\nError en chmod para archivo2");
```

```
exit(-1);
}
return 0;
}
```

- Lo que hace el programa es crear un archivo llamado **archivo1** con permisos de lectura, escritura y ejecución para el grupo, seguidamente pone la máscara a 0 con la orden **umask(0)** y después crea otro archivo llamado **archivo2**, con los mismos permisos que el archivo anterior.
- Luego comprueba que se puede acceder a los atributos del primer archivo con la orden **stat.**
- Después con **chmod** cambiamos los permisos del primer archivo haciendo un AND lógico del estado del archivo(accediendo al struct **atributos**) con el negado del permiso de ejecución para el grupo, con lo que le quitamos el permiso de ejecución para el grupo. También activamos la asignación del GID del propietario al GID efectivo del proceso que ejecute el archivo.
- Por último con **chmod** cambiamos los permisos del segundo archivo para que tenga todos los permisos para el propio usuario, permiso de lectura y escritura para el grupo, y lectura para el resto de usuarios.

EJERCICIO 2

Realiza un programa en C utilizando las llamadas al sistema necesarias que acepte como entrada:
• Un argumento que representa el ***'pathname'*** de un directorio.
• Otro argumento que es un ***número octal de 4 dígitos*** (similar al que se puede utilizar para cambiar los permisos en la llamada al sistema `chmod`).
El programa tiene que usar el número octal indicado en el segundo argumento para cambiar los permisos de todos los archivos que se encuentren en el directorio indicado en el primer argumento.
El programa debe proporcionar en la salida estándar una línea para cada archivo del directorio que esté formada por:

```
<nombre_de_archivo> : <permisos_antiguos> <permisos_nuevos>
```

Si no se pueden cambiar los permisos de un determinado archivo se debe especificar la siguiente información en la línea de salida:

```
<nombre_de_archivo> : <errno> <permisos_antiguos>
```

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
#include<errno.h>
#include<unistd.h>
#include<dirent.h>
```

```
int main(int argc, char *argv[])
{
	DIR *direct;
	unsigned int permisos;
	char *pathname;
	struct stat atributos;
	struct dirent *ed;
	char cadena[100];
	char cadena2[100];
	extern int errno;



	if (argc==3)
	{
		pathname=argv[1];
		direct=opendir(pathname);
		permisos=strtol(argv[2],NULL,8);
	}
	else{
		printf("Uso: ejercicio2.c <pathname> <permisos>\n");
		exit(-1);
	}


	readdir(direct);

	while((ed=readdir(direct))!=NULL){
		sprintf(cadena,"%s/%s",pathname,ed->d_name);

		if(stat(cadena,&atributos) < 0) {
			printf("\nError al intentar acceder a los atributos de
archivo");
			perror("\nError en lstat");
			exit(-1);
		}
		if(S_ISREG(atributos.st_mode)){

			sprintf(cadena2,"%s",ed->d_name);


			printf("%s: %o ",cadena2,atributos.st_mode);

			chmod(cadena,permisos);

			if(chmod(cadena,permisos) < 0) {
				printf("Error: %s\n",strerror(errno));
			}
			else{
				stat(cadena,&atributos);
				printf("%o \n",atributos.st_mode);
			}

		}
	}
```

```
	closedir(direct);

	return 0;
}
```

EJERCICIO 3

Programa una nueva orden que recorra la jerarquía de subdirectorios existentes a partir de uno dado como argumento y devuelva la cuenta de todos aquellos archivos regulares que tengan permiso de ejecución para el *grupo* y para *otros*. Además del nombre de los archivos encontrados, deberá devolver sus números de inodo y la suma total de espacio ocupado por dichos archivos. El formato de la nueva orden será:

```
$ ./buscar <pathname>
```

donde <*pathname*> especifica el nombre del directorio a partir del cual queremos que empiece a analizar la estructura del árbol de subdirectorios. En caso de que no se le de argumento, tomará como punto de partida el *directorio actual*. Ejemplo de la salida después de ejecutar el programa:

```
Los i-nodos son:
./a.out 55
./bin/ej 123
./bin/ej2 87
...
Existen 24 archivos regulares con permiso x para grupo y otros
El tamaño total ocupado por dichos archivos es 2345674 bytes
```

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<string.h>
#include<stdlib.h>
#include<stdio.h>
#include<errno.h>
#include<unistd.h>
#include<dirent.h>
#define mymask(mode) ((mode) & ~S_IFMT)
// Permisos de ejecución para grupo y otros.
#define S_IFXGRPOTH  011
// Se define la macro con la regla para comprobar si tiene permiso x en grupo y
otros.
#define regla1(mode) ((((mode) & ~S_IFMT) & 011) == S_IFXGRPOTH)

void buscar_dir(DIR *direct, char pathname[], int *reg, int *tamanio){
	struct stat atributos;
	struct dirent *ed;
	DIR *direct_act;
	char cadena[500];

	while((ed=readdir(direct)) != NULL){
		// Ignorar el directorio actual y el supoerior
	  if (strcmp(ed->d_name, ".") != 0 && strcmp(ed->d_name, "..") != 0){
			sprintf(cadena,"%s/%s",pathname,ed->d_name);

			if(stat(cadena,&atributos) < 0) {
```

```
                        printf("\nError al intentar acceder a los atributos de
archivo");
                        perror("\nError en lstat");
                        exit(-1);
                }

                if (S_ISDIR(atributos.st_mode)){
                        if ((direct_act = opendir(cadena)) == NULL)
                            printf("\nError al abrir el directorio: [%s]\n",
cadena);
                        else
                            buscar_dir(direct_act, cadena, reg, tamanio);
            }else{

                        printf("%s %ld \n", cadena, atributos.st_ino);
                        if (S_ISREG(atributos.st_mode)){
                            if (regla1(atributos.st_mode)){
                                (*reg)++;
                                (*tamanio) += (int) atributos.st_size;

