Prácticas de Arquitectura de Ordenadores

Joaquín Seoane

Mónica Cortés

Prácticas de Arquitectura de Ordenadores
por Joaquín Seoane y Mónica Cortés

Fecha de publicación 16 de Octubre de 2011

Índice de prácticas

Lista de figuras

Introducción

Esta es la documentación contiene información sobre las prácticas de la asignatura *Arquitectura de Ordenadores* para el curso *2011-2012*. Reúne contribuciones de los distintos profeserores presentes y pasados de la asignatura, entre los que se cuentan Alejandro Alonso, Pedro Alonso, Angel Álvarez, Juan Carlos Dueñas, Gregorio Fernández, Joaquín Seoane, David Larrabeiti, Tomás de Miguel, Marifeli Sedano, Mercedes Garijo, Juan Antonio de la Puente, Mónica Cortés, Antonio José Gómez Flechoso, Jose María del Álamo y Amalio Nieto.

Para cada práctica se dan los objetivos, la especificación y quizá ejemplos y programas de prueba. Así mismo se proporcionan soluciones compiladas en forma de objetos y/o ejecutables. De este modo el alumno puede ver el comportamiento de una solución que cumple el enunciado y ejercitar la suya con un programa de prueba funcional. En la versión HTML de este documento, los ficheros ofrecidos, en fuente y objeto, se pueden obtener directamente con el navegador.

Práctica 1. Acceso y uso básico del sistema

1.1. Objetivos

Familiarizarse con un sistema GNU/Linux y, por similitud, con cualquier sistema basado en Unix, al nivel de usuario, tanto en modo texto como gráfico (entrada, órdenes, documentación, y algo del intérprete de órdenes).

1.2. Especificación

Lo que sigue es una guía de acciones sugeridas. Puede desviarse de este esquema lo que crea necesario para comprender mejor el sistema. Emplee el tiempo necesario en esta práctica, de modo que se sienta cómodo con el entorno. El entorno gráfico en el laboratorio está configurado de un manera que puede ser distinta de la que ofrezcan por omisión distintas distribuciones de Linux. Lo que en adelante se explica se refiere a la configuración del Laboratorio.

1.3. Arranque del sistema

1. Encienda la máquina.
2. Arránquela en Linux (puede que al arrancar le salga un menú para elegir el sistema operativo; si no lo tiene, la máquina arrancará en modo Linux directamente).

1.4. Iniciación en modo texto

La estación arrancará normalmente en modo gráfico, al disponer de hardware y software apropiados. Sin embargo, en sistemas pequeños, muchos de los cuales se utilizan en sistemas de telecomunicaciones, carecen de modo gráfico o no funciona. Para poderlos controlar es necesario poder trabajar en modo texto.

1. La estación arrancará en modo gráfico. Sin embargo, para entender mejor es sistema, es mejor empezar en modo texto, aunque sólo sea inicialmente. Para ello pulse **Control**-**Alt**-**F1**.
2. Autentifíquese, dando su nombre de usuario (`login`) y contraseña (`password`).
3. Ejecute algunas órdenes informativas sin parámetros (por ejemplo, **date**, para ver fecha y hora, **cal**, para ver el calendario del mes en curso, **pwd**, para ver en qué directorio estamos).
4. Ejecute algunas órdenes informativas con opciones y parámetros (por ejemplo, **date `-u`**, para ver fecha y hora universal, o **cal *`2007`***, para ver el calendario de 2007).
5. Familiarícese con el manual en línea tradicional. Mire la página de manual de **cal** (con **man *`cal`***) y ejercite algunos de los parámetros y opciones que se documentan. Aprenda a seleccionar secciones del manual donde buscar algo que posiblemente esté en varias (recuerde que la sección 1 es para programas, la 2 para llamadas al sistema, la 3 para otras rutinas, etc). Por ejemplo, haga **man *`chmod`***, **man `-f` *`chmod`***, **man *`2 chmod`*** y **man *`1 chmod`*** . Controle la salida con la tecla de espaciado para avanza y la letra b para retroceder. Salga con **q**.
6. Puede familiarizarse con el manual en hipertexto **info** con **man *`info`*** e **info *`info`***. Muévase por las páginas del **info** con las teclas de cursor o la barra espaciadora. Entre en elementos de los menús con el cursor seguido de retorno de carro o con **m***`nombre`* (completando posiblemente el

nombre con el tabulador). Muévase por los nodos al menos con **p** (previo), **n** (siguiente) o **u** (arriba). Salga con **q**. Aproveche para echar un ojo a la documentación de las órdenes de manipulación de ficheros de GNU con **info *coreutils*** y busque la que se refiera a **chmod**.

7. Verifique que se puede navegar por la Web en modo texto con **lynx** con parámetro http://www.lab.dit.upm.es
   - Muévase por los enlaces con las teclas de cursor (arriba y abajo).
   - Visite páginas pulsando retorno de carro con el enlace seleccionado
   - Navege por la historia de páginas visitadas con las teclas de cursor izquierda y derecha.
   - Si le parece interesante entre en la documentación con ?.
   - Termine con q.
8. Desconéctese del modo texto (logout) ejecutando la orden **exit**.

1.5. Continuando en un entorno gráfico

Vamos a seguir practicando el modo texto, pero es más cómodo y vistoso en modo gráfico. Hay varios entornos gráficos, unos más avanzados (por ejemplo, `KDE` y `Gnome`), otros más sencillos, pero menos consumidores de memoria y procesador, siendo más aptos para las máquinas menos potentes (por ejemplo `LXDE` o `Xfce`). Aquí se hace énfasis en Gnome, porque es el que conocen la mayoría de los alumnos, aunque trataremos de no cerrar alternativas.

Todos los entornos gráficos disponibles están basados en un *sistema de ventanas X*, y sobre él se ejecutan un *gestor de ventanas* y, generalmente un *entorno de escritorio*, además de las aplicaciones gráficas.

1. Entre en modo gráfico con **Control**-**Alt**-**F7** (en otras versiones de Linux puede ser otra tecla de función, ya que esto es configurable; suele haber varias consolas de texto, seleccionables con **Control**-**Alt**-**F1**, **Control**-**Alt**-**F2**, **Control**-**Alt**-**F3**, etc.; al final de ellas están las consolas gráficas, normalmente una).
2. Identifíquese y autentifique su identidad con la contraseña.
3. Arranque un pseudoterminal (seleccionando del menú `Aplicaciones -> Herramientas del Sistema -> Terminal`). También puede encontrar en el escritorio algún icono con apariencia de televisor para lanzarlo. El pseudoterminal se llama **gnome-terminal**[1] y pruebe algunas órdenes de unix de las que ya probó en modo texto (por ejemplo, **date, cal,...**).
4. Verifique que puede lanzar del mismo modo aplicaciones gráficas desde el pseudoterminal, dando su nombre. Pruebe por ejemplo un calculador (**gnome-calculator** en GNOME o **kcalc** en KDE), o incluso otro pseudoterminal (**gnome-terminal** en GNOME o **konsole** en KDE). Observe que, en general, se puede lanzar una aplicación de un entorno gráfico en el otro, si bien se puede perder alguna funcionalidad y eficiencia. Puede terminar la aplicación gráfica por su método natural, generalmente en el menú de archivo; por medio del gestor de ventanas, generalmente un aspa en una esquina de la ventana; o pulsando la combinación de teclas **Control**-**C** en el pseudoterminal, que mandará una señal de terminación al proceso en ejecución.
5. Determine en qué directorio está, con **pwd**, vea que directorios ordinarios hay con, **ls `-F`**. Navegue por los directorios con **cd**, **cd *`directorio`***, **cd `.`**, **cd `..`**, etc, viendo qué ficheros hay en ellos, con **ls**, de qué tipo son, con **ls `-F`**, intentando averiguarlos tipos de los datos contenidos con **file**. Haga esa navegación con rutas absolutas y relativas.
6. Repita lo anterior con una navegador gráfico, como **nautilus**, bien lanzado desde la consola, bien desde el menú `Lugares`. Observe que este programa también utiliza sus heurísticos para determinar los tipos de ficheros, mostrando distintos iconos según el tipo.

7. Examine ficheros de texto, con **cat** ***fichero***. Por ejemplo, con **cat** ***/usr/share/dict/words*** vuelque en pantalla el diccionario de palabras inglesas. Puede usar la barra de *scroll* (desplazamiento) para ver más texto del que cabe en la ventana.
8. Examine detalladamente ficheros de texto con un paginador. Por ejemplo, puede ver el diccionario de palabras inglesas con **less** ***/usr/share/dict/words***, moviéndose por el fichero con las teclas de cursor y **AvPag** y **RePag**. Puede pulsar **h** para obtener ayuda, y **q** para terminar).
9. Copie `/etc/passwd` en su cuenta con **cp**. Cambie de nombre su copia del fichero con **mv** y llámelo `passwd-copia`. Aprenda a comprimir y descomprimir ficheros con **gzip** y **gunzip** sobre su copia. Compruebe la reducción de tamaño con **ls -l**. Haga lo mismo con **bzip2** y **bunzip2** y vea cual da mejor resultado (en tiempo de compresión y espacio ocupado).
10. Invoque la ayuda del sistema por medio del icono en forma de salvavidas. Con ella puede acceder en forma de hipertexto web a las páginas de manual (**man**) de unix y al (**info**).
11. *Cambie su contraseña* inicial por otra más segura (ninguna palabra de ningún idioma común, mezcle números, letras mayúsculas y minúsculas y caracteres especiales). Para ello debe acceder al servicio de contraseñas vía Web[2] .
12. Observe que hay un directorio en su cuenta con un significado especial. Se llama *Escritorio* y si copiamos un fichero en él, se verá en el escritorio en forma de icono.
13. Salga del sistema de ventanas con la opción de terminar del menú.

1.6. Parada del sistema

Todo sistema operativo moderno requiere ser informado de que se le va a apagar, antes de hacerlo. De este modo se vuelcan a disco todas las informaciones que aún estan en memoria y se paran todos los programas ordenadamente, para lo cual suele haber alguna operación de apagar (en nuestro caso del gestor de pantallas). En modo no gráfico se hace pulsando **Control**-**Alt**-**Suprimir**. En modo gráfico, el gestor de pantalla que le permite identificarse y autentificarse, también le permite dar la orden de apagado.

En el caso del laboratorio, como el disco se regenera al arrancar el sistema y los datos de las cuentas van a un servidor remoto, se puede apagar directamente. No obstante, conviene esperar unos segundos desde la última escritura, para dar tiempo a que se propague al servidor.

Notas

1. O **konsole** en `KDE`.
2. También puede hacerlo con la orden **smbpasswd -r** ***ldap-ng.lab.dit.upm.es***.

Práctica 2. Introducción al sistema de ficheros

2.1. Objetivos

Familiarizarse con un mínimo del sistema de ficheros a nivel de usuario para continuar con las prácticas.

2.2. Especificación

Lo que sigue es una guía de acciones sugeridas. Puede desviarse de este esquema lo que crea necesario para comprender mejor el sistema.

2.3. Navegación

1. Visite con **cd `directorio`** y liste con **ls `-F directorio`** los directorios siguientes: `/etc`, `/bin`, `/usr/bin`, `/sbin`, `/usr/sbin`, `/usr/share`, `/usr/share/doc`, `/usr/share/man/man1`, `/usr/share/man/es/man1`, `/boot`, `/home/`, etc. Fíjese que detrás del nombre de fichero puede venir un carácter que indica si es directorio, ejecutable ordinario, programilla de shell, etc (mire el manual de **ls**).
2. Entre en su cuenta con **cd** (sin argumentos) y vea qué ficheros y directorios hay, incluso los ocultos (que empiezan por `.`), con **ls `-aF`**.

2.4. Detalles de los ficheros

1. Use **ls `-l`** o **ls `-ld`** para ver detalles de ficheros y directorios. Determine, para ellos el tipo de objeto (al menos fichero, directorio, enlace simbólico mirando el primer carácter de cada línea: `-`, `d`, `l`, ...). Por ejemplo:

```
ls -ld
/
/etc
/etc/hosts
/etc/motd
/etc/printcap
/bin/bash
/bin/sh
~
/tmp
/usr/bin/passwd
```

2. Determine, para ellos el usuario propietario, el grupo propietario, los permisos del propietario, grupo y los demás (`rwx`). Observe la `t` de `/tmp` y la `s` de `/usr/bin/passwd`.
3. Determine la longitud y fecha y hora de última modificación de cada uno de ellos.

4. Determine el uso de disco de cada uno de ellos con **du**.
5. Use **stat** para todo lo anterior.
6. Para los enlaces simbólicos, determine los datos anteriores para el fichero original.

2.5. Dispositivos

1. Liste, con **ls -l**, lo que hay en `/dev` y vea que la mayoría de los objetos son dispositivos o seudodispositivos de bloques (`b`) o de caracteres (`c`).
2. Localice el tipo y los números mayor y menor para algunos dispositivos, como `/dev/psaux` (ratón), `/dev/ttyS0` y `/dev/ttyS1` (líneas serie), `/dev/tty00`, `/dev/tty01`, etc (seudoterminales de texto), `/dev/pts/0`, `/dev/pts/1`, etc (seudoterminales de gráficos), `/dev/tty` (terminal de control del proceso), `/dev/fd0` y `/dev/fd1` (disqueteras), `/dev/sda` (disco duro), `/dev/sda1`, `/dev/sda2`, etc (particiones de `/dev/sda`), etc.
3. Localice el tipo y los números mayor y menor para algunos seudodispositivos, como `/dev/null` (sumidero vacío), `/dev/zero` (fuente de ceros), `/dev/urandom` (fuente de números aleatorios).

2.6. Seudoficheros de `/proc/`

Mire algunos que caracterizan la máquina:

- **cat `/proc/cpuinfo`**
- **cat `/proc/devices`**
- **cat `/proc/ioports`**
- **cat `/proc/interrupts`**

2.7. Sistemas de ficheros y montaje

Ejecute **mount** y vea los sistemas de ficheros montados.

- ¿Qué discos o particiones locales se montan y donde? ¿de qué tipo son los sistemas de ficheros?
- ¿Qué directorios remotos se montan y donde? ¿con qué protocolo?
- ¿Qué sistemas de ficheros virtuales se montan y donde? Trate de explicar al menos `proc`, `sysfs`, `tmpfs` y `devpts`.

Use **df** para ver espacio ocupado y libre en cada sistema de ficheros.

2.8. Instantáneas

1. Observe que existe un directorio oculto en su cuenta llamado *.snapshot*. Este directorio contiene copias de seguridad del contenido de su cuenta. Se realizan de forma periódica y automática. Visualice el contenido de este directorio con la orden **ls**, verá los periodos configurados para la realización de las copias y el número de copias guardadas.
2. Recupere un fichero de su copia de seguridad. Para ello sólo necesita copiar un fichero de alguno de los directorios bajo *.snapshot* al directorio que desee.

Práctica 3. La shell y herramientas

3.1. Objetivos

Familiarizarse con intérprete de órdenes de Linux al nivel de usuario y algunas herramientas.

3.2. Especificación

Lo que sigue es una guía de acciones sugeridas. Utilice un Terminal de texto dentro del entorno gráfico. Puede desviarse de este esquema lo que crea necesario para comprender mejor el sistema. Emplee el tiempo necesario en esta práctica, de modo que se sienta cómodo con el entorno.

3.3. Comodines y sustituciones

Familiarícese con los comodines y sustituciones de la *shell* con algunas órdenes como

- **echo** `*`
- **echo** `.*`
- **echo** `'*'`
- **echo** `/usr/include/a*h`
- **echo** `/usr/include/[abc]*`
- **echo** `/usr/include/[a-d]*`
- **echo** `$PATH`
- **arch**
- **echo** `tengo un procesador de tipo 'arch'`[1]

3.4. Historia

Familiarícese con los mecanismos de historia y de terminación de nombres de la *shell*:

- **history**
- Pulse las teclas de control de cursor arriba, abajo, izquierda, derecha; inserte caracteres; ejecute.
- Use el tabulador para completar nombres de órdenes o ficheros no ambiguos (si el nombre es ambiguo pita; si se pulsa el tabulador otra vez, lista los nombres que casan con el prefijo).