                            }

                        }
            }
        }

      }
      closedir(direct);
}

int main(int argc, char *argv[])
{
      DIR *direct;
      char pathname[500];
      int reg=0,tamanio=0;



      if (argc==2)
      {
            strcpy(pathname,argv[1]);
      }
      else{
            strcpy(pathname,".");
      }



      if((direct=opendir(pathname)) == NULL){
            printf("\nError al abrir directorio\n");
            exit(-1);
      }



      printf("Los inodos son: \n\n");
```

```
	buscar_dir(direct,pathname,&reg,&tamanio);

	printf("Hay %d archivos regulares con permiso x para grupo y otros\n",reg);
	printf("El tamaño total ocupado por dichos archivos es %d bytes\n",tamanio);


	return 0;
}
```

Sesion 3

EJERCICIO 1

Implementa un programa en C que tenga como argumento un número entero. Este programa debe crear un proceso hijo que se encargará de comprobar si dicho número es un número par o impar e informará al usuario con un mensaje que se enviará por la salida estándar. A su vez, el proceso padre comprobará si dicho número es divisible por 4, e informará si lo es o no usando igualmente la salida estándar.

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

int main(int argc, char *argv[]){

	if (argc != 2){}
		perror("\nError , parametro = un numero a comprobar");
		exit(-1);
	}


	pid_t pid;

	pid = fork();

	if (pid < 0){
		perror("\nError en el fork");
		exit(-1);
	}
	else

	if (pid == 0)
	{	 /* Proceso Hijo */
	printf("Hola , soy el proceso hijo y mi pid es: %d  y el de mi padre es %d,
y voy a comprobar si el numero introducido es par o impar \n ", getpid() ,
getppid());
	int numero =  atoi(argv[1]);

	if ((numero % 2) == 1)
		printf("El numero %d introducido es impar \n" , numero);
	else printf("El numero %d introducido es par \n" , numero);

	}
	else  if (pid){
		printf("Hola , soy el proceso padre y mi pid es: %d  y el de mi hijo
es %d, y voy a comprobar si el numero introducido es divisible por 4 \n", getpid()
, pid);
	int numero =  atoi(argv[1]);

	if (numero % 4 == 0)
		printf("El numero %d introducido es divisible por 4 \n" , numero);
```

```
      else printf("El numero %d introducido es indivisible por 4 \n" , numero);
      }
return(0);
}
```

EJERCICIO 2

¿Qué hace el siguiente programa? Intenta entender lo que ocurre con las variables y sobre todo con los mensajes por pantalla cuando el núcleo tiene activado/desactivado el mecanismo de buffering.

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

int global=6;
char buf[]="cualquier mensaje de salida\n";

int main(int argc, char *argv[])
{
int var;
pid_t pid;

var=88;
if(write(STDOUT_FILENO,buf,sizeof(buf)+1) != sizeof(buf)+1) {
      perror("\nError en write");
      exit(-1);
}
//(1)if(setvbuf(stdout,NULL,_IONBF,0)) {
//    perror("\nError en setvbuf");
//}
printf("\nMensaje previo a la ejecución de fork");

if( (pid=fork())<0) {
      perror("\nError en el fork");
      exit(-1);
}
else if(pid==0) {  //proceso hijo ejecutando el programa
            global++;
            var++;
      } else  //proceso padre ejecutando el programa
            sleep(1);

printf("\npid= %d, global= %d, var= %d\n", getpid(),global,var);
exit(0);

}
```

- Lo que hace el programa es declaras una variable global con valor **6** y un string cualquiera llamado **buf**. En la función principal se declara una variable entera y una de tipo pid_t. Da el valor **88** a la variable entera y procede a imprimir por pantalla el string de antes mediante la función **write** en la salida estándar (**STDOUT_FILENO**).

- Luego procede a crear el proceso hijo con la función **fork** y lo asigna a la variable tipo pid_t **pid.** Si pid vale 0 se ejecuta el código del proceso hijo, que lo único que hace es incrementar tanto la variable global como la entera var.
- Si no vale 0 (es decir, su valor es el PID del proceso hijo) ejecuta el código del proceso padre, que solamente hace un sleep de 1 segundo.

EJERCICIO 3

Implementa un programa que lance cinco procesos hijo. Cada uno de ellos se identificará en la salida estándar, mostrando un mensaje del tipo `Soy el hijo PID`. El proceso padre simplemente tendrá que esperar la finalización de todos sus hijos y cada vez que detecte la finalización de uno de sus hijos escribirá en la salida estándar un mensaje del tipo:

```
Acaba de finalizar mi hijo con <PID>
Sólo me quedan <NUM_HIJOS> hijos vivos
```

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

int main(){

  int i, estado;
  pid_t PID;

  //CREAMOS HIJOS
  for(i=0; i<5; i++){
    if((PID = fork())<0){
      perror("Error en fork\n");
      exit(-1);
    }
    if(PID==0){//Hijo imprime y muere
      printf("Soy el hijo PID = %i\n", getpid());
      exit(0);
    }
  }

  //ESPERAMOS HIJOS
  for(i=4; i>=0; i--){
    PID = wait(&estado);
    printf("Ha finalizado mi hijo con PID = %i\n", PID);
    printf("Solo me quedan %i hijos vivos\n", i);
  }
}
```

EJERCICIO 4

Implementa una modificación sobre el anterior programa en la que el proceso padre espera primero a los hijos creados en orden impar (1º,3º,5º) y después a los hijos pares (2º y 4º).

```
#include<sys/types.h>        //Primitive system data types for abstraction of
implementation-dependent data types.
                                                  //POSIX Standard: 2.6 Primitive
System Data Types <sys/types.h>
#include<unistd.h>                //POSIX Standard: 2.10 Symbolic Constants
<unistd.h>
#include<stdio.h>
#include<errno.h>
#include<stdlib.h>

int main(int argc, char *argv[])
{
       int PIDs[5];
       pid_t PID;
       int estado, hijos = 5, i;
       //vaciar el bujer, para que los printf salgan como deben.
       if(setvbuf(stdout,NULL,_IONBF,0)) {
             perror("\nError en setvbuf")  ;
       }

       printf("Se va a empezar a ejecutar el proceso creadoar de hijos\n" ) ;

       for(i=1; i < 6; i++){
          if( (PIDs[i-1]=fork())<0) {
                perror("\nError en el fork" ) ;
                exit(-1);
           }
           else if(PIDs[i-1]==0) {  //proceso hijo ejecutando el programa
                printf("Soy el hijo %d\n", getpid());
                exit(0);
           }
       }

       for(i=0; i<5; i = i +2){  //proceso padre ejecutando el programa
            waitpid(PIDs[i],&estado);
            printf("Acaba de finalizar mi hijo con PID = %d y estado %d\n",
PIDs[i], estado);
           printf("Solo me quedan %d hijos vivos, este es el %dº hijo.\n", --
hijos, i+1);
       }

       for(i=1; i<4; i = i +2){  //proceso padre ejecutando el programa
              waitpid(PIDs[i],&estado);
              printf("Acaba de finalizar mi hijo con PID = %d y estado %d\n",
PIDs[i], estado);
             printf("Solo me quedan %d hijos vivos, este es el %dº hijo.\n", --
hijos, i+1);
       }

       exit(0);
}
```

EJERCICIO 5

¿Qué hace el siguiente programa?

```
/*
```

```
tarea5.c
Trabajo con llamadas al sistema del Subsistema de Procesos conforme a POSIX
2.10
*/
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdio.h>
#include<errno.h>
int main(int argc, char *argv[]){
      pid_t pid;
      int estado;
      if( (pid=fork())<0) {
            perror("\nError en el fork");
            exit(-1);
      }
      else if(pid==0) { //proceso hijo ejecutando el programa
          if( (execl("/usr/bin/ldd","ldd","./tarea8",NULL)<0)) {
            perror("\nError en el execl");
            exit(-1);
        }
      }
      wait(&estado);
      printf("\nMi hijo %d ha finalizado con el estado %d\n",pid,estado);
      exit(0);
}
```

EJERCICIO 6

Escribe un programa que acepte nombres de programa escritos en la entrada estándar, los ejecute en `background` o `foreground` según lo desee el usuario, y proporcione en la salida estándar el resultado de la ejecución de dichos programas. Es decir, construye un shell reducido a la funcionalidad de ejecución de programas.

```
#include<sys/types.h>
#include<sys/wait.h>
#include<unistd.h>
#include<stdio.h>
#include<errno.h>
#include<stdlib.h>

int main(int argc, char *argv[]){
  pid_t pid;
  int estado;

  if(argc<2){
        printf("Sintaxis incorrecta\n")  ;
        exit(-1);
  }

  if( (pid=fork())<0) {
    perror("\nError en el fork");
    exit(-1);
  }
  else
  if(pid==0) { //proceso hijo ejecutando el programa
    if( (execl(argv[1] , "orden",NULL)<0)) {
    perror("\nError en el execl");
```

```
    exit(-1);
    }
  }
  wait(&estado);
  printf("\nMi hijo %d ha finalizado con el estado %d\n",pid,estado);
  exit(0);
}
```

Sesion 4

EJERCICIO 1

Consulte en el manual las llamadas al sistema para la creación de archivos especiales en general (mknod) y la específica para archivos FIFO (mkfifo). Pruebe a ejecutar el siguiente código correspondiente a dos programas que modelan el problema del productor/consumidor, los cuales utilizan como mecanismo de comunicación un cauce FIFO.
Determine en qué orden y manera se han de ejecutar los dos programas para su correcto funcionamiento y cómo queda reflejado en el sistema que estamos utilizando un cauce FIFO. Justifique la respuesta.

```
//consumidorFIFO.c

//Consumidor que usa mecanismo de comunicacion FIFO.



#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <unistd.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#define ARCHIVO_FIFO "ComunicacionFIFO"


int main(void)
{
int fd;
char buffer[80];// Almacenamiento del mensaje del cliente.
int leidos;


//Crear el cauce con nombre (FIFO) si no existe
umask(0);
mknod(ARCHIVO_FIFO,S_IFIFO|0666,0);
//también vale: mkfifo(ARCHIVO_FIFO,0666);


//Abrir el cauce para lectura-escritura
if ( (fd=open(ARCHIVO_FIFO,O_RDWR)) <0) {
perror("open");
exit(-1);
}


//Aceptar datos a consumir hasta que se envíe la cadena fin
while(1) {
leidos=read(fd,buffer,80);
if(strcmp(buffer,"fin")==0) {
close(fd);
```

```
return 0;

}

printf("\nMensaje recibido: %s\n", buffer);

}


return 0;

}
```

```
//productorFIFO.c

//Productor que usa mecanismo de comunicacion FIFO


#include<sys/types.h>

#include<sys/stat.h>

#include<fcntl.h>

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#include<errno.h>

#define ARCHIVO_FIFO "ComunicacionFIFO"


int main(int argc, char *argv[])

{

int fd;


//Comprobar el uso correcto del programa

if(argc != 2) {
```

```
printf("\nproductorFIFO: faltan argumentos (mensaje)");

printf("\nPruebe: productorFIFO <mensaje>, donde <mensaje> es una cadena de
caracteres.\n");

exit(-1);

}



//Intentar abrir para escritura el cauce FIFO

if( (fd=open(ARCHIVO_FIFO,O_WRONLY)) <0) {

perror("\nError en open");

exit(-1);

}



//Escribir en el cauce FIFO el mensaje introducido como argumento

if( (write(fd,argv[1],strlen(argv[1])+1)) != strlen(argv[1])+1) {

perror("\nError al escribir en el FIFO");

exit(-1);

}



close(fd);

return 0;

}
```

- Para el correcto funcionamiento del productor/consumidor primero tenemos que ejecutar el programa consumidor, ya sea en un terminal en segundo plano o en primer plano.Una vez iniciado el proceso consumidor ya podemos ejecutar el productor (en el mismo terminal si el consumidor está en segundo plano o en otro terminal si éste está en primer plano) pasándole como parámetro entre comillas el string a escribir. Como el consumidor está en un bucle infinito esperando a la entrada estándar, imprimirá el mensaje recibido inmediatamente, hasta recibir la cadena “fin”.
- Queda reflejado que estamos usando un cauce FIFO porque permanece el archivo FIFO que hemos usado para la comunicación.

EJERCICIO 2

Consulte en el manual en línea la llamada al sistema pipe para la creación de cauces sin nombre. Pruebe a ejecutar el siguiente programa que utiliza un cauce sin nombre y describa la función que realiza. Justifique la respuesta.

```
/*

tarea6.c

Trabajo con llamadas al sistema del Subsistema de Procesos y Cauces conforme a
POSIX 2.10

*/


#include<sys/types.h>

#include<fcntl.h>

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>


int main(int argc, char *argv[])

{

int fd[2], numBytes;

pid_t PID;

char mensaje[]= "\nEl primer mensaje transmitido por un cauce!!\n";

char buffer[80];


pipe(fd); // Llamada al sistema para crear un cauce sin nombre


if ( (PID= fork())<0) {

perror("fork");

exit(1);

}
```

```
if (PID == 0) {

//Cierre del descriptor de lectura en el proceso hijo

close(fd[0]);

// Enviar el mensaje a través del cauce usando el descriptor de escritura

write(fd[1],mensaje,strlen(mensaje)+1);

exit(0);

}

else { // Estoy en el proceso padre porque PID != 0

//Cerrar el descriptor de escritura en el proceso padre

close(fd[1]);

//Leer datos desde el cauce.

numBytes= read(fd[0],buffer,sizeof(buffer));

printf("\nEl número de bytes recibidos es: %d",numBytes);

printf("\nLa cadena enviada a través del cauce es: %s", buffer);

}


return(0);

}
```

- Lo primero que hace es crear un cauce con la orden **pipe**, pasándole como parámetro el vector de enteros **fd,** lo cual asigna por defecto el modo lectura a fd[0] y el de escritura a fd[1].
- Después crea el hijo con la orden **fork.** Seguidamente entramos en la zona de código del hijo comprobando que el PID sea 0, y lo primero que hace es cerrar el descriptor de lectura del mismo. Después envía el mensaje a través del cauce usando el descriptor de escritura fd[1], mediante la orden **write.**
- En la zona de código del proceso padre, en cambio, lo primero que hacemos es cerrar el descriptor de escritura fd[1] y seguidamente lee los datos del cauce usando el descriptor de lectura fd[0], mediante la orden **read**. Por último imprime el número de bytes de la cadena recibida y la misma cadena.
- Hecho esto hemos conseguido la comunicación por cauces deseada en la dirección deseada, la cual también se podría hacer alrevés.