3.5. Redirecciones

Familiarícese con las redirecciones de la *shell* con algunas órdenes como:

- **cal**
- **cal** `>fichero`
- **cal** `>> fichero`
- **sort** `< /etc/passwd > fichero`
- **cal** | **hexdump -C**
- **man *ls*** | **tr -s ' ' '\n'** | **sort** | **uniq -c** | **sort -n** | **tail -5**
- **man *ls*** | **tr -s ' ' '\n'** | **sort** | **uniq -c** | **tee *espia*** | **sort -n** | **tail -5**

Si no conoce alguna de las órdenes anteriores, mire su página de manual.

3.6. Seudoficheros de `/dev/`

- **cat /dev/null**
- **echo *hola*** `> /dev/null.`
- **cat /dev/null**
- **wc /dev/null**
- **wc */etc/**** `> /dev/null`
- **wc */etc/**** `> /dev/null 2>&1`
- **hexdump -Cv /dev/null**
- **hexdump -Cv /dev/urandom**
- **hexdump -Cv /dev/zero**
- **hexdump -Cv /dev/zero** | **head -100**
- **echo *hola*** `> /dev/tty.`
- Si **tty** da `/dev/pts/3`, hacer **echo *hola*** `> /dev/pts/3` desde el mismo terminal y desde otro.

3.7. Variables y entorno

Familiarícese con las variables de la shell que están en el entorno:

- Vea lo que valen las variables de entorno `PATH`, `LANG` y `PAGER`. Ojo, con **echo *$VARIABLE*** no se sabe si una variable está en el entorno o no, y si el valor que da es vacío, no se sabe si está definida o no. Para asegurarse haga **printenv | grep *VARIABLE***; si no está en el entorno, expórtela con **export *VARIABLE***.

- Vea lo que muestran **date** y **man *ls*** antes y después de cambiar `LANG` al valor `en_US` (inglés/Estados Unidos). Luego restáurela a su valor original.
- Cambie la variable `PAGER` a `/bin/cat` e intente hacer un **man *ls***. Explique el resultado y luego restaure la variable a un paginador de verdad (**more** o **less**).
- Vea donde la shell encuentra algunos programas con **which**. Cambie la variable `PATH` a `/bin` y vea si se encuentra esos programas.

3.8. Control de procesos

Familiarícese con el control de procesos de la *shell*.

- Lanzando un proceso en *segundo plano (background)*: **wc */usr/bin/**** `> /dev/null 2>&1 &`
- Viendo los procesos controlados por el terminal con **ps**. Observando los distintos campos, como PID, PPID, STAT y TTY.
- Viendo los procesos padre de los procesos de usuario **ps fjx** (columna PPID).
- Usando el programa **top** para ver todos los procesos del sistema ordenados por uso de procesador o de memoria.
- Viendo los datos de los procesos como ficheros en los directorios `/proc/nº`, done `nº` es un número de proceso. En `/proc/self` se observan los del proceso en curso.
  - **cat /proc/1/cmdline**
  - **hexdump -C /proc/1/cmdline**
  - **hexdump -C /proc/self/cmdline**
  - Averigüe el nº de proceso de su shell con **ps** y haga
    - **hexdump -C /proc/nº/cmdline**
    - **hexdump -C /proc/nº/environ**,
    - **cat /proc/nº/status**
    - **ls -l /proc/nº/exe**
    - **ls -l /proc/nº/cwd**
    - **ls -l /proc/nº/exe**
    - etc.

- Suspendiendo trabajos con **Control**-**Z**.
- Viendo los trabajos que hay asociados a la sesión con **jobs**.
- Pasando trabajos de suspendidos a activos *en segundo plano* con **bg** y a *primer plano* con **fg**.

3.9. Señales en la shell de Unix

Familiarícese con el uso de señales desde la *shell*, siguiendo, por ejemplo, esta secuencia de actividades:

- Lance un proceso largo en *segundo plano (background)*. Por ejemplo:

  wc ***/*/*/*** `>/dev/null  2>&1 &`

- Vea los procesos controlados por el terminal, con **ps**, y los trabajos, con **jobs**. Ahí estará un proceso ejecutando el **wc**, y el trabajo correspondiente.
- Añada un trabajo multiproceso como este, que calcula el número de veces que aparece cada palabra en el Quijote[2]:

  wget -O - -q --limit-rate=10k
  http://www.gutenberg.org/ebooks/2000.txt.utf8 | **tr -sc** ***'[:alpha:]' '\n'*** |
  sort | **uniq -c** | **sort -n** `> palabras &`

  y vea los trabajos y los procesos que hay.
- Mande la señal de terminar un trabajo y vea qué sucede. Por ejemplo:

  kill *%1*

  si 1 fuera el primer trabajo.
- Mande la señal de terminar un proceso del otro trabajo y vea qué sucede. Por ejemplo:

  kill ***5431***

  si 5431 fuera el número de proceso de **uniq**, por ejemplo.
- Observe que la orden **kill** sin opciones manda la señal de terminar (`15` o `SIGKILL`), como en:

  kill -15 ***5431***

  o

  kill -SIGKILL ***5431***
- Mire las señales disponibles y sus números con **kill -l**.
- Pulsando **CONTROL-C**, se manda la señal `SIGINT` a los procesos ligados al terminal. Por defecto abortan, pero pueden ignorarlo o hacer algo.
  - Observe que la **bash** lo ignora.
  - Observe que **cat** aborta.
  - Observe que la **bc** la atrapa, diciendo las instrucciones para terminar.
  - Haga que la **bash** la atrape y diga algo. Por ejemplo:

    trap "echo; echo Hola $USER" INT

    Observe que la señal se nombra numéricamente o simbólicamente, si el prefijo SIG.

3.10. Edición de textos y programillas de shell

- Familiarícese brevemente con un editor de textos, si no lo estuviera ya. Para ello use el editor de textos de su elección. Uno sencillo es **gedit** de Gnome o **kwrite** o **kate** de KDE, que colorean sintácticamente programas en distintos lenguajes de programación. También han disponibles editores que no requieren entorno gráfico como **nano** o **pico**. [3]
- Edite un fichero `p1.sh` que contenga este programilla de shell:

```
echo Mirando los ficheros de `pwd`
echo Encontrados *
```

  Ejecútelo con **bash** **`p1.sh`** y observe el resultado.

- Aproveche que el núcleo de Linux usa **bash** para ejecutar ficheros ejecutables con texto, directamente. Use **chmod +x p1.sh** para hacerlo ejecutable y ejecútelo como cualquier programa.
- Observe que la shell es un intérprete de programas (créelo como `p2.sh` y pruebe):

```
echo Mirando los ficheros de `pwd`
for i in *
do
  echo Encontrado $i en `pwd`
done
```

- Observe que también puede recibir argumentos (créelo como `p3.sh` y pruebe):

```
echo Mirando los ficheros de que me pasan
for i in $*
do
  echo Encontrado $i
done
```

- Busque en el sistema alguna utilidad que le permita visualizar las diferencias entre ficheros con **man -k *compar*** . Visualize con **man** la información del manual de las órdenes sugeridos para ver cómo se usan. Recuerde que las órdenes están en la sección 1 del manual. Use una de las órdenes para visualizar las diferencias entre `p3.sh`y `p3.sh`. Pruebe al menos **diff** y **meld**.
- Use al menos **diff3** y **meld** para ver las diferencias entre los tres ficheros.

Notas

1. El intérprete de órdenes reemplaza las órdenes entrecomilladas con comillas inversas por la salida estándar de la orden.
2. **wget** es un cliente de web no interactivo al que limitamos el caudal para que el grupo de procesos dure un tiempo suficiente. El programa **tr** convierte las secuencias no alfabéticas en fin de línea. El programa **uniq** agrupa palabras iguales consecutivas y las escribe una vez precedidas de un contador.
3. Por supuesto también puede usar otros editores más profesionales, como **vim**, **gvim** o **emacs**, pero la dificultad en aprenderlos es mucho mayor.

Práctica 4. Compilación, ejecución, interpretación, rendimiento.

4.1. Objetivos

Introducir las diferencias y semejanzas entre Java y C.

Comprender el funcionamiento de diversas máquinas virtuales que se ven en un ordenador, incluyendo las definidas por lenguajes de alto nivel traducidos (C y Java), interpretados (Bash), el nivel máquina convencional, el nivel del sistema operativo y el nivel de la máquina virtual Java.

Así mismo observar el rendimiento de programas funcionando en distintos entornos de compilación y ejecución.

4.2. Examen de programas en C y sus equivalentes en Java

Estudie los programas siguientes, comparando las versiones en C y en Java: `nada.c`:

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

int main (int argc, char* argv[]) {
  int i, N;
  N = atoi(argv[1]);
  for (i=0; i<N; i++) ;
  printf("i=%d\n", i);
  exit(0);
}
```

`Nada.java`:

```
public class Nada {

  public static void main (String[] args) {
    int i, N;
    N= Integer.parseInt(args[0]);
    for (i=0; i<N; i++);
    System.out.println("i="+i);
  }

}
```

`rutinas.c`:

```
#include <stdlib.h>
```

```
#include <stdio.h>

int rutina(int p) { return p+p; };

int main (int argc, char* argv[]) {
  int i, N, r=0;
  N = atoi(argv[1]);
  for (i=0; i<N; i++)
     r= rutina(i);
  printf("i=%d,r=%d\n", i,r);
  exit(0);
}
```

y `Rutinas.java`:

```
public class Rutinas {

  static int rutina(int p) { return p+p; };

  public static void main(String[] args) {
    int i, N, r=0;
    N= Integer.parseInt(args[0]);
    for (i=0; i<N; i++)
      r= rutina(i);
    System.out.println("i="+i+",r="+r);
  }

}
```

Todos ellos toman como argumento una cadena de caracteres numérica que utilizan para determinar el número de vueltas de un bucle. Si no se pasa la cadena de caracteres numérica, en los programas en C se producirá una violación de memoria, y en los programas en Java una excepción por índice incorrecto. Finalmente escriben uno o dos valores. Los programas en C además informan al sistema operativo que terminan con éxito (`exit(0)`); ello no es necesario en los programas en Java, que automáticamente entregan un cero al sistema operativo si no terminan por excepción.

4.3. Compilación, ejecución y medida de tiempos

4.3.1. C sin optimizar

Compile y monte (con **gcc *`programa.c`* `-o programa`**) los programas en C y ejecútelos (con **`./programa` *`vueltas`*** [1]). Busque un valor de *`vueltas`* que haga que los programas duren varios segundos[2]. En lugar de usar su reloj de pulsera para medir el tiempo, utilice la orden intrínseca de la *shell* **time**:

```
time ./programa vueltas
```

Verá que el tiempo transcurrido (`real`) es algo mayor que el tiempo de procesador gastado en modo usuario (`user`), y que el tiempo de procesador gastado dentro del núcleo (`sys`) es despreciable. Si la máquina está cargada, el tiempo transcurrido será bastante mayor que la suma de usuario y sistema, ya que el procesador hay que repartirlo con otros procesos. Para los dos programas determine la duración de cada pasada por el bucle. Restando la duración de la pasada en el bucle vacío podemos

determinar el tiempo atribuible a las secuencias de instrucciones necesarias para llamar y retornar de un procedimiento sencillo.

4.3.2. C optimizado

Compile los programas anteriores optimizando (con la opción `-O`) y obtenga las medidas anteriores[3]. Verá mejoras sustanciales y probablemente sorprendentes. En particular, el optimizador del compilador de C podría ser tan inteligente que decida que el bucle computacionalmente no sirve para nada y lo elimine, con lo que el tiempo observado es nulo. Si no se elimina el bucle, se mejora mucho su eficiencia. Con la opción `-O2` o `-O3` seguro que elimina el bucle.

4.3.3. Java sin optimizar

Compile los programas Java para la máquina virtual Java (con **javac *`Clase.java`***). Interprételos seguidamente con una máquina virtual Java (con **java `-Xint` `Clase` *`vueltas`*** [4]) y mida su duración con la orden intrínseca **time**. Determine el tiempo atribuible a a las rutinas.

4.3.4. Java optimizado en interpretación

El compilador **javac** no tiene una opción de optimización. Sin embargo muchas máquinas virtuales Java, si determinan que una porción de código virtual se ejecuta un cierto número de veces, la traducen a código nativo del procesador *al vuelo* (*just in time*). Eso hace que código repetitivo se ejecute con eficiencia comparable al código nativo. El intérprete **java** hace eso por defecto (dependiendo de la instalación) y se puede forzar con la opción `-Xmixed`. Repita las medidas anteriores con esta optimización.

4.3.5. Java compilado a código nativo

Aunque Java normalmente se traduce a código intermedio que luego se interpreta, puede compilarse a código nativo directamente. Lo haremos con el compilador de GNU, con ayuda de su frontal para Java **gcj**. Puede usarlo así: **gcj** `--main=Clase` *`Clase.java`* `-o Clase`. Puede usar la opción `-O` para optimizar.

4.4. Llamadas al sistema

Mida la duración de una llamada al sistema sencilla (`time`) con el programa llamadas.c (el tipo time_t está definido como un entero):

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

int main (int argc, char* argv[]) {
  int i, s, N;
  time_t t;
  N = atoi(argv[1]);
  for (i=0; i<N; i++)
    t= time(NULL);
```

```
  printf("%d\n", i);
  exit(0);
}
```

Compárela con la duración de la llamada a un procedimiento ordinario sencillo. Anote cuántas veces más dura esta llamada al sistema que la llamada a un procedimiento. Observe también el porcentaje de tiempo de sistema respecto al tiempo de usuario. Tenga en cuenta que la interfaz entre el lenguaje C y la verdadera llamada al sistema consta de una serie de instrucciones que contribuyen de forma no despreciable al tiempo en modo usuario.

4.5. Examen y rendimiento en un lenguaje interpretado

Los intérpretes de órdenes (*shell*) son lenguajes de muy alto nivel. El usado en el laboratorio (**bash**) tiene cierta capacidad de cálculo entero y podemos hacer programas que hagan lo mismo que los anteriores: `nada.sh`:

```
#!/bin/bash
N=$1
i=0
while [[ $i -lt $N ]]
do
  i=$(($i+1))
done
echo i=$i
```

y `rutinas.sh`:

```
#!/bin/bash

rutina() {
  r=$(($1+$1))
}

N=$1
i=0
while [[ $i -lt $N ]]
do
  i=$(( $i+1 ))
  rutina $i
done
echo i=$i, r=$r
```

Para ejecutarlos, se puede invocar al intérprete. Por ejemplo:

`bash nada.sh` ***`100000`***

Y si quiere ver una traza de lo que ocurre, invocarlo con la opción `-x`:

```
bash -x nada.sh 100000
```

No obstante, el núcleo de Linux, viendo los primeros bytes de un fichero (el llamado número mágico), determina si un fichero que se manda ejecutar tiene instrucciones nativas o debe ser interpretado por un programa (por defecto **/bin/sh**, a menos que aparezan los dos caracteres `#!` al principio, seguidos de la ruta donde está el intérprete a utilizar. En nuestro caso podemos dar permiso de ejecución y ejecutar con o sin medida de tiempos:

```
chmod +x nada.sh
./nada.sh 100000
time ./nada.sh 100000
```

Con estos dos programas, mida el tiempo de ejecución de la rutina.

4.6. Ficheros ofrecidos

- Programa `nada.c`.
- Programa `rutinas.c`.
- Programa `llamadas.c`.
- Clase `Nada.java`.
- Clase `Rutinas.java`.
- Programilla de **bash** `nada.sh`.
- Programilla de **bash** `rutinas.sh`.

4.7. Resultados pedidos

- Rellene la tabla siguiente, con las medidas de los gastos de procesador de la pruebas (en nanosegundos por vuelta). No se incluye la compilación C optimizada, porque elimina bucles:

Medida	C sin optimizar	Java sin JIT	Java con JIT	Bash
No hacer nada				
Llamar a rutinas				
Llamar al sistema		No aplicable	No aplicable	No aplicable

- Rellene la tabla siguiente, estimando el tiempo en nanosegundos atribuible la llamada a un procedimiento sencillo, en los distintos casos.

Medida	C sin optimizar	Java sin JIT	Java con JIT	Bash
Llamada, ejecución y retorno de rutinas sencillas				

- Escriba el modelo y características del procesador (frecuencia de reloj y rendimiento en *bogomips*), que puede obtener ejecutando **cat `/proc/cpuinfo`**. Si tiene más de un procesador, ponga las de uno de ellos.

Parámetro	**Valor**
Marca	
Modelo	
MHz	
Bogomips	

4.8. Examen del código ensamblador generado (opcional)

4.8.1. Por el traductor de C al nivel máquina convencional

Determine cuales son las instrucciones máquina convencionales cuya duración se mide con `rutinas.c`. Para ello genere el código ensamblador de `rutinas.c` y `nada.c` usando la opción `-S` del compilador **gcc**. Luego examine las diferencias de ambos listados con alguna herramienta de comparación [5] y averigüe las instrucciones responsables. Observe que si compila optimizando (opciones `-O` y `-S`), el programa será más corto y probablemente no tenga el bucle. No dedique demasiado esfuerzo a entender el listado en ensamblador, ya que no se pide conocer dicho lenguaje.

4.8.2. Por el traductor de Java a su máquina virtual

Aunque el compilador que hemos usado no permite generar código ensamblador del nivel máquina convencional, podemos desensamblar con **javap `-c Clase > Clase.s`**. Trate de averiguar las instrucciones responsables de la ejecución de la rutina.

4.9. Resultados pedidos de la parte opcional

- Escriba las instrucciones máquina responsables de la ejecución de la rutina:

C sin optimizar	**Java**

C sin optimizar	**Java**

Notas

1. El directorio en curso no debería estar en la variable PATH, que contiene los directorios donde esperamos que haya programas ejecutables, y por tanto sus programas no deberían encontrarse sin dar su ruta. No obstante, el la configuración actual del laboratorio, el directorio en curso sí está en PATH, por lo que basta decir **programa *`vueltas`***.
2. El número de vueltas no puede ser mayor que el entero más grande (2^31-1=2147483647). Si lo pone mayor, se utilizará éste. Si su procesador es demasiado rápido, puede cambiar int por long long int, y `atoi` por `atoll`.
3. No confunda esta opción, con la letra `O` mayúscula, con la forma de nombrar el fichero de salida del compilador, con la `o` minúscula.
4. La opción `-Xint` obliga a una interpretación pura de una instrucción tras otra.
5. En modo gráfico puede usar **meld *`nada.s rutinas.s`*** o **tkdiff *`nada.s rutinas.s`***.

Práctica 5. Máquinas virtuales

5.1. Objetivos

Introducir las máquinas virtuales como concepto y como herramienta para aprender temas de sistemas operativos que requieran los poderes del administrador, entre los que se encuentra la habilidad de instalar un sistema operativo. Estos se pueden también aprender en una maquina real, pero tendría que ser una del estudiante.

5.2. VirtualBox

La máquina virtual a usar será `VirtualBox`. VirtualBox tiene una interfaz gráfica del mismo nombre y programas adicionales, entre los que **VBoxManage** es el más potente para configurar. Trabajaremos con la interfaz gráfica **VirtualBox**, siempre que podamos.

Una máquina virtual (huésped) en este caso consta de un ordenador virtual, con uno o varios discos virtuales, una capacidad determinada de memoria, y hardware diverso, siendo especialmente importantes los controladores de red. Inicialmente trabajaremos con discos virtuales preparados, y luego construiremos uno a partir de una imagen de CD.

En la Figura 5-1 vemos la pantalla inicial de **VirtualBox** sin máquinas virtuales. Pulsando el botón de crear una máquina nueva, nos lanza un asistente que nos pide el nombre de la máquina y tipo de sistema operativo, más que nada para asignarle una configuración por omisión (ver Figura 5-2) y seguidamente recursos como la cantidad de memoria (Figura 5-3) y el disco virtual principal, del que se puede usar uno existente o crear uno nuevo, en cuyo caso habrá que instalarle un sistema operativo a partir de un CD, por ejemplo.

Figura 5-1. Pantalla inicial

Figura 5-2. Nombre y tipo de máquina

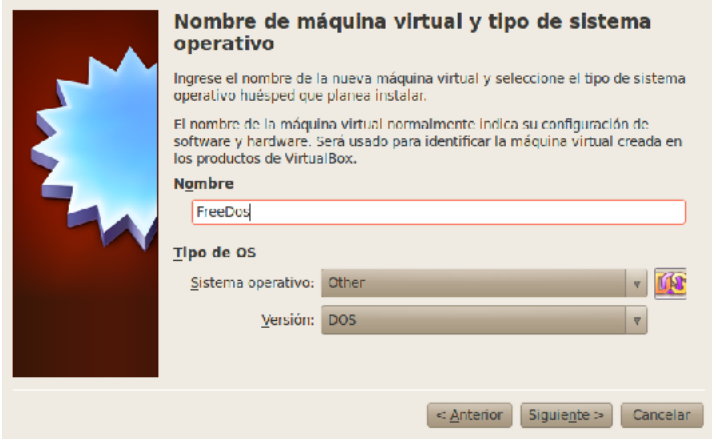
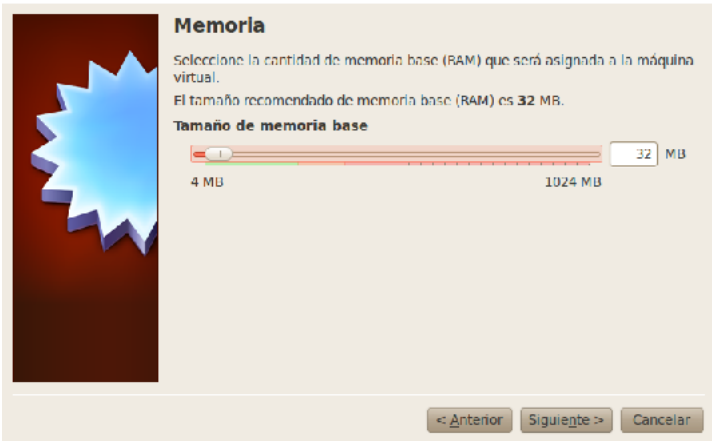


Figura 5-3. Memoria a asignarle



Después de creada la máquina virtual, se puede reconfigurar, cambiando, por ejemplo, la cantidad de memoria, o los discos virtuales, CDs, disquetes, etc. que tenga. También se pueden configurar las tarjetas de red que hay y cómo se conectan.

5.3. La red virtual

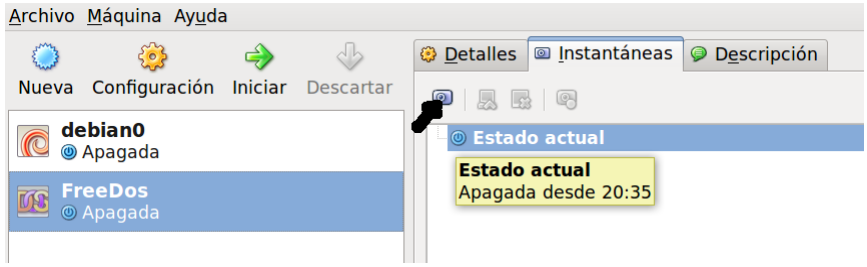
Las máquinas virtuales Linux en esta práctica tienen una interfaz conectada mediante un encaminador NAT (Network Address Traslation) que permitirá comunicarlas con la máquina anfitriona y con el mundo exterior, usando la dirección pública. No obstante, internamente la dirección de la interfaz correspondiente es privada, asignada dinámicamente por el protocolo DHCP, aunque la misma para todas, por lo que no sirve para comunicarse entre sí.

5.4. Obtención y manejo de discos virtuales

Como las cuentas de laboratorio tienen una cuota de disco reducida y los discos virtuales resultan ser

grandes, en el laboratorio se va a trabajar con diferencias de discos virtuales ya preparados e inmutables propiedad del usuario responsable de la asignatura. Si quiere hacer algún cambio y que no se pierda, debe crear una *foto* (*snapshot*) privada que guardará las diferencias en su cuenta (ver Figura 5-4).

Figura 5-4. Toma de fotografías



Los discos virtuales inmutables se ofrecen en el servidor de cuentas, montados por NFS en `/mnt/arqo/maqvirt`. De allí se registrarán en VirtualBox como discos asociados a una máquina virtual.

No obstante las diferencias de discos virtuales pueden crecer rápidamente y dejarle sin cuota de disco. Verfique su cuota con **quota `-v`** (hay un límite blando, que se permite superar durante una semana, y uno duro, insuperable). Limpie su cuenta, mirando previamente lo que más ocupa (por ejemplo, con **du `-a` | sort `-rn`**).

Aquellos que decidan trabajar en su propia máquina, pueden acceder a los discos virtuales en http://www.lab.dit.upm.es/~arqo/maqvirt.

5.5. Máquina con FreeDOS

`FreeDos` es una versión libre del viejo MS-DOS, monousuario y monotarea, uno de los primeros sistemas operativos, tras CP/M, para ordenadores personales con disco y con procesador Intel x86. Use la imagen de disco preparada en `ARQOFreeDos.vdi` e incorpórela en una máquina virtual (sin olvidar la foto) con sistema operativo de tipo `Other/DOS` (Figura 5-2) y 16 Megas de RAM (Figura 5-3).

Arranque la máquina virtual y haga lo que sigue. Tenga cuidado con el ratón. Aunque no lo use, lo puede atrapar el sistema operativo huésped y para liberarlo hay que pulsar la tecla CTRL de la derecha.

- Observe que la interfaz de usuario es en modo texto.
- Observe que no pide usuario ni contraseña.
- Examine el directorio en curso con **DIR**.
- Cambie al directorio `C:\FDOS\BIN` con **CD `C:\FDOS\BIN`** y mire lo que hay allí con **DIR**. ¿Qué significa la `C:`?
- Cree un directorio de trabajo, por ejemplo con **MKDIR `C:\ARQO`**.
- Cambie al directorio de trabajo y haga alguna tarea simple, como guardar el directorio raíz con **DIR C:\ > DIRRAIZ.TXT** y **TYPE DIRRAIZ.TXT**.
- Normalmente los sistemas MSDOS se cerraban apagando sin más la máquina, corriendo el riesgo de perder datos. En FreeDos puede usar **HALT**.

5.6. Servidor Debian GNU Linux sin interfaz gráfica

- Use la imagen de disco preparada en `ARQODebServ.vdi` e incorpórela en una máquina virtual (sin olvidar la foto) con sistema operativo de tipo `Linux/Debian` y 128 Megas de RAM.
- Arranque la máquina virtual y haga lo que sigue. La interfaz de usuario es en modo texto y pide usuario y contraseña. El administrador (`root`) tiene contraseña `rootarqo`. Además hay un usuario genérico `arqo` con contraseña `arqoarqo`.
- Entre como `arqo`. Ejecute al menos un **whoami**, un **id**, un **pwd** un **ls `-la`** y salga con **exit**.
- Entre como `root` Ejecute al menos un **whoami**, un **id**, un **pwd** un **ls `-la`**, un **ls `-l /`** y use **halt** para apagar.

5.7. Estación gráfica Debian GNU Linux

Use la imagen de disco preparada en `ARQODebWS.vdi` e incorpórela en una máquina virtual (sin olvidar la foto) con sistema operativo de tipo `Linux/Debian` y 192 Megas de RAM. Arranque la máquina virtual y haga lo que sigue.

- La interfaz de usuario es en modo gráfico, con escritorio GNOME predeterminado. Entre como usuario `arqo` (con contraseña `arqoarqo`). No se permite entrar como `root` por seguridad, dada la complejidad de los programas gráficos.
- Lance un terminal ordinario en el menú `Aplicaciones -> Accesorios -> Terminal` y compruebe su identidad.
- Lance un terminal de `root` con contraseña `rootarqo` en el menú `Aplicaciones -> Accesorios -> Terminal de Root` y compruebe su identidad.
- Use el menú `Sistema -> Apagar` para apagar.

5.8. Usuarios y permisos

- Arranque la máquina servidora y cree un grupo de usuarios llamado `estudiantes` con **addgroup**. Verifique que el grupo aparece en `/etc/group`, la base de datos de grupos.
- Cree un par de usuarios del grupo `estudiantes` con **adduser**, uno para usted y otro para un amigo o amiga. Verifique que los usuarios aparecen en `/etc/passwd`, la base de datos de usuarios, y en `/etc/group`. Observe que en Debian se crea también, por defecto, un grupo unitario para cada usuario, con el mismo nombre.
- Verifique que se han creado los directorios de los usuarios en `/home` con la propiedad correcta (use **ls `-l /home`**).
- Cree un directorio `/home/estudiantes`, propiedad del *segundo usuario* creado y del grupo `estudiantes`, con permisos de lectura, escritura y ejecución (atravesarlo) para todos los miembros del grupo. Use **chown** y **chmod**.
- Salga de la cuenta de `root` y entre en la suya. Verifique su identidad y pertenencia a grupos con **id**. Verifique que puede escribir ficheros en su cuenta y en `/home/estudiantes`, pero no en la de su amigo.
- Cambie de cuenta a la de `root` para modificar los permisos de `/home/estudiantes` para que los ficheros que allí cree sean propiedad del grupo `estudiantes`. Para ello hay que activar el permiso de cambio de identidad de grupo (**chmod `g+s /home/estudiantes`**).

- Verifíquelo creando un fichero con su usuario y mostrando el dueño y grupo del fichero mediante **ls -l**.

5.9. Paquetería

Un paquete es un conjunto de ficheros que colaboran en proporcionar servicio y que posiblemente dependan de otros. Además tiene asociadas acciones para cuando se instala y se desinstala. Debe ser el superusuario para ejecutar las órdenes de esta sección. Entre en la máquina servidora y haga lo siguiente:

- Mire los paquetes instalados con **dpkg -l**, comprobando que no está el paginador **less**.
- Actualice su conocimiento sobre los paquetes disponibles con **aptitude update**.
- Puede ver detalles del paquete con **apt-cache show less**.
- Puede instalar el paginador con **aptitude install less**, instalándose además todos los paquetes de los que depende.
- Puede buscar navegadores de web en modo texto con **apt-cache search text mode www browser**. Instálese uno: **lynx**, **w3m**, **links**, etc. Navegue.
- Instálese **gpm** para usar el ratón en modo texto.
- Elija un editor en modo texto. **nano** (que también se invoca como **pico**) es sencillo y viene preinstalado. Si no le gusta, puede instalar **jed**, **vim**, **elvis**, etc.
- Cree un directorio llamado prestaproc con **mkdir *prestaproc*** y cambie a ese directorio.

  Cópiese programas en C de la práctica de medida de prestaciones para hacerlos funcionar aquí. Puede traerlos directamente de la web con el navegador que instaló, o la práctica entera con **wget -r -nd *http://www.lab.dit.upm.es/~arqo/p/prestaproc***. El compilador de C está preinstalado.
- Use **halt** para apagar.

5.10. Entornos gráficos y gestores de ventanas (opcional)

El entorno de trabajo que tenemos instalado en la estación gráfica es GNOME, que consume gran cantidad de recursos. A veces nos es suficiente un simple gestor de ventanas, como el viejo `twm`. Puede probarlo directamente, porque está preinstalado. También puede probar otros gestores de ventanas y escritorios, cuidando de no llenar su cuota de disco.