EJERCICIO 3

Redirigiendo las entradas y salidas estándares de los procesos a los cauces podemos escribir un programa en lenguaje C que permita comunicar órdenes existentes sin necesidad de reprogramarlas, tal como hace el shell (por ejemplo ls | sort). En particular, ejecute el siguiente programa que ilustra la comunicación entre proceso padre e hijo a través de un cauce sin nombre redirigiendo la entrada estándar y la salida estándar del padre y el hijo respectivamente.

```
/*

tarea7.c

Programa ilustrativo del uso de pipes y la redirección de entrada y

salida estándar: "ls | sort"

*/


#include<sys/types.h>

#include<fcntl.h>

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>


int main(int argc, char *argv[])

{

int fd[2];

pid_t PID;


pipe(fd); // Llamada al sistema para crear un pipe


if ( (PID= fork())<0) {

perror("fork");

exit(1);

}
```

```
if(PID == 0) { // ls
//Establecer la dirección del flujo de datos en el cauce cerrando
// el descriptor de lectura de cauce en el proceso hijo
close(fd[0]);

//Redirigir la salida estándar para enviar datos al cauce
//-------------------------------------------------------
//Cerrar la salida estándar del proceso hijo
close(STDOUT_FILENO);

//Duplicar el descriptor de escritura en cauce en el descriptor
//correspondiente a la salida estándar (stdout)
dup(fd[1]);
execlp("ls","ls",NULL);
}
else { // sort. Estoy en el proceso padre porque PID != 0

//Establecer la dirección del flujo de datos en el cauce cerrando
// el descriptor de escritura en el cauce del proceso padre.
close(fd[1]);

//Redirigir la entrada estándar para tomar los datos del cauce.
//Cerrar la entrada estándar del proceso padre
close(STDIN_FILENO);

//Duplicar el descriptor de lectura de cauce en el descriptor
//correspondiente a la entrada estándar (stdin)
dup(fd[0]);
```

```
execlp("sort","sort",NULL);

}


return(0);

}
```

EJERCICIO 4

Compare el siguiente programa con el anterior y ejecútelo. Describa la principal diferencia, si existe, tanto en su código como en el resultado de la ejecución.

```
/*

tarea8.c

Programa ilustrativo del uso de pipes y la redirección de entrada y

salida estándar: "ls | sort", utilizando la llamada dup2.

*/


#include<sys/types.h>

#include<fcntl.h>

#include<unistd.h>

#include<stdio.h>

#include<stdlib.h>

#include<errno.h>


int main(int argc, char *argv[])

{

int fd[2];

pid_t PID;


pipe(fd); // Llamada al sistema para crear un pipe


if ( (PID= fork())<0) {
```

```
perror("\Error en fork");
exit(-1);
}
if (PID == 0) { // ls
//Cerrar el descriptor de lectura de cauce en el proceso hijo
close(fd[0]);

//Duplicar el descriptor de escritura en cauce en el descriptor
//correspondiente a la salida estda r (stdout), cerrado previamente en
//la misma operación
dup2(fd[1],STDOUT_FILENO);
execlp("ls","ls",NULL);
}
else { // sort. Proceso padre porque PID != 0.
//Cerrar el descriptor de escritura en cauce situado en el proceso padre
close(fd[1]);

//Duplicar el descriptor de lectura de cauce en el descriptor
//correspondiente a la entrada estándar (stdin), cerrado previamente en
//la misma operación
dup2(fd[0],STDIN_FILENO);
execlp("sort","sort",NULL);
}

return(0);
}
```

- El resultado de la ejecución es el mismo en ambos.

- Las diferencias a nivel de código son que el primer programa usa la orden **close** para cerrar la salida estándar y así dejar la entrada del descriptor de lectura del hijo libre y luego **dup** para duplicar el descriptor de escritura en el cauce**.** Lo mismo hace con el proceso padre pero con la entrada estándar.

- El segundo usa solamente la orden **dup2**, que hace lo mismo que las otras dos juntas en una sola orden, cerrando el descriptor antiguo y duplicando el descriptor después. Se garantiza que la llamada es atómica, por lo que si por ejemplo, si llega una señal al proceso, toda la operación transcurrirá antes de devolverle el control al núcleo para gestionar la señal.
- Así pues hemos creado un cauce en el que el proceso hijo ejecuta la orden **ls** y lo redirecciona al descriptor de escritura de salida deseado, no a la salida estándar, con lo cual el proceso padre, en el que también hemos redirigido la entrada para que no sea la estándar, recibe el la información de la orden que ejecutó el hijo y seguidamente ejecuta la orden **sort**. Con la salida que da el programa podemos comprobar que el cauce se ha efectuado correctamente.

EJERCICIO 5

Este ejercicio se basa en la idea de utilizar varios procesos para realizar partes de una computación en paralelo. Para ello, deberá construir un programa que siga el esquema de computación maestro-esclavo, en el cual existen varios procesos trabajadores (esclavos) idénticos y un único proceso que reparte trabajo y reúne resultados (maestro). Cada esclavo es capaz de realizar una computación que le asigne el maestro y enviar a éste último los resultados para que sean mostrados en pantalla por el maestro.
El ejercicio concreto a programar consistirá en el cálculo de los números primos que hay en un intervalo. Será necesario construir dos programas, maestro y esclavo, que darán lugar a tres procesos, 1 maestro y 2 esclavos. Además se tendrá que tener en cuenta la siguiente especificación:

1. El intervalo de números naturales donde calcular los número primos se pasará como argumento al programa maestro. El maestro creará dos procesos esclavos y dividirá el intervalo en dos subintervalos de igual tamaño asignando cada uno de ellos a un esclavo (también pasando cada subintervalo como argumento a cada esclavo), los cuales realizarán el cálculo de números primos en el subintervalo asignado. Por ejemplo, si al maestro le proporcionamos el intervalo entre 1000 y 2000, entonces un esclavo debe calcular en devolver los números primos comprendidos en el subintervalo entre 1000 y 1500, y el otro esclavo entre 1501 y 2000.

2. Conforme cada esclavo vaya obteniendo cada número primo se lo irá enviando al maestro como un dato entero (4 bytes) a través de un cauce sin nombre. Los dos cauces sin nombre necesarios, uno para comunicar cada esclavo con el maestro, los creará el maestro inicialmente. Una vez que un esclavo haya calculado y enviado (uno a uno) al maestro todos los primos en el correspondiente intervalo terminará.

3. El maestro irá recibiendo y mostrando en pantalla (también uno a uno) los números primos calculados por los esclavos en orden creciente.

4. Los esclavos calcularán los números primos de cada subintervalo aplicando el siguiente método concreto: un número n es primo si no es divisible por algún k tal que $2 < k < \text{sqrt}(n)$, donde sqrt corresponde a la función de cálculo de la raíz cuadrada (consulte dicha función en el manual).

ESCLAVO.C

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <math.h> // Incluir en la compilación con gcc -lm

int esPrimo(int n){
    int i;
    int limite = sqrt(n);
    int es_primo = 1;

    for (i = 2; i <= limite && es_primo; i++)
        if (n % i == 0)
            es_primo = 0;

    return es_primo;
};

int main(int argc, char *argv[]){
    int inicio, fin, i;

    inicio = atoi(argv[1]);
    fin = atoi(argv[2]);

    for (i = inicio; i < fin; i++)
        if (esPrimo(i))
            write(STDOUT_FILENO, &i, sizeof(int));

    return 0;
}
```

MAESTRO.C

```
#include<sys/types.h>
#include<sys/stat.h>
#include<unistd.h>
#include<stdio.h>
#include<errno.h>
#include<stdlib.h>
#include<string.h>
#include<fcntl.h>

int main(int argc, char *argv[]){
    int fd1[2];
    int fd2[2];
    int bytesLeidos, bytesLeidos2, val1, val2;
    pid_t esclavo1, esclavo2;
    int intervalos[6];
```

```
    char ini[10];
    char fin[10];

    if (argc < 3){
        perror("Uso: ./maestro <inicio> <fin>\n");
        exit(-1);
    }

    // División de intervalos:
    intervalos[0] = atoi(argv[1]);   // Maestro ini.
    intervalos[1] = atoi(argv[2]);   // Maestro fin.
    intervalos[2] = intervalos[0];   // Esclavo1 ini.
    intervalos[3] = ((intervalos[1]+intervalos[0]) / 2) - 1; // Esclavo1 fin.
    intervalos[4] = intervalos[3] + 1; // Esclavo2 ini.
    intervalos[5] = intervalos[1];     // Esclavo2 fin.

    // Se crean dos cauces.
    pipe(fd1);
    pipe(fd2);

    printf("\nNúmeros primos en el intervalo [%d,%d]:\n", intervalos[0],
intervalos[1]);

    // Primer esclavo
    esclavo1 = fork();
    sprintf(ini, "%d", intervalos[2]);
    sprintf(fin, "%d", intervalos[3]);
    if (esclavo1 == 0){ // Proceso hijo1.
        close(fd1[0]); //Cierro descriptor de lectura
        dup2(fd1[1],STDOUT_FILENO);
        if(execl("./esclavo", "esclavo", ini, fin, NULL) < 0) {
            perror("\nError en el execl");
            exit(-1);
        }
    }else{ // Proceso padre.
        close(fd1[1]);  // Se cierra el descriptor de escritura

        while((bytesLeidos = read(fd1[0],&val1, sizeof(int))) > 0){
            printf("%d ", val1);
        }
        close(fd1[0]);
        printf("\n");
    }

    // Segundo esclavo
    esclavo2 = fork();
    sprintf(ini, "%d", intervalos[4]);
    sprintf(fin, "%d", intervalos[5]);
    if (esclavo2 == 0){
        close(fd2[0]); //Cierro descriptor de lectura
        dup2(fd2[1],STDOUT_FILENO);
        if(execl("./esclavo", "esclavo", ini, fin, NULL) < 0) {
            perror("\nError en el execl");
            exit(-1);
        }
    }else{ // Proceso padre.
```

```
        close(fd2[1]);    // Se cierra el descriptor de escritura.

        while((bytesLeidos2 = read(fd2[0],&val2, sizeof(int))) > 0){
            printf("%d ", val2);
        }
        close(fd2[0]);
        printf("\n");
    }

    return 0;
}
```

Sesion 5

EJERCICIO 1

Compila y ejecuta los siguientes programas y trata de entender su funcionamiento.

```
/*
 envioSignal.c
 Trabajo con llamadas al sistema del Subsistema de Procesos conforme a POSIX 2.10
 Utilización de la llamada kill para enviar una señal:
 0: SIGTERM
 1: SIGUSR1
 2: SIGUSR2
  a un proceso cuyo identificador de proceso es PID.
 SINTAXIS: envioSignal [012] <PID>
*/


#include <sys/types.h> //POSIX Standard: 2.6 Primitive System Data Types
// <sys/types.h>
#include<limits.h> //Incluye <bits/posix1_lim.h> POSIX Standard: 2.9.2 //Minimum
//Values Added to <limits.h> y <bits/posix2_lim.h>
#include <unistd.h> //POSIX Standard: 2.10 Symbolic Constants         <unistd.h>
#include <sys/stat.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <signal.h>
#include <errno.h>
int main(int argc, char *argv[])
 {
          long int pid;
          int signal;
          if(argc<3) {
          printf("\nSintaxis de ejecución: envioSignal [012] <PID>\n\n");
          exit(-1);
           }
         pid= strtol(argv[2],NULL,10);
         if(pid == LONG_MIN || pid == LONG_MAX)
           {
         if(pid == LONG_MIN)
         printf("\nError por desbordamiento inferior LONG_MIN %d",pid);
         else
            printf("\nError por desbordamiento superior LONG_MAX %d",pid);
            perror("\nError en strtol");
            exit(-1);
           }
        signal=atoi(argv[1]);
        switch(signal) {
            case 0: //SIGTERM
            kill(pid,SIGTERM); break;
            case 1: //SIGUSR1
            kill(pid,SIGUSR1); break;
            case 2: //SIGUSR2
            kill(pid,SIGUSR2); break;
            default : // not in [012]
            printf("\n No puedo enviar ese tipo de señal");
              }
}
```