- Arranque la máquina servidora, después de dar el nombre de usuario, pero antes de entrar, puede elegir en un menú situado en la parte inferior de la pantalla, si queremos GNOME o twm. Elija `twm` y verá una pantalla limpia, únicamente con la imagen de fondo. Con el botón izquierdo del ratón se desplegará un menú con el que puede elegir aplicaciones y cerrar la sesión.
- Puede instalar el escritorio ligero `lxde` con **aptitude install lxde-core**.
- Puede salir e iniciar la sesión con el gestor de ventanas de `lxde`, que se llama `openbox`, más moderno que `twm` (el menú sale con el botón derecho del ratón).
- Puede salir e iniciar la sesión con el escritorio ligero `lxde` y probarlo. Se sale con el icono inferior derecho.
- Observe que la siguente vez que entre se pondrá, por omisión, el último entorno que usó.

5.11. Instalación de un Linux mínimo (Opcional)

Instalar esta máquina virtual le ocupará unos 880 Megabytes de disco. Verifique si tiene suficiente cuota de disco antes de iniciar esta tarea.

Cree una máquina virtual nueva, denominada `debarqo` a partir de un CD de instalación, cuya imagen puede encontrar en `debian-6.0.2.1-i386-netinst.iso` o un nombre parecido. Cree pues dicha máquina virtual nueva de 128 Megabytes de RAM y 1,5 Gigabyte de disco duro *creciente* no reservado (va creciendo bajo demanda hasta esa cantidad) y póngale, como CD-ROM (menú de almacenamiento) la imagen antes mencionada. Luego arránquela del CD y comience la instalación en disco duro:

- Elija la instalación ordinaria en modo texto (es parecida a la gráfica y más ligera).
- Elija el idioma, país y teclado.
- Elija la interfaz de red de instalación, si lo pregunta. Será la primera, `eth0`, que se comunicará con los servidores a través de un NAT.
- Elija nombre y dominio de la máquina. Por ejemplo, `debarqo` y `lab.dit.upm.es`. No se va a usar.
- Elija servidor de software (preferiblemente en España, en p/maqvirt/ftp://ftp.rediris.es, sin especificar proxy, ya que hay uno transparente en el laboratorio).
- Elija contraseña del administrador (`root`) y el nombre y contraseña de un usuario genérico.
- Elija zona horaria.
- Divida el disco en particiones ordinarias, con ayuda del particionado manual. Se crearán dos particiones primarias separadas para el sistema de ficheros y la zona de paginación. Para ahorrar disco anfitrión,, la de paginación será la primera y pequeña (eg: 100Mb). La otra será `ext4` y *arrancable*. Deje que el instalador dé formato a los sistemas de ficheros e instale el sistema básico.
- A la hora de elegir las tareas que va a realizar el sistema, desmarcaremos todas, incluyendo `Sistema estándar`, para tener un sistema muy pequeño.
- Permita instalar el cargador **grub**.
- Deje reiniciar la máquina.
- Si lo desea, instale todos los paquetes que instaló antes y use esta máquina para prácticas sucesivas. En ese caso, para ahorrar espacio, borre el otro servidor. Para que funcione correctamente en la red virtual, debe modificar `/etc/network/interfaces` para que ponga:

```
auto lo eth0 eth1
iface lo inet loopback
iface eth0 inet dhcp
iface eth1 inet dhcp
```

5.12. Prácticas en las máquinas virtuales (opcional)

La mayoría de las prácticas que siguen están pensadas para hacerlas en la máquina real, pero puede hacerlas en la máquina virtual que desee. En particular, la estación de trabajo gráfica tiene todo lo necesario y, si algo faltara, podría instalarlo, cuidando su cuota de disco; *¡No actualice el software, ya que se pasará de cuota!* . Probablemente deba subir la cantidad de memoria principal, dejando al menos 600 Mb para la anfitriona. Para mayor comodidad, puede ponerla en pantalla completa.

Las imágenes proporcionadas de servidor y estación montan al arrancar, en el directorio `/casa`, un recurso de la máquina anfitriona llamado `casa`, que puede configurase en el menú `Configuración -> Carpetas compartidas` con una ruta de la máquina anfitriona, que puede ser el directorio donde tenga sus prácticas. De ese modo puede aceder a ellas desde la máquina real y las virtuales. Los ficheros aparecerán como pertenecientes al usuario `arqo`. Esto se especifica en una línea del fichero `/etc/fstab` que pone:

```
casa /casa vboxsf defaults,uid=1000,gid=1000 0 0
```

donde `uid=1000,gid=1000` especifican el usuario `arqo` del grupo `arqo` y que puede cambiar a otro usuario interno.

Práctica 6. Programación del tratamiento de señales

6.1. Objetivos

Entender la programación con señales de de unix.

6.2. Señales en la interfaz de programación de Unix

Compile el programa `psignal.c`:

```
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>

#define VUELTAS 10000000000LL

void confirmar(int sig) {
  char resp[100];
  write(1, "Quiere terminar? (s/n):", 24);
  read(0, resp, 100);
  if (resp[0]=='s') exit(0);
}

int main(void) {
  long long int i;
  signal(SIGINT, SIG_IGN);
  write(1, "No hago caso a CONTROL-C\n", 25);
  for (i=0; i<VUELTAS; i++);
  signal(SIGINT, SIG_DFL);
  write(1, "Ya hago caso a CONTROL-C\n", 25);
  for (i=0; i<VUELTAS; i++);
  signal(SIGINT, confirmar);
  write(1, "Ahora lo que digas\n", 19);
  for (i=0; i<VUELTAS; i++);
  exit(0);
}
```

y ejecútelo, intentando abortarlo en varios momentos con **CONTROL-C**. Si los tiempos de los bucles son demasiado cortos o demasiado largos, ajuste el contador de los bucles[1].

Ejecútelo en una ventana y mándele la señal de interrupción desde otro terminal (con **kill *-SIGINT proceso***) para ver que el comportamiento es equivalente [2].

Modifíquelo para que si se le manda la señal de terminar (`SIGTERM`) escriba el mensaje `"No quiero terminar"` y continúe por donde iba.

6.3. Ficheros ofrecidos

- `psignal.c`.
- El ejecutable de la modificación pedida: `psignal2`.

6.4. Resultados pedidos

- Explicación del comportamiento del programa `psignal.c`.
- La modificación pedida: `psignal2.c`.

Notas

1. Como estamos al límite de la capacidad de los enteros, para alargar el bucle utilizamos un `long long int`. Tenga en cuenta que en C, las constantes de ese tipo llevan el sufijo `LL`, como en `10000000000000000LL`.
2. Puede ver desde otra ventana todos los procesos ligados a cualquier perminal con **ps u**.

Práctica 7. Programación de procesos y hebras

7.1. Objetivos

Entender los mecanismos de creación y terminación de procesos de Unix. Iniciarse con las hebras.

7.2. Requisitos

Estudio del control de procesos de Unix. Estudio de las funciones **fork**, **wait** y **sleep**, **write** y **exit**.

7.3. Creación simple de procesos

Compile el programa `pfork.c`:

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

const int nc= 10;
int i, j;
char c;

int main(int argc, char *argv[])
{
    int np= atoi(argv[1]);

    for (i = 1; i <= np; i++) {
        if (fork() == 0) {
            for (j = 0; j < nc; j++) {
                c = 'A' + i - 1;
                write(1, &c, 1);
                sleep(i);
            }
            exit(0);
        }
    }
    exit(0);
}
```

y ejecútelo con diferentes parámetros, explicando los resultados. Pruebe, por ejemplo, con 3 procesos:

```
./pfork 3
```

Utilice **ps** para ver los procesos que se han creado y vea que uno desaparece al cabo de 10 segundos, el segundo al cabo de 20 segundos y el tercero al cabo de 30 segundos. Con la opción `-l` puede ver también el identificador de proceso del padre (`PPID`). Observe que todos ellos son hijos del proceso 1, ya que su padre ha terminado inmediatamente y han quedado huérfanos.

7.4. Espera por la terminación

Ahora ejecute de la siguiente modificación (`pforkwait.c`), explicando el resultado:

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

const int nc= 10;
int i, j;
int pidfork, pidwait, estado;
char c;

int main(int argc, char *argv[])
{
    int np= atoi(argv[1]);

    for (i = 1; i <= np; i++) {
        if ((pidfork = fork()) == 0) {
            for (j = 0; j < nc; j++) {
                c = 'A' + i - 1;
                write(1, &c, 1);
                sleep(i);
            }
            exit(0);
        }
        printf("Arrancado el proceso %d\n", pidfork);
    }
    for (i = 1; i <= np; i++) {
        pidwait = wait(&estado);
        printf("Termina el proceso %d", pidwait);
        if (WIFEXITED(estado)) printf(" con exit(%d)\n", WEXITSTATUS(estado));
        else if (WIFSIGNALED(estado)) printf(" con la seÃ±al %d\n", WTERMSIG(estado));
    }
    exit(0);
}
```

Ejecútelo también en *la sombra (background)* y vea el número de procesos y las relaciones de paternidad. Pueden usar las opciones `fu` de **ps** para verlo gráficamente (con esta opción también vemos todos los procesos del usuario ligados a cualquier terminal). Durante el funcionamiento mate alguno de los hijos con **kill *`pid`*** y vea como se reporta.

7.5. Trazas para ver llamadas al sistema

Use **strace** con la opción `-f` para buscar las verdaderas llamadas al sistema de `fork`, `wait`, `sleep` y `exit`. Observe que la de `fork` (`clone`) también sirve para crear hebras. Ojo, la que aparece en las trazas no es la del manual, sino `sys_clone`.

7.6. Uso rudimentario de hebras

Podemos usar la función `clone` de GNU para crear hebras si la llamamos con ciertos parámetros. El primero la función a ejecutar en la hebra, la segunda la pila a usar (la ponemos muy grande para no tener problemas). Como al mayoría de las arquitecturas tiene pilas que crecen hacia abajo, hay que ajustar la pila a su dirección superior. Las variables locales de cada hebra hay que ponerlas en la función, para que vaya una copia distinta en cada pila. Para que sean hebras reales hay que compartir la memoria virtual y el sistema de ficheros, al menos (`CLONE_VM|CLONE_FILES`). Finalmente hay que pasar un parámetro para diferenciar lo que hace cada hebra. Este programa: (`pclone1.c`) debería funcionar perfectamente:

```
#define _GNU_SOURCE
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>

const int nc= 10;

int escribe(void* pc)
{
    int i, j;
    char c;
    i= *(int*)pc;
    for (j = 0; j < nc; j++) {
        c = 'A' + i - 1;
        write(1, &c, 1);
        sleep(i);
    }
    return(0);
}

int main(int argc, char *argv[])
{
    int i;
    int np= atoi(argv[1]);
    for (i = 1; i <= np; i++)
        clone(escribe, malloc(100000)+100000, CLONE_VM | CLONE_FILES, &i);
    exit(0);
}
```

y, sin embargo no lo hace, ya que a las hebras no les da tiempo a recoger el parámetro. Hace falta algo de *sincronización*. Aquí (`pclone2.c`) lo haremos con espera activa; en Laboratorio de Programación de Sistemas se estudiarán mecanismos mejores:

```
#define _GNU_SOURCE
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sched.h>

const int nc= 10;
int cogido=0;

int escribe(void* pc)
{
    int i, j;
```

```
    char c;
    i= *(int*)pc;
    cogido=1;
    for (j = 0; j < nc; j++) {
        c = 'A' + i - 1;
        write(1, &c, 1);
        sleep(i);
    }
    return(0);
}

int main(int argc, char *argv[])
{
    int i;
    int np= atoi(argv[1]);
    for (i = 1; i <= np; i++) {
        clone(escribe, malloc(100000)+100000, CLONE_VM | CLONE_FILES, &i);
        cogido=0;
        while (!cogido);
    }
    exit(0);
}
```

7.7. Ficheros ofrecidos

- `pfork.c`.
- `pforkwait.c`.
- `pclone1.c`.
- `pclone2.c`.

7.8. Resultados pedidos

- Explicación del funcionamiento de `pfork.c` y `pforkwait.c`.
- Explicación del número de procesos visibles en las ejecuciones de ambos programas y sus relaciones de paternidad.
- Explicación del funcionamiento de `pclone2.c`.

Práctica 8. Dibujo de imágenes contenidas en ficheros

8.1. Objetivos

Comprender y saber utilizar las operaciones básicas de colaboración entre programas en procesos distintos.

8.2. Requisitos

Estudio de **fork**, **execlp**, **kill**, **wait**, **waitpid**, **sleep**, **usleep** y **nanosleep**.

8.3. Especificación

Haga un pequeño programa que presente, separadas por intervalos de tiempo de 3 segundos, una serie de imágenes pasadas como parámetros.

8.4. Realización

La visualización de cada imagen debe hacerse pasándosela como argumento al programa **display**, invocado así:

```
display imagen
```

Debe mantenerse durante 3 segundos y luego borrarse, para pasar a mostrar otra imagen o se termine.

Para borrar una imagen, basta dar la orden de terminar el proceso que la visualiza mandándole la señal de terminar (`15` o `SIGTERM`). Para evitar que queden procesos *zombis*, conviene recoger el resultado de la muerte del proceso con `wait` o `waitpid` (no obstante, si no se hace, no pasa nada, ya que al terminar el programa, los zombis desaparecen.

El programa **display** debe buscarse en el `PATH`, por lo que recomendamos usar `execlp`. Para esperar entre una foto y la siguiente, pueden usarse las rutinas `sleep` o `usleep` o directamente la llamada al sistema `nanosleep`.

Se recomienda empezar a escribir el programa realizando pequeños pasos y comprobando su buen funcionamiento. Estos pasos pueden ser:

- Crear un programa sencillo que acepte una imagen como parámetro y la muestre usando display.
- Ampliar el programa para que la imagen sólo se muestre durante 3 segundos.
- Ampliar el programa para que se puedan dar como parámetros más de una imagen.

8.5. Prueba

Puede probarse el programa con unos dibujos de flora y fauna de riberas que pueden encontrarse en `imagenes/`.

8.6. Ficheros ofrecidos

- Una solución compilada: `pinta`.
- Las imágenes para probar: `imagenes/`.

8.7. Resultados pedidos

- El módulo fuente: `pinta.c`.

Práctica 9. Programación de entrada/salida básica

9.1. Objetivos

Comprender las operaciones básicas de entrada/salida de un secuencial del sistema operativo.

9.2. Estudio de un programa de copia de ficheros

Estúdiese las operaciones `open`, `read`, `write`, `close` y `exit` y estudie y ejecute esta versión simplificada del programa **cp**, de copia de ficheros, que toma como parámetros un fichero origen y uno destino, solamente. Note que si el fichero destino existe, es reemplazado, y si su tamaño era mayor, es acortado.

`cpsimple.c`:

```
#define TAMANO 1024

#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>

char buf[TAMANO];

static void error(char* mensaje) {
   write(2, mensaje, strlen(mensaje)); exit(1);
}

int main(int argc, char* argv[]) {
    int leidos, escritos, origen, destino;

    if (argc != 3) error("Error en argumentos\n");
    if ((origen = open(argv[1], O_RDONLY)) < 0) error("Error en origen\n");
    if ((destino = open(argv[2], O_CREAT
                              | O_WRONLY
                              | O_TRUNC, 0666)) < 0) error("Error en destino\n");
    while ((leidos = read(origen, buf, TAMANO)) > 0) {
       if((escritos = write(destino, buf, leidos)) < 0) error("Error en escritura\n");
    }
    if (leidos < 0 ) error("Error en lectura\n");
    close(origen);
    close(destino);
    exit(0);
}
```

Observe que se dan inicialmente permisos de lectura y escritura para todo el mundo (`0666 = rw-rw-rw-`); esto es normal, ya que los permisos excesivos se retiran con una `umask` apropiada[1].

9.3. Trazado de llamadas al sistema

Seguidamente ejecútelo controlado por el programa **strace**, que nos informa de las llamadas al sistema ejecutadas. Por ejemplo,

strace ./cpsimple `/bin/ls` `mi_ls` .

Puede poner la salida en un fichero para estudiarla mejor (por ejemplo, con **strace `-o traza` ./cpsimple `/bin/ls` `mi_ls`**). Compare esa traza con la del programa **cp**.