```
/*
 reciboSignal.c
 Trabajo con llamadas al sistema del Subsistema de Procesos conforme a POSIX 2.10
 Utilización de la llamada sigaction para cambiar el comportamiento del proceso
 frente a la recepción de una señal.
*/

#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
#include <errno.h>
static void sig_USR_hdlr(int sigNum)
  {

    if(sigNum == SIGUSR1)
    printf("\nRecibida la señal SIGUSR1\n\n");
    else if(sigNum == SIGUSR2)
    printf("\nRecibida la señal SIGUSR2\n\n");
   }
```

```
int main(int argc, char *argv[])
   {
    struct sigaction sig_USR_nact;
    if(setvbuf(stdout,NULL,_IONBF,0))
       {
    perror("\nError en setvbuf");
       }

//Inicializar la estructura sig_USR_na para especificar la nueva acción para la
señal.

sig_USR_nact.sa_handler= sig_USR_hdlr;


//'sigemptyset' inicia el conjunto de señales dado al conjunto vacío.

sigemptyset (&sig_USR_nact.sa_mask);
sig_USR_nact.sa_flags = 0;

//Establecer mi manejador particular de señal para SIGUSR1
if( sigaction(SIGUSR1,&sig_USR_nact,NULL) <0)
    {
perror("\nError al intentar establecer el manejador de señal para SIGUSR1");
exit(-1);
    }
//Establecer mi manejador particular de señal para SIGUSR2
if( sigaction(SIGUSR2,&sig_USR_nact,NULL) <0)
    {
perror("\nError al intentar establecer el manejador de señal para SIGUSR2");
exit(-1);
    }
for(;;)
   {
   }
}
```

EJERCICIO 2

Escribe un programa en C llamado `contador`, tal que, cada vez que reciba una señal que se puede manejar, muestre por pantalla la señal y el número de veces que se ha recibido ese tipo de señal. En el cuadro siguiente se muestra un ejemplo de ejecución del programa.

```
kawtar@kawtar-VirtualBox:~$ ./contador &
[2] 1899
kawtar@kawtar-VirtualBox:~$
No puedo capturar la señal 9
No puedo capturar la señal 19
No puedo capturar la señal 32
No puedo capturar la señal 33
kill -SIGINT 1899
kawtar@kawtar-VirtualBox:~$ La señal 2 se ha recibido 1 veces
kill -SIGINT 1899
La señal 2 se ha recibido 2 veces
kill -34 1899
kawtar@kawtar-VirtualBox:~$ La señal 34 se ha recibido 1 veces
kill -111 1899
```

```
bash: kill: 111: especificación de señal inválida
kawtar@kawtar-VirtualBox:~$ kill -19 1899 // el programa no puede capturar la
señal 19
[2]+ Detenido            ./contador
kawtar@kawtar-VirtualBox:~$ kill -cont 1899
La señal 18 se ha recibido 1 veces
kawtar@kawtar-VirtualBox:~$ kill -KILL 1899
[2]+  Terminado (killed)    ./contador
```

Contador.c

```
#include <stdio.h>
#include <signal.h>

static int j;
static int contadores[31];
static void handler (int i){
            contadores[i]++;
            printf("\n La señal %d , se ha realizado %d veces. " , i ,
contadores[i]);
}

int main()
{
struct sigaction sa;
sa.sa_handler = handler; // ignora la señal
sigemptyset(&sa.sa_mask);


//Reiniciar las funciones que hayan sido interrumpidas por un manejador
sa.sa_flags = SA_RESTART;
      int contadores[31];

      for ( j = 1 ; j<=31 ; j++)
            contadores[j] = 0;

      int i;
      for ( i = 1 ; i<=60 ; i++){
            if (sigaction(i, &sa, NULL) == -1)
      {
            printf("error en el manejador");}}while(1);


}
```

EJERCICIO 3

Escribe un programa que suspenda la ejecución del proceso actual hasta que se reciba la señal `SIGUSR1`.

```
#include <stdio.h>
#include <signal.h>
```

```
int main()

{
sigset_t new_mask;

/* inicializar la nueva mascara de señales */
sigemptyset(&new_mask);

sigfillset(&new_mask);

sigdelset(&new_mask , SIGUSR1);
// esperar a cualquier señal menos a SIGUSR1
sigsuspend(&new_mask);

}
```

EJERCICIO 4

Compila y ejecuta el siguiente programa y trata de entender su funcionamiento.

```
//tarea12.c

#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>

static int signal_recibida = 0;

static void manejador (int sig)
{
          signal_recibida = 1;
}

int main (int argc, char *argv[])
{
    sigset_t conjunto_mascaras;
    sigset_t conj_mascaras_original;
    struct sigaction act;

    //Iniciamos a 0 todos los elementos de la estructura act
    memset (&act, 0, sizeof(act));

    act.sa_handler = manejador;

    if (sigaction(SIGTERM, &act, 0)) {
        perror ("sigaction");
        return 1;
    }

    //Iniciamos un nuevo conjunto de mascaras
    sigemptyset (&conjunto_mascaras);
    //Añadimos SIGTERM al conjunto de mascaras
    sigaddset (&conjunto_mascaras, SIGTERM);
```

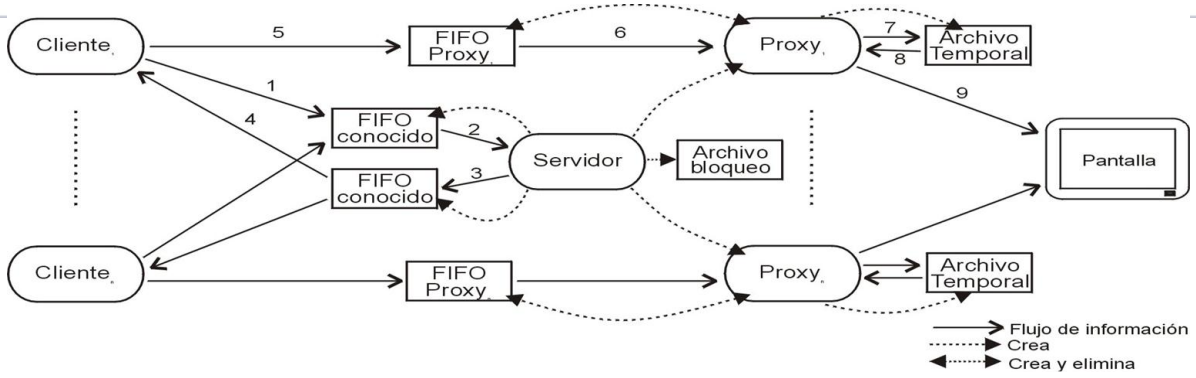
```
    //Bloqueamos SIGTERM
    if (sigprocmask(SIG_BLOCK, &conjunto_mascaras, &conj_mascaras_original) < 0) {
       perror ("primer sigprocmask");
       return 1;
    }

    sleep (10);