Observe que hay bastantes llamadas antes de que nuestro programa llegue a ejecutarse. Forman el *prólogo* y no es necesario que entienda la mayoría de ellas, al menos de momento. Además observe que **exit** no es una llamada al sistema (lo es **exit_group** o **_exit**, dependiendo del sistema).

9.4. Entrada salida de bajo y alto nivel

Cambie la rutina `error` por lo siguiente, que utiliza una rutina de biblioteca, en lugar de una llamada directa al núcleo. Verifique que funcionan de forma similar y las trazas son equivalentes. Para ello invoque el programa con algún error (eg: con número de parámetros incorrecto, con fichero origen que no exista, etc).

```
#include <stdio.h>

static void error(char* mensaje) {
   fprintf(stderr, "%s", mensaje); exit(1);
}
```

9.5. Ficheros ofrecidos

- Programa `cpsimple.c`.
- Programa compilado `cpsimple2`, como `cpsimple`, pero con la rutina `error` cambiada.

9.6. Resultados pedidos

- Diga qué diferencias importantes observa entre las trazas de **cpsimple**, **cpsimple2** y las de **cp**.

Notas

1. La `umask` es un mecanismo de seguridad de Unix que permite retirar permisos en la creación de ficheros de cualquier programa que invoquemos. Así, si creamos un fichero con permisos `0666`, pero la `umask` vale `0022`, los permisos resultantes serán `0644`. Puede ver o modificar la `umask` con la orden de la shell **umask** y puede ver una explicación más detallada con **man umask**.

Práctica 10. Programación de entrada/salida aleatoria

10.1. Objetivos

Practicar la entrada/salida aleatoria.

10.2. Lectura aleatoria

El siguiente programa imprime carácter de un punto arbitrario de un fichero, accediendo aleatoriamente. El primer parámetro es el nombre del fichero y el segundo el número de octeto a imprimir. El octeto se copia a la salida estándar y se imprime su valor en hexadecimal entre paréntesis. Estúdielo y pruébelo.

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>

static void error(char* mensaje) {
   fprintf(stderr, "%s", mensaje); exit(1);
}

int main(int argc, char* argv[]) {
    int f;
    char c;
    off_t pos;

    if (argc != 3) error("Error en argumentos\n");
    if ((f = open(argv[1], O_RDONLY)) < 0) error("Error en origen\n");
    if (lseek(f, pos=atoi(argv[2]), SEEK_SET) < 0) error("Error en posicionamiento\n");
    if(read(f, &c, 1) != 1) error("Error lectura\n");
    printf("%s[%ld]= %c (%x hex)\n", argv[1], pos, c, c);
    exit(0);
}
```

10.3. Escritura aleatoria

Modifique el programa anterior para que escriba un carácter en un punto arbitrario de un fichero, accediendo aleatoriamente. El primer parámetro es el nombre del fichero, el segundo el número de octeto a cambiar y el tercero el carácter a poner.

10.4. Ficheros ofrecidos

- Programa `leepos.c`.
- Programa `escpos`.

10.5. Resultados pedidos

- Cree el programa modificado `escpos.c` y pruébelo.

Práctica 11. Etiqueta de programas

11.1. Objetivos

Aconsejar algunas nociones de etiqueta (buen comportamiento) de programas en cuanto información sobre errores.

11.2. Identificación del programa, mensajes significativios, valor de exit

El siguiente programa modifica el `leepos.c` de la práctica de e/s aleatoria para que cumpla la siguiente etiqueta:

- Todos los mensajes salen por la salida de error (ya estaba).
- Que el programa se identifique siempre.
- Que si los argumentos son incorrectos, diga como se usa el programa.
- Que si no hay errores salga con un valor de retorno igual cero (ya estaba).
- Que si una llamada al sistema produce un error, dé un mensaje apropiado y salga con un valor de retorno distinto de cero (en este caso hemos usado directamente `errno`).

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

static char *p;

static void uso(void) { fprintf(stderr, "Uso:\n    %s f p\n", p); exit(1); }

static void error(void) { perror(p); exit(errno); }

int main(int argc, char* argv[]) {
    int f;
    char c;
    off_t pos;

    p= argv[0];
    if (argc != 3) uso();
    if ((f = open(argv[1], O_RDONLY)) < 0) error();
    if (lseek(f, pos=atoi(argv[2]), SEEK_SET) < 0) error();
    if(read(f, &c, 1) != 1) error();
    printf("%s[%ld]= %c (%x hex)\n", argv[1], pos, c, c);
    exit(0);
}
```

11.3. Internacionalización y localización

Es conveniente que los programas hablen el idioma de los usuarios. Aunque no vamos a aprender a hacer multilingües, sí vamos a hacer uso de funciones multilingües. Observe que usamos `setlocale` para *localizar* los mensajes de `perror` y que tal como tenemos la variable LANG, salgan en castellano.

```
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <locale.h>

static char *p;

static void uso(void) { fprintf(stderr, "Uso:\n    %s f p\n", p); exit(1); }

static void error(void) { perror(p); exit(errno); }

int main(int argc, char* argv[]) {
    int f;
    char c;
    off_t pos;

    p= argv[0];
    if (argc != 3) uso();
    setlocale(LC_ALL, ""); /* Mensajes de error en idioma local */
    if ((f = open(argv[1], O_RDONLY)) < 0) error();
    if (lseek(f, pos=atoi(argv[2]), SEEK_SET) < 0) error();
    if(read(f, &c, 1) != 1) error();
    printf("%s[%ld]= %c (%x hex)\n", argv[1], pos, c, c);
    exit(0);
}
```

11.4. Ficheros ofrecidos

- Programa `leeposingles.c`.
- Programa `leeposinter.c`.

11.5. Resultados pedidos

- Verificación de la etiqueta de informe de errores de `leeposingles.c`:
  - Verifique que los mensajes salen por la salida de error (redirigiéndola a un fichero con `2>`).
  - Que el programa se identifica en los errores.

- Que si los argumentos son incorrectos, diga como se usa el programa.
- Que si no hay errores salga con un valor de retorno igual cero (**echo $?**).
- Que si hay errores salga con un valor de retorno distinto de cero (**echo $?**).

Ejemplos

```
./leeposingles leeposingles.c 3
echo $?
./leeposingles leeposingles.c
echo $?
./leeposingles leeposingles.c 2> errores
echo $?
cat errores
./leeposingles leeposingles.c -1
echo $?
./leeposingles XXXXX 3
echo $?
./leeposingles /etc/shadow 3
echo $?
./leeposingles /etc/shadow 3 2> errores
cat errores
... etc.
```

- Verifique internacionalización de `leeposinter.c`:

```
echo $LANG
./leeposinter leeposinter.c -1
./leeposinter XXXXX 3
./leeposinter /etc/shadow 3
LANG=fr_FR.utf8
echo $LANG
./leeposinter leeposinter.c -1
./leeposinter XXXXX 3
./leeposinter /etc/shadow 3
LANG=C
echo $LANG
./leeposinter leeposinter.c -1
./leeposinter XXXXX 3
./leeposinter /etc/shadow 3
LANG=es_ES.utf8
./leeposinter /etc/shadow 3
... etc
```

Práctica 12. Comunicación entre procesos

12.1. Objetivos

Entender los principios básicos de las comunicaciones entre procesos locales y remotos, especialmente en Unix e Internet.

12.2. Pipes con nombre (FIFOs)

Abra dos ventanas. En una de ellas cree una FIFO con

mkfifo `pruebafifo`

Haga de ella la entrada estándar de un programa; Por ejemplo,

sort < `pruebafifo`.

Observe que **sort** no hace nada. En la otra ventana escriba un texto de varias líneas desordenadas por la FIFO. Por ejemplo, haga

cat > `pruebafifo`

Introduzca algunas líneas y termine con CONTROL-D. Explique lo que sucede.

Ahora haga justo lo contrario. Empiece por el **cat**. Escriba y termine con CONTROL-D. Haga el **sort**. ¿Qué sucede? Explíquelo.

12.3. Búsqueda de objetos de comunicaciones

Busque los que hay en su sistema de ficheros, limitándose a los directorios `/var`, `/home` y `/tmp`, para no perder tiempo, con:

find `/var /home /tmp -type s`

find `/var /home /tmp -type p`

Seguramente no encontrará ninguno de tipo FIFO en su sistema, bien porque no los haya o porque estén en directorios de root protegidos. Sólo aparecerá el creado antes.

Para alguno de ellos, mire qué programa lo tiene abierto, si lo hay, con **/usr/sbin/lsof *`objeto`***.

Mire qué conexiones hay establecidas con sockets (en dominio Unix y en dominio Internet) y por qué proceso, usando **netstat `-p`**.

Mire qué servicios hay pendientes de conexión con sockets y por qué proceso servidor, usando **netstat `-pa`**.

12.4. Uso de sockets en el dominio Internet

A continuación mostramos un cliente y un servidor iterativo del servicio `daytime`, que nos da la fecha y la hora del servidor. El cliente `ctiempo.c` toma como primer parámetro la dirección IP del servidor y como segundo parámetro el número de puerto:

```
/* Cliente de daytime con direcciones IP */
```

```
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <locale.h>

#define LINELEN 128

static char *p;

static void uso(void) { fprintf(stderr, "Uso:\n    %s direccion puerto\n", p); exit(1);

static void error(void) { perror(p); exit(errno); }

int main(int argc, char *argv[]) {
  struct sockaddr_in sin;
  int s, n, ne;
  char buf[LINELEN];
  char *direccion, *puerto;

  p= argv[0];
  if (argc != 3) uso();
  setlocale(LC_ALL, ""); /* Mensajes de error en idioma local */
  direccion = argv[1];   /* IP del servidor */
  puerto = argv[2];      /* Puerto del servidor */
  bzero(&sin, sizeof(sin)); /* Hay que poner todo a ceros */
  sin.sin_family = AF_INET;
  sin.sin_port = htons((u_short)atoi(puerto)); /* puerto en formato de red */
  if (!inet_aton(direccion, &sin.sin_addr)) { /* dirección IP en formato de red */
fprintf(stderr, "direccion en formato incorrecto\n"); exit(1); }
  s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP); /* creo socket */
  if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0) error();  /* Llamo al servid
  while ((n = read(s, buf, LINELEN)) > 0) /* Lee toda la respuesta */
    ne = write(1, buf, n);                /* y la copia a la salida estándar */
  if ((n < 0) || (ne < 0)) error();
  exit(0);
}
```

El servidor `stiempo.c` toma como único parámetro el número de puerto del servicio.

```
/* Servidor de daytime */
#include <unistd.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <locale.h>
#include <time.h>

#define QLEN    5

static char *p;
```

```
static void uso(void) { fprintf(stderr, "Uso:\n    %s puerto\n", p); exit(1); }

static void error(void) { perror(p); exit(errno); }

int main(int argc, char *argv[]) {
  struct sockaddr_in sin;
  char    *pts;
  char    *puerto;
  time_t  ahora;
  int  s, s2;

  p = argv[0];
  setlocale(LC_ALL, ""); /* Mensajes de error en idioma local */
  if (argc != 2) uso();
  puerto = argv[1];
  bzero(&sin, sizeof(sin)); /* Hay que poner todo a ceros */
  sin.sin_family = AF_INET;
  sin.sin_addr.s_addr = INADDR_ANY;
  sin.sin_port = htons((u_short)atoi(puerto));
  s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
  if (bind(s, (struct sockaddr *)&sin, sizeof(sin))<0) error();
  listen(s, QLEN);
  while (1) {
     if ((s2 = accept(s, NULL, NULL)) < 0) error();
     time(&ahora);
     pts = ctime(&ahora);
     write(s2, pts, strlen(pts));
     close(s2);
  }
}
```

Compílelos y ejecútelos en su propia máquina y contra la de un compañero cercano (cliente de uno y servidor del otro y servidor de uno y cliente del otro). Puede averiguar una dirección IP de su máquina con la orden **ifconfig**. No puede usar puertos inferiores a 1024 (son privilegiados),

12.5. Uso de un servicio de nombres

El cliente visto necesita la dirección IP del servidor, pero la gente prefiere utilizar nombres, resueltos en algún servicio de nombres, como el DNS. Use `gethostbyname` para resolver el servidor (mire el manual para ver como se usa; fíjese que se le puede pasar un nombre o una dirección IP en formato humano).

12.6. Interoperabilidad con Java (OPCIONAL)

Verifique la interoperabilidad con los programas Java del libro de teoría (fíjese que en este caso el puerto y la dirección IP se determinan en tiempo de compilación); el cliente, `DateClient.java`:

```
/* Client program requesting current date from server.
 * Figure 3.27
 * @author Gagne, Galvin, Silberschatz
 * Operating System Concepts with Java - Eighth Edition
 * Copyright John Wiley & Sons - 2010.  */

import java.net.*;
```

```
import java.io.*;

public class DateClient {
    public static void main(String[] args) throws IOException {
        InputStream in = null;
        BufferedReader bin = null;
        Socket sock = null;
        try {
            sock = new Socket("127.0.0.1",6013);
            in = sock.getInputStream();
            bin = new BufferedReader(new InputStreamReader(in));
            String line;
            while( (line = bin.readLine()) != null)
                System.out.println(line);
        }
        catch (IOException ioe) { System.err.println(ioe); }
                finally { sock.close(); }
    }
}
```

y el servidor `DateServer.java`:

```
/* Time-of-day server listening to port 6013.
 * Figure 3.26
 * @author Gagne, Galvin, Silberschatz
 * Operating System Concepts with Java - Eighth Edition
 * Copyright John Wiley & Sons - 2010.  */

import java.net.*;
import java.io.*;

public class DateServer {
    public static void main(String[] args) {
        try {
            ServerSocket sock = new ServerSocket(6013);
            // now listen for connections
            while (true) {
                Socket client = sock.accept();
                // we have a connection
                PrintWriter pout = new PrintWriter(client.getOutputStream(), true);
                // write the Date to the socket
                pout.println(new java.util.Date().toString());
                client.close();
            }
        }
        catch (IOException ioe) { System.err.println(ioe); }
    }
}
```

12.7. Ficheros ofrecidos

- `ctiempo.c`.
- `stiempo.c`.
- El ejecutable del cliente de `daytime` pedido: `ctiempon`.
- `DateClient.java`.
- `DateServer.java`.

12.8. Resultados pedidos

- Fuentes del cliente de `daytime` con resolución de nombres: `ctiempon.c`.

Práctica 13. Programación de servicios Web especializados

13.1. Objetivos

Entender los principios básicos del web programado sin herramientas de ayuda.

13.2. Servidor Web trivial

Realice un servidor web que muestre la fecha y la hora local a un navegador remoto, mostrando además el nombre del usuario con que ejecuta el servidor, y el nombre de la máquina. Ponga como parámetro el número de puerto, que no podrá ser privilegiado (menor que 1024). El servidor va a esperar conexiones e inmediatamente responderá una cabecera HTTP apropiada y los datos pedidos en HTML. La máquina la averiguará con la llamada al sistema `gethostname` y el nombre del usuario con `getuid` y `getpwuid`, campo `pw_gecos` (ver manuales). La respuesta puede ser algo así:

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: webstiempo

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head> <title>Servidor web de practicas de ARQO </title> </head>
<body>
  <h1>Usuario</h1>
  <p>JOAQUIN SEOANE PASCUAL</p>
  <h1>Servidor</h1>
  <p>l142</p>
  <h1>Fecha y hora</h1>
  <p>Thu Nov 18 10:01:14 2010</p>
</body>
</html>
```

Pruébelo, preferentemente entre máquinas distintas. Por ejemplo, para la respuesta anterior, si el servidor se hubiera colocado en el puerto 8080 de la máquina `l134`, se habrá usado el URL `http://l134:8080`.

13.3. Servidor Web navegable

Realice un servidor web que muestre la información anterior separada por páginas, dando una el usuario, otra la fecha y la hora, y otra la máquina. La raíz tendría enlaces a los tres. Los URL para en caso anterior serían:

- `http://l134:8080`,
- `http://l134:8080/nombre`,

- `http://l134:8080/tiempo` y
- `http://l134:8080/maquina`.

El servidor debe esperar a que el navegador diga `GET /` o `GET /nombre` , etc, y responder con el HTML y cerrar la conexión[1]

La respuesta de la raiz puede ser algo así:

```
HTTP/1.1 200 OK
Content-Type: text/html
Server: webstiempo2

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
  <head> <title>Servidor web de practicas de ARQO </title> </head>
  <body>
    <ul>
      <li><a href="nombre">Usuario</a></li>
      <li><a href="tiempo">Fecha y hora</a></li>
      <li><a href="maquina">Maquina</a></li>
    </ul>
  </body>
</html>
```

y las de las ramas como quiera. Puede imitar la de la solución compilada (en el navegador pedir *ver los fuentes*).

13.4. Resultados pedidos

- Fuentes del servidor web `webstiempo.c`.
- Fuentes del servidor web `webstiempo2.c`.

Notas

1. En realidad estamos forzando un poco el protocolo HTTP, pero el propósito de la práctica es demostrativo y no nos interesan los detalles.

Práctica 14. Comunicación con máquinas virtuales

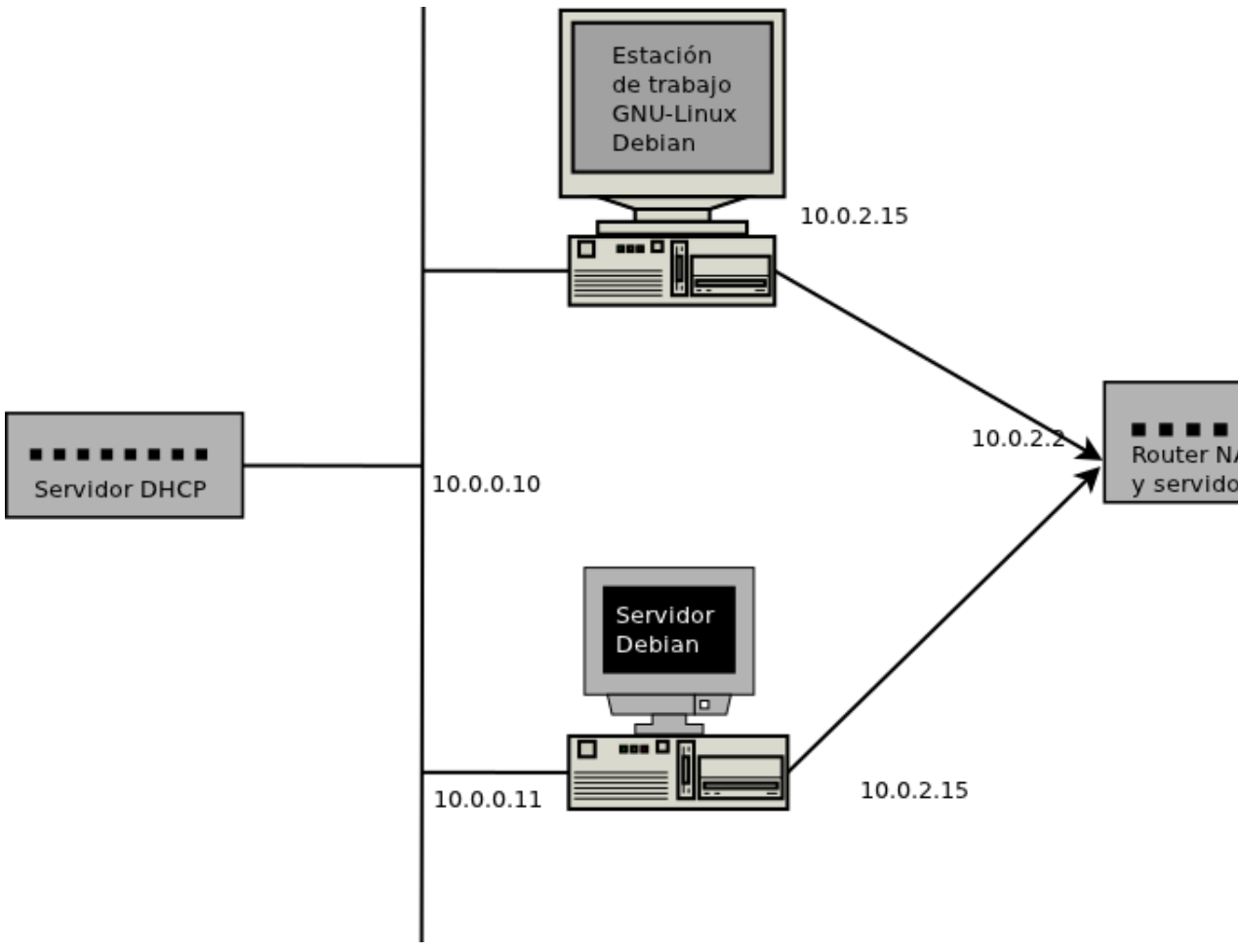
14.1. Objetivos

Simularemos que conectamos nuestro servidor y nuestra estación de trabajo a dos redes, una para la comunicación con el mundo exterior y otra interna, para comunicarse entre ellas.

14.2. La red virtual

A las máquinas Linux se les configurarán dos interfaces. La primera de ellas estará conectada mediante un encaminador NAT (Network Address Traslation) que permitirá comunicarlas con el mundo exterior y con la máquina anfitriona, usando la dirección pública. No obstante, internamente la dirección de la interfaz correspondiente es privada, asignada dinámicamente por el protocolo DHCP, aunque es la misma para todas, por lo que no sirve para comunicarse entre sí. La segunda estará conectada a una intranet (`intnet`), que permitirá que todas se vean entre sí. En ambas redes un servidor `dhcp` asignará direcciones a las interfaces. Véase la Figura 14-1:

Figura 14-1. Estructura de la red virtual



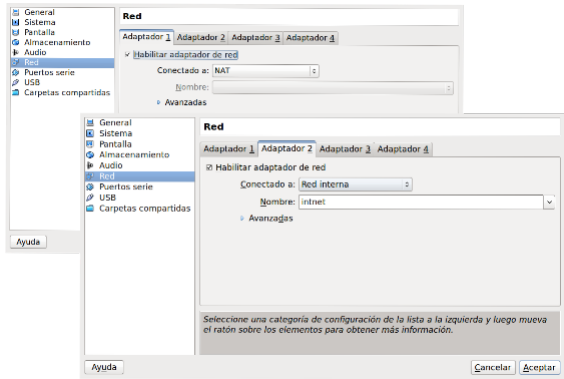
Para configurar esto se *"Configuración -> Red"* y se activa la primera interfaz como NAT y la segunda como interna (nombre `intnet`), como se ve en la Figura 14-2. El servidor DHCP de la intranet hay que activarlo y decirle que sirva un rango de direcciones determinado. Por ejemplo, con:

VBoxManage dhcpserver add --netname intnet --netmask 255.255.255.0 --ip 10.0.0.1 --lowerip 10.0.0.10 --upperip 10.0.0.20 --enable

Si dice que el servidor ya existe, cambie `add` por `modify`.

En cambio, el servidor DHCP de la red NAT ya está configurado.

Figura 14-2. Configuración de la red virtual



14.3. Comprobación de conectividad

- Si no lo ha hecho ya, configure el servidor DHCP de la intranet para que asigne direcciones en el rango 10.0.0.10 .. 10.0.0.20.

  ```
  VBoxManage dhcpserver  add --netname intnet --netmask 255.255.255.0  --ip 10.0.0.1 --lowerip 1
  ```

- Aranque su máquina servidora `ARQODebServ` o `debarqo` y entre como `root`.
- Aranque su estación de trabajo `ARQODebWS` y abra un terminal de `root`.
- Mire, en ambas máquinas virtuales como `root`, su configuración de red con **ifconfig**. La primera interfaz de red es `eth0` y la segunda `eth1`. Existe además `lo` (loopback) que representa la propia máquina. Identifique el número IP asignado a cada interfaz.
- Verifique la conectividad en estación de trabajo y servidor, por ejemplo con:

  ```
  ping 127.0.0.1 # la propia máquina
  ping 10.0.0.10 # el servidor
  ping 10.0.0.11 # la estación de trabajo
  ping 138.4.26.58 # el servidor web del laboratorio
  ping http://www.lab.dit.upm.es/~arqo # el servidor web del laboratorio por nombre
  ping l122.lab.dit.upm.es # la máquina anfitriona, dando su nombre (supongamos l122)
  ```

- Si desea nombrar las maquinas virtuales por nombre, añada una correspondencia al fichero `/etc/hosts` de cada máquina. En la estación:

  ```
  10.0.0.10       arqodebserv
  ```
  y en el servidor:

```
10.0.0.11       arqodebws
```

Pruebe en ambas:

```
ping arqodebserv
ping arqodebws
```

14.4. Instalación de servicios de red

- Instale en su máquina servidora `ARQODebServ` o `debarqo` el servicio `web` instalando el paquete `apache2`. Verifique que funciona desde la máquina virtual usando en navegador en modo texto que instaló en la práctica Máquinas Virtuales o desde la estación de trabajo virtual `ARQODebWS` con un navegador gráfico.
- Instale el cliente de terminal remoto cifrado `openssh-client`. Conéctese por terminal remoto cifrado a su máquina anfitriona o a cualquiera del laboratorio, verifique que está en su cuenta y salga. Suponiendo que su nombre sea `lola` y su máquina la l122, conéctese con **ssh *lola@l122.lab.dit.upm.es***, ejecute al menos un **whoami**, un **pwd** y un **ls** y salga con **exit**.
- Instale el servidor de terminal remoto cifrado `openssh-server` y verifique que funciona con **ssh *arqo@localhost***, ejecute al menos un **whoami**, un **pwd** y un **ls** y salga con **exit**.

14.5. Acceso desde el exterior

Si se desea, se le pueden poner reglas que conecten puertos de la máquina anfitriona a puertos de máquinas huésped. Estas reglas han de configurarse con la máquina huésped apagada, por tanto, si la tiene corriendo, ejecute **halt** como usuario `root`.

Por ejemplo, se puede redirigir el puerto 8080 al puerto 80 del servidor virtual `ARQODebServ` con:

VBoxManage modifyvm ARQODebServ --natpf1 "guesweb,tcp,,8080,,80"

con lo que se puede acceder al servidor web de `ARQODebServ` desde el anfitrión con **firefox http://localhost:8080**, o desde cualquier sitio con **firefox http://l122.lab.dit.upm.es:8080**, si `l122.lab.dit.upm.es` es el anfitrión.

Se prodría, pero *¡no lo haga todavía!*, redirigir el puerto 2222 al puerto 22 del servidor virtual `ARQODebServ` con:

VBoxManage modifyvm ARQODebServ --natpf1 "guestssh,tcp,,2222,,22"

con lo que se puede acceder al servidor ssh de `ARQODebServ` desde el anfitrión con **ssh arqo@localhost -p 2222**, desde cualquier sitio con **ssh arqo@l122.lab.dit.upm.es -p 2222**, si `l122.lab.dit.upm.es` es el anfitrión.

Como todas las máquinas tienen las mismas contraseñas, hacer esto es muy peligroso. Cambie las contraseñas de `root` y de `arqo` antes.

Una vez configuradas las redirecciones, arranque su servidor virtual y verifique que los servicios de su máquina virtual pueden accederse desde el exterior conectándose al servidor de web y accediendo a la cuenta del usuario `arqo` mediante `ssh` desde la máquina anfitriona.

Práctica 15. Servicios Web en máquinas virtuales

15.1. Objetivos

Explorar más formas de hacer aplicaciones Web usando las máquinas virtuales. Para ello debe arrancar las dos, poner los servidores en la máquina servidora y probarlos con un navegador desde la estación de trabajo.

15.2. Servidor web autónomo

- Si no lo hizo ya, pruebe el servidor web la práctica de comunicaciones **webstiempo2** en la máquina servidora con un navegador en la estación de trabajo[1].

15.3. Servidor web estático

- Vamos ahora a usar el servidor web de verdad, para que haga lo que queremos, sin ocuparnos de la gestión. Ponga un página estática en `/var/www/index.html` con tres páginas hijas (similar al servidor web navegable **webstiempo2.c**, salvo que todos los ficheros deben tener extensión html y la información es estática). El servidor web las pondrá visibles en el puerto 80. Compruébelo. Para no perder tiempo, puede encontrar un ejemplo en p/webvm/ejemplos/var/www/index.html, p/webvm/ejemplos/var/www/nombre.html, p/webvm/ejemplos/var/www/maquina.html y p/webvm/ejemplos/var/www/tiempo.html:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
  <head> <title>Servidor web practicas de ARQO</title> </head>
  <body>
    <ul>
      <li><a href="nombre.html">Usuario</a></li>
      <li><a href="tiempo.html">Fecha y hora</a></li>
      <li><a href="maquina.html">Maquina</a></li>
    </ul>
  </body>
</html>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
  <head> <title>Servidor web practicas de ARQO: maquina</title> </head>
  <body> <h1>colibri</h1> </body>
</html>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
  <head> <title>Servidor web practicas de ARQO: tiempo</title> </head>
  <body><h1>Wed Jan 12 12:24:01 2011 </h1> </body>
</html>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
  <head><title>Servidor web practicas de ARQO</title> </head>
  <body><h1>Joaquin Seoane„,</h1> </body>
</html>
```

15.4. Páginas dinámicas

- Para hacer páginas dinámicas, puede hacer un CGI, que es un programa que recibe información por variables de entorno y conversa por su salida y entrada estándar, sin preocuparse de las comunicaciones. En nuestro caso los programas deben estar en `/usr/lib/cgi-bin`. Los URL son de la forma `http://maquina/cgi-bin/programa?pregunta`. Antes de nada vamos a ver qué valen esas variables de entorno. Para ello podemos usar el programilla de shell que las saca en texto plano

  `entorno.sh`:

```
#!/bin/bash
echo "Content-Type: text/plain"
echo
echo Hola, el entorno es:
echo
printenv
```

  o uno que las saca en HTML:

  `entornohtml.sh`:

```
#!/bin/bash
echo "Content-Type: text/html"
echo
echo '<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
  <head> <title>Entorno</title> </head>
  <body>
    <h1>Hola, el entorno es:</h1>'
printenv | while read variable
do
  echo "<p>$variable</p>"
done
echo '  </body>
</html>'
```

De todas las variables, las que nos interesan son `SCRIPT_NAME` y `QUERY_STRING`. Puede, en C, usarse **getenv()** para obtenerlas, en lugar de recorrer la lista de variables de entorno. Por ejemplo, este CGI (`cgivars.c`) las muestra:

```
#include "stdio.h"
#include "stdlib.h"

int main(int argc, char **argv, char **envp)
{
  printf("Content-Type: text/plain\n\n");
  printf("La variable SCRIPT_NAME vale %s\n", getenv("SCRIPT_NAME"));
  printf("La variable QUERY_STRING vale %s\n", getenv("QUERY_STRING"));
  exit(0);
}
```

No olvide que `cgivars.c` hay que compilarlo y hacer ejectutables para todo el mundo los CGI con

```
chmod ugo+x entorno.sh
entornohtml.sh cgivars
```

15.5. Resultados pedidos

Se pide hacer un CGI en C que muestre la informción dinámicamente. Para ello limpie `webstiempo.c` y cree `wcgi.c`, del que damos una versión compilada en `ejemplos/usr/lib/cgi-bin/wcgi`.

15.6. Ficheros ofrecidos

- Los ejemplos p/webvm/ejemplos/var/www/index.html, p/webvm/ejemplos/var/www/nombre.html, p/webvm/ejemplos/var/www/maquina.html, p/webvm/ejemplos/var/www/tiempo.html, p/webvm/ejemplos/usr/lib/cgi-bin/entorno.sh, p/webvm/ejemplos/usr/lib/cgi-bin/entornohtml.sh, p/webvm/ejemplos/usr/lib/cgi-bin/cgivars.c.
- El cgi compilado `ejemplos/usr/lib/cgi-bin/wcgi`.