    //Restauramos la señal – desbloqueamos SIGTERM
    if (sigprocmask(SIG_SETMASK, &conj_mascaras_original, NULL) < 0) {
       perror ("segundo sigprocmask");
       return 1;
       }

    sleep (1);

   if (signal_recibida)
       printf ("\nSenal recibida\n");
   return 0;
}
```

- El programa crea una máscara, en la que solo añade la señal SIGTERM. Aplica la máscara y con ello bloquea la señal SIGTERM. Realiza un sleep de 10s, y si durante este tiempo le mandamos dicha señal no reaccionará porque está bloqueada. (la variable signal_recibida se activará) Una vez termina de "dormir", desbloquea la señal SIGTERM reanudando la máscara antigua y comprueba si la hemos introducido comprobando si la variable signal_recibida está activada , si es así nos muestra que la señal ya ha sido recibida , si no acabará el programa sin mostrar nada.

Sesión 6

Esta práctica tratará aspectos relacionados con procesos, señales y cauces con nombre o archivos FIFO.

Se pretende construir un *spool* concurrente de impresión en pantalla. Su funcionamiento general consiste en controlar el acceso de procesos clientes al recurso compartido, en este caso la pantalla, garantizando la utilización en exclusión mutua de dicho recurso. Recuerde que un sistema *spool* para impresión imprime un documento sólo cuando éste se ha generado por completo. De esta forma, se consigue que un proceso no pueda apropiarse indefinidamente, o durante mucho tiempo si el proceso es lento, del recurso compartido. Como mecanismo de comunicación/sincronización entre procesos se van a utilizar *pipes* con nombre (archivos FIFO) y en algún caso señales. En la siguiente figura se muestra el esquema general a seguir para la implementación:



Siguiendo los mensajes numerados obtendremos las interacciones entre procesos para poder llevar a cabo la impresión de un archivo, según se explica a continuación:

1. Un cliente solicita la impresión de un archivo enviando un mensaje (cuyo contenido no tiene importancia) al servidor a través de un FIFO cuyo nombre es conocido.

2. El servidor lee esta petición delegando la recepción e impresión del documento en un proceso (*proxy*) que crea específicamente para atender a dicho cliente. Una vez servida dicha petición, el *proxy* terminará.

3. El servidor responde al cliente a través de otro FIFO de nombre conocido, informando de la identidad (PID) del *proxy* que va a atender su petición. Este dato es la base para poder comunicar al cliente con el *proxy*, ya que éste creará un nuevo archivo FIFO especifico para esta comunicación, cuyo nombre puede ser el propio PID del *proxy*. El *proxy* se encargará de eliminar dicho FIFO cuando ya no sea necesario (justo antes de terminar su ejecución).

4. El cliente lee esta información que le envía el servidor, de manera que así sabrá donde enviar los datos a imprimir.

5. Probablemente el cliente necesitará enviar varios mensajes como éste, tantos como sean necesarios para transmitir toda la información a imprimir. El final de la transmisión de la información lo indicará con un fin de archivo.

6. El *proxy* obtendrá la información a imprimir llevando a cabo probablemente varias lecturas como ésta del FIFO.

7. Por cada lectura anterior, tendrá lugar una escritura de dicha información en un archivo temporal, creado específicamente por el *proxy* para almacenar completamente el documento a imprimir.

8. Una vez recogido todo el documento, volverá a leerlo del archivo temporal justo después de comprobar que puede disponer de la pantalla para iniciar la impresión en exclusión mutua.

9. Cada lectura de datos realizada en el paso anterior implicará su escritura en pantalla.

Tenga en cuenta las siguientes consideraciones:

- Utilice un tamaño de 1024 bytes para las operaciones de lectura/escritura (mediante las llamadas al sistema *read*/*write*) de los datos del archivo a imprimir.

- Recuerde que cuando todos los procesos que tienen abierto un FIFO para escritura lo cierran, o dichos procesos terminan, entonces se genera automáticamente un fin de archivo que producirá el desbloqueo del proceso que esté bloqueado esperando leer de dicho FIFO, devolviendo en este caso la llamada al sistema *read* la cantidad de 0 Bytes leídos. Si en algún caso, como por ejemplo en el servidor que lee del FIFO conocido no interesa este comportamiento, entonces la solución más directa es abrir el FIFO en modo lectura/escritura (*O_RDWR*) por parte del proceso servidor. Así nos aseguramos que siempre al menos un proceso va a tener el FIFO abierto en escritura, y por tanto se evitan la generación de varios fin de archivo.

- Siempre que cree un archivo, basta con que especifique en el campo de modo la constante S_IRWXU para que el propietario tenga todos los permisos (lectura, escritura, ejecución) sobre el archivo. Por supuesto, si el sistema se utilizara en una situación real habría que ampliar estos permisos a otros usuarios.

- Utilice la función *tmpfile* incluida en la biblioteca estándar para el archivo temporal que crea cada *proxy*, así su eliminación será automática. Tenga en cuenta que esta función devuelve un puntero a la estructura *FILE* (*FILE* *), y por tanto las lecturas y escrituras se realizarán con las funciones de la biblioteca estándar *fread* y *fwrite* respectivamente.

- No deben quedar procesos *zombis* en el sistema. Podemos evitarlo atrapando en el servidor las señales *SIGCHLD* que envían los procesos *proxy* (ya que son hijos del servidor) cuando terminan. Por omisión, la acción asignada a esta señal es ignorarla, pero mediante la llamada al sistema *signal* podemos especificar un manejador que ejecute la llamada *wait* impidiendo que los procesos *proxy* queden como *zombis*.

- Por cuestiones de reusabilidad, el programa *proxy* leerá de su entrada estándar (constante *STDIN_FILENO*) y escribirá en su salida estándar (constante *STDOUT_FILENO*). Por tanto, hay que redireccionar su entrada estándar al archivo FIFO correspondiente, esto se puede llevar a cabo mediante la llamada al sistema *dup2*.

- Para conseguir el bloqueo/desbloqueo de pantalla a la hora de imprimir, utilice la función que se muestra a continuación. Esta recibe como parámetros un descriptor del archivo que se utiliza para bloqueo (por tanto ya se ha abierto previamente) y la orden a ejecutar, es decir, bloquear (*F_WRLCK*) o desbloquear (*F_UNLCK*):

```
void bloqueodesbloqueo (int dbloqueo, int orden) {
struct flock cerrojo;
// Inicializamos el cerrojo para bloquear todo el archivo
cerrojo.l_type= orden;
cerrojo.l_whence= SEEK_SET;
cerrojo.l_start= 0;
cerrojo.l_len = 0;

//Si vamos a bloquearlo y ya lo esta, entonces el proceso
duerme
if (fcntl(dbloqueo, F_SETLKW, &cerrojo) == -1) {
  perror ("Proxy: problemas al bloquear para impresion");
  exit(1);
}
}
```

También se proporciona un programa denominado *clientes.c* capaz de lanzar hasta 10 clientes solicitando la impresión de datos. Para simplificar y facilitar la comprobación del funcionamiento de todo el sistema, cada uno de los clientes pretende imprimir un archivo de tamaño desconocido pero con todos los caracteres idénticos, es decir, un cliente imprimirá sólo caracteres *a*, otro sólo caracteres *b*, y así sucesivamente. El formato de ejecución de este programa es:

prompt> **clientes** <nombre_fifos_conocidos> <número_clientes>

El argumento <nombre_fifos_conocidos> es un único nombre, de forma que los clientes suponen que el nombre del FIFO conocido de entrada al servidor es dicho nombre concatenado con el carácter "e". En el caso del Fifo de salida, se concatena dicho nombre con el carácter "s".

Implemente el resto de programas según lo descrito, para ello necesitará además de las funciones y llamadas al sistema comentadas anteriormente, otras como: *mkfifo* (crea un archivo FIFO), *creat* (crea un archivo normal), *unlink* (borra un archivo de cualquier tipo).

Tenga en cuenta que el servidor debe ser un proceso que está permanentemente ejecutándose, por tanto, tendrás que ejecutarlo en background y siempre antes de lanzar los clientes. Asegúrate también de que el servidor cree los archivos FIFO conocidos antes de que los clientes intenten comunicarse con él.

PROXY.C

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
```

```
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<errno.h>

#define BLOCK_FILE "blockfile"

void bloqueodesbloqueo (int dbloqueo, int orden) {
    struct flock cerrojo;
    // Inicializamos el cerrojo para bloquear todo el archivo
    cerrojo.l_type= orden;
    cerrojo.l_whence= SEEK_SET;
    cerrojo.l_start= 0;
    cerrojo.l_len = 0;
    //Si vamos a bloquearlo y ya lo esta, entonces el proceso duerme
    if (fcntl(dbloqueo, F_SETLKW, &cerrojo) == -1) {
        perror ("Proxy: problemas al bloquear para impresion");
        exit(1);
    }
}

int main(int argc, char *argv[]){
    char buffer[1024];
    int nbytes, dbloqueo;
    FILE *tmp = tmpfile();
    char fifoproxy[256];

    // Leer bloques de 1024 bytes y escribirlos en archivo temporal.
    while((nbytes = read(STDIN_FILENO, buffer, 1024)) > 0){
        fwrite(buffer, sizeof(char), nbytes, tmp);
    }

    // Abrir archivo "cerrojo".
    if ((dbloqueo = open(BLOCK_FILE, O_RDWR)) == -1)
        printf("Error al abrir blockfile\n");

    // Bloquar cerrojo.
    bloqueodesbloqueo(dbloqueo, F_WRLCK);
    // Inicio de E.M.
    // Leer fichero temporal hasta fin de archivo.
    while(!feof(tmp)){
        fread(buffer, sizeof(char), 1024, tmp);
        // Escribir en la salida estandar.
        write(STDOUT_FILENO, buffer, 1024);
    }
    // Fin de E.M.
    // Desbloquear cerrojo.
    bloqueodesbloqueo(dbloqueo, F_UNLCK);

    // Eliminar su fifo antes de terminar.
    sprintf(fifoproxy,"fifo.%d", getpid());
    unlink(fifoproxy);

    exit(0);
}
```

SERVIDOR.C

```
#include<sys/types.h>
#include<sys/stat.h>
#include<fcntl.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<errno.h>
#include <signal.h>

#define tamano 1024
#define longnombre 50
#define MAX_CLIENTS 100

static void signal_handler(int sigNum){
    pid_t pid;
    char fifoproxy[longnombre];

    // Capturar pid del hijo y eliminar su archivo fifo asociado.
    pid = wait(NULL);
}


int main(int argc, char *argv[]){
    int fde, fds, fdc, tmp_cli;
    char nombrefifoe[longnombre];
    char nombrefifos[longnombre];
    char fifoproxy[longnombre];
    int leidos, proxypid;
    pid_t pid;


    //Comprobar el uso correcto del programa
    if(argc != 2) {
        printf("Uso: Servidor <nombre_fifo>");
        exit(-1);
    }

    // Manejador de señal SIGCHLD.
    signal(SIGCHLD, signal_handler);

    // Componer nombre del archivo fifo
    sprintf(nombrefifos,"%ss",argv[1]);
    sprintf(nombrefifoe,"%se",argv[1]);

    // Crear archivos fifo
    umask(0);
    mkfifo(nombrefifoe,0666);
    mkfifo(nombrefifos,0666);
```

```
    //Intentar abrir los archivos fifo.
    if((fds=open(nombrefifos,O_RDWR)) < 0){
        perror("\nNo se pudo abrir el archivo fifo del servidor.");
        exit(-1);
    }

    if((fde=open(nombrefifoe,O_RDWR)) < 0){
        perror("\nNo se pudo abrir el archivo fifo del cliente.");
        exit(-1);
    }

    // Creación del archivo bloqueo que se utilizará como cerrojo.
    umask(0);
    if((fdc=open("blockfile", O_CREAT, 0666))<0){
        printf("\nError al crear el archivo de bloqueo.");
        exit(-1);
    }

    // Lanzar procesos mientras se reciba algo del archivo fifo cliente.
    while((leidos=read(fde, &tmp_cli, sizeof(int))) != 0){
        pid = fork();
        if (pid == 0){ //Proceso hijo.
            // Obtener pid del nuevo proceso
            proxypid = getpid();
            // Construir cadena "proxy.getpid()" para archivo fifo.
            sprintf(fifoproxy,"fifo.%d", proxypid);
            // Crear archivo fifo.
            umask(0);
            mkfifo(fifoproxy,0666);
            // Escribe el pid del proxy en fifos
            write(fds, &proxypid, sizeof(int));
            // Abrir fifo para lectura.
            int fifo = open(fifoproxy,O_RDONLY);
            // Redirigir entrada standar -> archivofifo
            dup2(fifo, STDIN_FILENO);
            execlp("./proxy", "proxy", NULL);
            exit(0);
        }
    }

    return 0;
}
```