15.7. Resultados pedidos

- El cgi `wcgi.c`.

Notas

1. Si desea acceder al servidor **webstiempo2** desde la máquina anfitriona, recuerde añadir una nueva redirección mediante **VBoxManage** y use el puerto elegido al arrancar el servidor de tiempo. Alternativamente, puede parar el servidor apache ejecutando **/etc/init.d/apache2 stop** y arrancar **webstiempo2** en el puerto 80, que ya tiene una redirección configurada.

Práctica 16. Memoria virtual

16.1. Objetivos

Entender el espacio de direcciones virtual de los procesos.

16.2. Especificación

16.2.1. Examen del espacio de direcciones virtuales de los procesos

Examine el espacio de direcciones virtuales de distintos procesos mirando ficheros `maps` del sistema de ficheros del núcleo `proc` (montado en el directorio `/proc`). Este sistema de ficheros proporciona información mantenida por el núcleo y, además, puede servir para configurarlo. En particular, hay directorios cuyos nombres son los números de procesos en ejecución y, dentro de ellos puede encontrarse información sobre dichos procesos. Por ejemplo, puede hacer **cat *`/proc/1534/maps`*** para ver el espacio de direcciones del proceso 1534 [1]. O **cat *`/proc/self/maps`***, para el del proceso en curso, que tendrá el programa **cat** (el enlace simbólico `/proc/self` apunta siempre al proceso en ejecución que lo mira).

La información que puede ver está estrucurada en forma de tabla de seis columnas:

Direcciones	Protecciones y modo de compartición	Desplazamiento	Dispositivo	Inodo	Nombre de fichero
Rango de direcciones virtuales.	`r` si se puede leer, `w` si se puede escribir, `x` si se puede ejecutar, `s` si se comparte con otros procesos, `p` si es privado; es decir, se comparte, pero se duplican las páginas donde se escribe.	Desplazamiento del segmento en el fichero, si es que el segmento ha venido de fichero.	Dispositivo físico de respaldo del segmento (número mayor y menor), si es que el segmento ha venido de fichero.	Número de inodo de respaldo del segmento, si es que el segmento ha venido de fichero.	Nombre del fichero respaldo del segmento, si es que el segmento ha venido de fichero. Aquí veremos el código y los datos inicializados del programa, el cargador dinámico, las bibliotecas dinámicas usadas, catálogos de mensajes, etc.

Por ejemplo, este puede ser el contenido de `/proc/self/maps` si lo leemos con **cat**:

```
00110000-00263000 r-xp 00000000 00:10 31         /lib/tls/i686/cmov/libc-2.11.1.so
00263000-00264000 ---p 00153000 00:10 31         /lib/tls/i686/cmov/libc-2.11.1.so
00264000-00266000 r--p 00153000 00:10 31         /lib/tls/i686/cmov/libc-2.11.1.so
00266000-00267000 rw-p 00155000 00:10 31         /lib/tls/i686/cmov/libc-2.11.1.so
00267000-0026a000 rw-p 00000000 00:00 0
00b98000-00b99000 r-xp 00000000 00:00 0          [vdso]
00d5e000-00d79000 r-xp 00000000 00:10 25         /lib/ld-2.11.1.so
```

```
00d79000-00d7a000 r--p 0001a000 00:10 25         /lib/ld-2.11.1.so
00d7a000-00d7b000 rw-p 0001b000 00:10 25         /lib/ld-2.11.1.so
08048000-08054000 r-xp 00000000 00:10 256        /bin/cat
08054000-08055000 r--p 0000b000 00:10 256        /bin/cat
08055000-08056000 rw-p 0000c000 00:10 256        /bin/cat
0945f000-09480000 rw-p 00000000 00:00 0          [heap]
b7847000-b7848000 rw-p 00000000 00:00 0
b785c000-b785e000 rw-p 00000000 00:00 0
bfed5000-bfef6000 rw-p 00000000 00:00 0          [stack]
```

Observe que la memoria virtual de los procesos se divide en *regiones*. En este caso la décima región contiene el código del programa, con permisos de lectura y ejecución, contenido en el fichero `/bin/cat` (inodo `256` del dispositivo `00:10`), La duodécima región contiene los datos estáticos, con permisos de lectura y escritura, contenidos a partir de la posición `0xc000` del mismo fichero. La décimo-tercera región no está respaldada por disco, y está destinada a memoria dinámica del proceso. Las regiones primera a cuarta corresponden a la biblioteca estándar de C, y las séptima, octava y novena al cargador dinámico, que hace posible que el programa enlace correctamente con la biblioteca. La dieciseis corresponde a la pila y está muy alejada del resto, para que pueda crecer hacia abajo. Mire el manual del sistema de ficheros `proc` para más información (**man 5 *proc***).

16.2.2. Determinación de direcciones virtuales

El siguiente programa direcciones.c, imprime en hexadecimal la dirección virtual de diversas partes de sí mismo. Ejecútelo y explique, razonadamente, en qué región está cada dirección impresa.

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

char e1[0x100], e2[0x100];

char *m = "Hola amigo";

void escribe(char *texto, void *dir) {
 printf("DirecciÃ³n de %-4s = %10x (%10u)\n",
   texto, (unsigned int)dir, (unsigned int)dir);
}

int main(void) {
 char *p1 = malloc(0x1000);
 char *p2 = malloc(0x1000);
 escribe("main", main);
 escribe("e1", &e1);
 escribe("e2", &e2);
 escribe("m", &m);
 escribe("*m", m);
 escribe("p1", &p1);
 escribe("p2", &p2);
 escribe("*p1", p1);
 escribe("*p2", p2);
 sleep(1000);
}
```

Para entender completamente los resultados puede añadir al final del programa una espera larga (por ejemplo, `sleep(200);`) y lanzarlo en *background*, procediendo a examinar el fichero `/proc/`*`número_de_proceso`*`/maps`).

16.2.3. Determinación de la accesibilidad de direcciones virtuales propias

El siguiente programa accesible.c intenta determinar la accesibilidad en lectura y escritura de la dirección virtual que se le pasa como parámetro (en decimal, a menos que empiece por `0`, que será octal, o por `0x`, que será hexadecimal). Los accesos a direcciones no disponibles ocasionan una excepción (`SIGSEGV` o violación de segmento), que es tratada por las rutinas `nolee` y `noescribe`, gracias a la operación `signal`.

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

char dato; char *dir;

void nolee(int s) {
   printf("  no legible\n");
   sleep(1000);
   exit(0);
}

void noescribe(int s) {
   printf("  no escribible\n");
   sleep(1000);
   exit(0);
}

int main(int argc, char *argv[]) {
   if (argc != 2) {
      fprintf(stderr, "%s: se necesita una direccion\n", argv[0]);
      exit(1);
   }
   dir = (char *)strtoul(argv[1], NULL, 0);
   printf(
     "Probando la direccion virtual 0x%x (%u)\n",
     (unsigned int)dir, (unsigned int)dir);
   signal(SIGSEGV, nolee);
   dato = *dir;
   printf("  legible\n");
   signal(SIGSEGV, noescribe);
   *dir = dato;
   printf("  escribible\n");
   sleep(1000);
   exit(0);
}
```

Determine la accesibilidad de direcciones virtuales del significativas para el programa, como la `0`, y otras que puedan estar en zonas de datos y de código. Será necesario utilizar el mismo truco de antes (un `sleep` antes de cada `exit` y mirar el mapa de memoria mientras no termina). Observe que algunas implementaciones de Linux, como la del laboratorio, ponen la pila y el montículo en una dirección aleatoria, por motivos de seguridad.

16.2.4. Exploración del espacio de direcciones virtuales

El siguiente programa explora.c intenta acceder en lectura a la primera posición de todas las páginas su espacio de memoria virtual, informando de los rangos de direcciones legibles. Los accesos a direcciones no legibles ocasionan una excepción (`SIGSEGV` o violación de segmento), que es tratada por la rutina `viola`, gracias a la operación `signal`. Después de esta excepción, no puede reintentarse la instrucción. Por ello se guarda el estado al comienzo del bucle de barrido con `sigsetjmp` y se salta ahí con `siglongjmp`. Ejecútelo e interprete los resultados[2].

```
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
#include <setjmp.h>
#include <stdlib.h>

typedef enum {si, no, no_sabe, me_da_igual} legibilidad;

char *dir, *dir0;
legibilidad acc0;
int tpagina;
sigjmp_buf estado;

void imprime_rango(char *d1, char* d2, legibilidad acc) {
   printf("%8x-%8x (%10u-%10u) ", d1, d2, d1, d2);
   if (acc == si) printf("legible\n");
   else printf("ilegible\n");
}

void informa(legibilidad acc) { /* Sólo imprime rangos */
   if (acc0 == no_sabe) acc0 = acc;
   else
     if (acc != acc0) {
        imprime_rango(dir0, dir-1, acc0);
        dir0 = dir; acc0 = acc;
     }
   dir = dir + tpagina;
   if (dir == NULL) {
      imprime_rango(dir0, dir-1, acc);
      exit(0);
   }
}

void viola(int s) { /* Procesa violaciones de memoria */
   informa(no);
   siglongjmp(estado, 1);
}

int main(void) {
   char dato;
   tpagina = getpagesize();
   acc0 = no_sabe;
   dir = NULL; dir0 = dir;
   signal(SIGSEGV, viola);
   sigsetjmp(estado, 1); /* Salva estado, incluido tratamiento de señales */
   for (;;) {
      dato = *dir;  /* Si es ilegible, no se ejecuta */
      informa(si);
   }
}
```

Compare los resultados con el mapa del propio proceso en ejecución, que puede obtener como antes, insertando una instrucción `sleep` al final y lanzándolo en *background*.

16.3. Ficheros ofrecidos

- Programa `direcciones.c`.
- Programa `accesible.c`.
- Programa `explora.c`.

16.4. Resultados pedidos

- Región correspondiente a cada direción impresa por `direcciones.c`.
- Accesibilidad de al menos las direcciones virtuales `0`, una de código y otra de datos del programa `accesible.c`.
- Mapa de accesibilidad de regiones de `explora.c`, comentado.

Notas

1. En algunas versiones de Linux, como la del laboratorio, el fichero `maps` sólo es legible por el propietario del proceso.
2. Los programas que usen estas funciones deben compilarse sin optimizar, ya que el compilador desconoce su significado.

Práctica 17. Entrada/salida proyectada en memoria virtual

17.1. Objetivos

Comprender el acceso a ficheros vía memoria virtual.

17.2. Requisitos

Las otras prácticas de entrada/salida de ficheros.

17.3. Entrada/salida secuencial por memoria virtual

Estudie y verifique este programa, que realiza la copia de ficheros asignándolos a direcciones virtuales:

```
#include <unistd.h>
#include <stdio.h>
#include <fcntl.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <sys/stat.h>

static void error(char* mensaje) {
   fprintf(stderr, "%s", mensaje);
   exit(1);
}

int main(int argc, char* argv[]) {
    off_t longitud, l;
    struct stat fileinfo;
    int origen, destino;
    caddr_t zona1, zona2;
    char *p1, *p2;

    if (argc != 3)
       error("Error en argumentos\n");
    if ((origen = open(argv[1], O_RDONLY)) < 0)
       error("Error en origen\n");
    if ((destino = open(argv[2], O_CREAT
                              | O_RDWR
                              | O_TRUNC, 0666)) < 0)
       error("Error en destino\n");
    if (fstat(origen, &fileinfo) == -1)
       error("Error en origen\n");
    longitud = fileinfo.st_size;
    if (ftruncate(destino, longitud) == -1)
       error("Error al truncar\n");
    if ((zona1 = mmap(0, longitud, PROT_READ, MAP_PRIVATE,
                      origen, (off_t)0)) == (caddr_t)(-1))
                         error("Error al proyectar origen\n");
    if ((zona2 = mmap(0, longitud, PROT_WRITE, MAP_SHARED,
                      destino, (off_t)0)) == (caddr_t)(-1))
                         error("Error al proyectar destino\n");
```

```
    p1 = (char *)zona1; p2 = (char *)zona2;
    for (l = 0; l < longitud; l++)
        *(p2++) = *(p1++);
    exit(0);
}
```

17.4. E/S aleatoria

Cuando verdaderamente es útil la entrada/salida por memoria virtual es en el acceso aleatorio. Haga versiones de los programas pedidos en la práctica de E/S aleatoria: **leepos** y **escpos**.

17.5. Ficheros ofrecidos

- El programa de muestra `cpmm.c`.

17.6. Resultados pedidos

- Los programas `leeposmm.c` y `escposmm.c` probados.

Práctica 18. Administración de discos

18.1. Objetivos

Indagar la administración de discos, particiones y sistemas de ficheros en disco. Puede hacer esta práctica en el servidor virtual o, con más comodidad, en la estación de trabajo virtual (como `root`).

18.2. Particiones, sistemas de ficheros, montajes simples

- Con la máquina virtual parada instale un disco de 120M en el controlador SATA. Verifique que los discos se han asignado correctamente a su máquina virtual en `Configuración -> Almacenamiento`. Arranque. Lo verá como:

  ```
  /dev/sdb
  ```

- Hágale seis particiones lógicas de tipos FAT 16 (06), Minix(81) y cuatro Linux (82), cada una de 20Mb aproximadamente, con **cfdisk `/dev/sdb`**. Se verán como

  ```
  /dev/sdb5
  /dev/sdb6
  /dev/sdb7
  /dev/sdb8
  /dev/sdb9
  /dev/sdb10
  ```

  ya que los sufijos 1 a 4 corresponden a particiones primarias (máximo 4).

- Cree 6 sistemas de ficheros de distintos tipos en cada una de ellas. Por ejemplo:

  ```
  mkfs.msdos /dev/sdb5 # necesita instalar el paquete dosfstools
  mkfs.minix /dev/sdb6
  mkfs.ext3 /dev/sdb7
  mkfs.ext4 /dev/sdb8
  mkfs.ext4 -b 2048 -N 1024 /dev/sdb9
  mkfs.ext4 -b 4096 -N 32 /dev/sdb10
  ```

  En volúmenes tan pequeños suelen usarse bloques de 1K. En las dos útimas particiones hemos especificado bloques de 2K y 4K y además el número de i-nodos.

- Obtenga información sobre los sistemas de ficheros `ext3` y `ext4`, por ejemplo, con:

  ```
  dumpe2fs -h /dev/sdb8
  ```

  Fíjese en los campos siguientes (damos un ejemplo):

  ```
  Filesystem UUID:          eb51f0a1-b60e-4549-8c2a-3df4c920c8b5
  Inode count:              4016
  Block count:              16032
  Reserved block count:     801
  Block size:               1024
  ```

  El primero es el UUID (identificador único universal), número que no se repite nunca y sirve para identificar los discos aunque se cambien de ranura o de máquina. El número de inodos nos da el máximo número de ficheros que se pueden crear, y el tamaño de bloque pequeño nos optimiza la fragmentación para ficheros pequeños, a costa de rendimiento. De todos los bloques, hay un porcentaje reservado a `root`.

- Monte cada uno de los sistemas de ficheros en un subdirectorio de `/mnt`, o donde quiera. Por ejemplo:

```
mkdir /mnt/msdos ; mount -t msdos /dev/sdb5 /mnt/msdos
mkdir /mnt/minix ; mount -t minix /dev/sdb6 /mnt/minix
mkdir /mnt/ext3 ; mount -t ext3 /dev/sdb7 /mnt/ext3
mkdir /mnt/ext4-1k ; mount -t ext4 /dev/sdb8 /mnt/ext4-1k
mkdir /mnt/ext4-2k ; mount -t ext4 /dev/sdb9 /mnt/ext4-2k
mkdir /mnt/ext4-4k ; mount -t ext4 /dev/sdb10 /mnt/ext4-4k
```

- Verifique que están montados con **mount**.
- Verifique qué sistemas de ficheros permiten crear enlaces duros, enlaces simbólicos, cambiar el propietario, hacer ejecutable un fichero (recuerde **ln**, **ln -s**, **chown**, **chmod**).
- Copie cualquier programa (por ejemplo **/usr/bin/cal**) al sistema de ficheros `ext3` que ha montado, desmóntelo, vea que en `/mnt/ext3` no hay nada, y móntelo de nuevo con las opciones `ro,noexec` (sólo lectura y no ejecutable):

  ```
  umount /mnt/ext3
  ls /mnt/ext3
  mount -o ro,noexec /dev/sdb7 /mnt/ext3
  ```

- Trate de escribir algo o ejecutar el programa.
- Desmonte de nuevo y monte sin opciones.
- Trate de escribir algo o ejecutar el programa.
- Trate de copiar más de 32 ficheros a `/mnt/ext4-4k`.
- Verifique que todos los de tipo ext* tienen un directorio `/lost+found` donde la herramienta de reparación de disco pondrá los ficheros huérfanos (sin inodo), y la cantidad de disco y de inodos totales, libres y usados:

  ```
  ls -l /mnt/*
  df /mnt/*
  df -i /mnt/*
  ```

- Verifique que un fichero pequeñito ocupa un bloque de disco:

  ```
  echo hola > /mnt/msdos/saludo
  echo hola > /mnt/minix/saludo
  echo hola > /mnt/ext3/saludo
  echo hola > /mnt/ext4-1k/saludo
  echo hola > /mnt/ext4-2k/saludo
  echo hola > /mnt/ext4-4k/saludo
  du /mnt/*/saludo
  ```

- Desmonte:

  ```
  umount /mnt/*
  mount
  ls /mnt/*
  ```

- Haga que cuando la máquina arranque estén montados los cuatro sistemas de ficheros. Para ello tiene que añadir a `/etc/fstab` las líneas:

  ```
  /dev/sdb5 /mnt/msdos msdos defaults 0 2
  /dev/sdb6 /mnt/minix minix defaults 0 2
  /dev/sdb7 /mnt/ext3  ext3  defaults 0 2
  /dev/sdb8 /mnt/ext4-1k  ext4  defaults 0 2
  /dev/sdb9 /mnt/ext4-2k  ext4  defaults 0 2
  /dev/sdb10 /mnt/ext4-4k  ext4  defaults 0 2
  ```

  Observe que en este fichero, las particiones del primer disco (`/dev/sda1` y `/dev/sda2`), se identifican por su UUID. Puede saber el UUID de un volumen con **blkid** .
- Reinicie y compruebe que están montados.

18.3. Uniones de directorios (opcional)

- Instale el paquete `mhddfs` y haga un montaje de unión de varios directorios. Por ejemplo, cree un par de ficheros en un directorio y otro par en otro. Luego cree un tercer directorio y haga en él la unión de los dos anteriores. Escriba dentro, desmonte y vea donde se ha escrito:

```
touch /mnt/ext3/x
touch /mnt/ext3/y
touch /mnt/ext4-1k/w
touch /mnt/ext4-1k/z
mkdir /mnt/union
mhddfs   /mnt/ext3 /mnt/ext4-1k /mnt/union
ls   /mnt/union
touch /mnt/union/t
ls   /mnt/union
df   /mnt/union
df   /mnt/ext3
df   /mnt/ext4-1k
fusermount -u  /mnt/union
ls   /mnt/union
ls   /mnt/ext3
ls   /mnt/ext4-1k
```

18.4. Listas de control de acceso (opcional)

Las listas de control de acceso son una opción más fina para poner permisos:

- Instale el paquete `acl`.
- Copie `/etc/passwd` y `/etc/shadow` al sistema de ficheros `ext3`.
- Vea los permisos de `/mnt/ext3/passwd` y `/mnt/ext3/shadow` copiados al sistema de ficheros `ext3`.
- Cambie a su usuario habitual con **su *`usuario`*** y vea que no puede modificar `/mnt/ext3/passwd` ni leer `/mnt/ext4/shadow`.
- Vuelva a ser administrador con **exit** .
- Desmonte el sistema de ficheros y móntelo con la opción `acl`:

```
umount /mnt/ext3
mount -o acl /dev/sdb7 /mnt/ext3
```

- Dé permiso de lectura y escritura en ambos ficheros a su usuario con **setfacl -m *`u:usuario:rw`* shadow passwd** .
- Verifique con **getfacl shadow passwd** .
- Cambie a su usuario habitual con **su *`usuario`*** y vea que puede ver y modificar ambos ficheros.

18.5. Resultados pedidos

- Todo lo que se ha pedido hacer.
- Sistemas de ficheros que permiten crear enlaces duros.
- Sistemas de ficheros que permiten crear enlaces simbólicos.

- Sistemas de ficheros que permiten cambiar el propietario y los permisos.
- El número de ficheros que el han cabido en `/mnt/ext4-4k` y el porcentaje de bloques sin usar que quedan.
- Para cada sistema de ficheros, el uso de disco del fichero `hola`.

Práctica 19. Estructura de los sistemas de ficheros

19.1. Objetivos

Indagar la estuctura de sistemas de ficheros en disco. Use la máquina virtual que usó en la práctica anterior.

19.2. Ficheros dispersos (con agujeros)

En muchos sistemas de ficheros no es necesario que tengan espacio asignado los bloques (unidades de asignación de espacio en disco) que sólo contienen ceros. Estos bloques no asignados se denominan agujeros. Demuéstrelo con este pequeño programa (ponagujero.c)[1], viendo la longitud del fichero que crea con **ls -l *fichero*** y viendo lo que ocupa con **du *fichero*** (en unidades de 1k[2]). También puede usar **ls -ls *fichero*** para ver todo.

```
#define _FILE_OFFSET_BITS 64
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <locale.h>

static char *p;

static void uso(void) { fprintf(stderr, "Uso:\n    %s f n\n", p); exit(1); }

static void error(void) { perror(p); exit(errno); }

int main(int argc, char *argv[]) {
   int fd;
   p= argv[0];
   if (argc != 3) uso();
   setlocale(LC_ALL, "");
   if ((fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0666))<0) error();;
   char dato='!';
   if (lseek(fd, atoll(argv[2]), SEEK_SET)<0) error();
   if (write(fd, &dato, 1)<1) error();
   exit(0);
}
```

Por ejemplo

```
./ponagujero f10000 10000
ls -l f10000
du f10000
ls -ls f10000
```

19.3. Uso de disco

Utilice el programa **ponagujero** para ver lo que ocurre para cada sistema de ficheros creados en la práctica de Administración de discos:

- ¿Qué ficheros se han podido crear? ¿Qué sistema de ficheros permite ficheros más grandes?
- De los que se han podido crear, ¿cuanto disco usan? ¿por qué?

Ejercítelo al menos con segundo parámetro igual a 1, 1000, 10000, 100000, 1000000, 10000000, 100000000, 1000000000 y 10000000000, examinando el fichero generado. Puede usar este programilla, creaficheros.sh, para ir más deprisa:

```
#!/bin/bash

for sf in msdos minix ext3 ext4-1k ext4-2k ext4-4k
do
  for long in 1 1000 10000 100000 1000000 10000000 100000000 1000000000 10000000000 1000
  do
    echo Probando $long en $sf
    ./ponagujero /mnt/$sf/f$long $long
    ls -ls /mnt/$sf/f$long
  done
done
```

Repita el experimento con ponagujero2.c (modifique creaficheros.sh para automatizarlo).

```
#define _FILE_OFFSET_BITS 64
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <locale.h>

static char *p;

static void uso(void) { fprintf(stderr, "Uso:\n    %s f n\n", p); exit(1); }

static void error(void) { perror(p); exit(errno); }

int main(int argc, char *argv[]) {
   int fd;
   p= argv[0];
   if (argc != 3) uso();
   setlocale(LC_ALL, "");
   if ((fd = open(argv[1], O_WRONLY | O_CREAT | O_TRUNC, 0666))<0) error();;
   char dato='!';
   if (write(fd, &dato, 1)<1) error();
   if (lseek(fd, atoll(argv[2]), SEEK_SET)<0) error();
   if (write(fd, &dato, 1)<1) error();
   exit(0);
}
```

19.4. Copia creando agujeros

Observe que este programa de copia (cpaguj.c):

```
#define TAMANO 1024

#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <locale.h>

static char *p;
static void uso(void) { fprintf(stderr, "Uso:\n    %s orig dest\n", p); exit(1); }
static void error(void) { perror(p); exit(errno); }

int main(int argc, char* argv[]) {
    char buf[TAMANO];
    int leidos, escritos, origen, destino, i;
    size_t longitud= 0;
    p= argv[0];
    if (argc != 3) uso();
    setlocale(LC_ALL, "");
    if ((origen = open(argv[1], O_RDONLY)) < 0) error();
    if ((destino = open(argv[2], O_CREAT|O_WRONLY|O_TRUNC, 0644)) < 0) error();
    for (;;) {
       if ((leidos = read(origen, buf, TAMANO)) < 0) error();
       longitud += leidos;
       if (leidos > 0) {
          for (i= 0; (buf[i]==0) && (i<leidos); i++);
          if (i == leidos) { if (lseek(destino, leidos, SEEK_CUR) < 0) error(); }
          else { if ((escritos = write(destino, buf, leidos)) < leidos) error(); }
       }
       else {
         if (ftruncate(destino, longitud) < 0) error();
         close(origen);
         close(destino);
         exit(0);
       }
    }
}
```

crea agujeros, si puede, cuando encuentra secuencias de ceros, cosa que **cpsimple** (de la práctica de entrada/salida básica) no hace. Observe también que el programa **cp** del sistema también lo hace (al menos con la opción `--sparse=always`).

Haga una copia del fichero f1000000 con distintos programas de copia de ficheros, **cpsimple**, **cpaguj** y **cp** sobre distintos tipos de sistemas de archivos (ext3 y ext4). Compare el uso de disco entre el origen y el destino de la copia completando la tabla de la sección de resultados pedidos. Puede usar este programilla, copiaficheros.sh, para ir más deprisa:

```
#!/bin/bash

PATH=$PATH:.:../es_basica
f=f1000000

for sf in ext3 ext4-1k
```

```
do
   d=/mnt/$sf
   cpsimple           $d/$f $d/x1
   cpaguj             $d/$f $d/x2
   cpaguj             $d/x1 $d/x3
   cp --sparse=always $d/$f $d/x4
   cp --sparse=always $d/x1 $d/x5
   du                 $d/$f $d/x?
done
```

19.5. Ficheros ofrecidos

- `ponagujero.c`.
- `ponagujero2.c`.
- `creaficheros.sh`.
- `copiaficheros.sh`.
- `cpaguj.sh`.

19.6. Resultados pedidos

- Para cada uno de los sistemas de ficheros complete las tablas siguientes con el tamaño y el uso de disco de los ficheros creados (si se han podido crear sin error; en caso contrario no ponga nada). Justifique brevemente los resultados indicando si se crean los ficheros y si tienen agujeros o no.

ponagujero	msdos	minix	ext3	ext4-1k	ext4-2k	ext4-4k
1						
1000						
10000						
100000						
1000000						
10000000						
100000000						
1000000000						
10000000000						
100000000000						
1000000000000						

ponagujero2	msdos	minix	ext3	ext4-1k	ext4-2k	ext4-4k
1						
1000						
10000						
100000						
1000000						
10000000						
100000000						
1000000000						
10000000000						
100000000000						
1000000000000						

- Para cada los sistemas de ficheros ext3 y ext4-1k complete las tablas siguientes con el uso de disco del origen y destino de la copia. Justifique brevemente los resultados. Use **strace** para ver las diferencias entre **cpaguj** y **cp**.

operación en `ext3`	origen	destino
cpsimple `/mnt/ext3/f1000000 /mnt/ext3/x1`		
cpaguj `/mnt/ext3/f1000000 /mnt/ext3/x2`		
cpaguj `/mnt/ext3/x1 /mnt/ext3/x3`		
cp `--sparse=always /mnt/ext3/f1000000 /mnt/ext3/x4`		
cp `--sparse=always /mnt/ext3/x1 /mnt/ext3/x5`		

operación en `ext4-1k`	origen	destino
cpsimple `/mnt/ext4-1k/f1000000 /mnt/ext4-1k/x1`		
cpaguj `/mnt/ext4-1k/f1000000 /mnt/ext4-1k/x2`		
cpaguj `/mnt/ext4-1k/x1 /mnt/ext4-1k/x3`		
cp `--sparse=always /mnt/ext4-1k/f1000000 /mnt/ext4-1k/x4`		

operación en `ext4-1k`	**origen**	**destino**
`cp --sparse=always /mnt/ext4-1k/x1 /mnt/ext4-1k/x5`		

Notas

1. El tipo off_t es de 32 bits por defecto. Puede hacerse de 64 bits si definimos previamente en el código la macro `_FILE_OFFSET_BITS` como 64, que es lo que hace el programa.
2. El manual puede decir que son de 512 bytes, lo que no es cierto en el laboratorio. Puede forzarlo con la opción `-k`. En cambio **stat** sí reporta el uso en unidades de 512 octetos. Ambos los llaman *bloques*, pero no tienen que ver ni con la unidad de asignación de espacio, ni con la de transferencia.

Práctica 20. Administración sistemas de ficheros en red (opcional)

20.1. Objetivos

Introducirse en la estuctura y administración de sistemas de ficheros en red. Use la máquina virtual donde suela trabajar: ARQODebServ o debarqo

20.2. Montajes remotos sshfs

Este sistema de ficheros es muy sencillo de usar y vale para acceder a cualquier máquina donde tengamos cuenta. Es muy seguro, pero poco eficiente.

- Instale el paquete `sshfs` y haga un montaje remoto de su cuenta en cualquier máquina del laboratorio. Por ejemplo:

```
mkdir /mnt/sshfs
sshfs usuario@maquina:     /mnt/sshfs # o bien
sshfs usuario@maquina:/home/usuario    /mnt/sshfs
mount
```

- Verifique que tiene acceso según los permisos de *usuario@maquina*.
- Desmonte con **fusermount -u /mnt/sshfs**

20.3. Montajes remotos por NFS

Vamos a hacerlos entre la estación de trabajo (ARQODebWS) y el servidor (ARQODebSrv) virtuales. Asegúrese que las máquinas cliente y servidora tienen configurada la red con dos interfaces tal como se hizo en la práctica de comunicaciones.

- Instale el paquete `nfs-client` en la estación.
- Haga un montaje remoto del directorio donde se publican las prácticas del laboratorio en el Web. Por ejemplo:

```
mkdir /mnt/arqo
mount  naslab:/home/arqo/lib/www  /mnt/arqo
mount
```

  Verá que no le está permitido el montaje remoto. Veremos como se configura.

- Instale el paquete `nfs-server` en el servidor.
- Intente montar el `/home` del servidor desde el cliente, en un directorio creado para ello en el cliente (por ejemplo `/homeremoto`).

```
mkdir /homeremoto
mount  10.0.0.10:/home  /homeremoto
mount
```

  Verá que no puede.

- Verifique que directorios exporta el servidor al cliente con **showmount -e *10.0.0.10***.
- Exporte en el servidor el directorio `/home` al cliente con **exportfs -orw 10.0.0.11:/home**.

- Verifique que directorios exporta el servidor al cliente con **showmount `-e` *`10.0.0.10`***.
- Reintente el montaje y compruebe que puede hasta escribir si los usuarios coinciden, salvo `root`, que no tiene acceso.
- Exporte en el servidor el directorio `/usr/share/doc` a todo el mundo, sólo lectura, con **exportfs `-oro '*':/usr/share/doc`**.
- Intente montar el `/usr/share/doc` del servidor desde el cliente, en un directorio creado para ello en el cliente (por ejemplo `/docremoto`). Verifique la propiedad de los ficheros.

```
mkdir /docremoto
mount  10.0.0.10:/usr/share/doc  /docremoto
mount
ls -l /docremoto
```

Práctica 21. Administración de arrays de discos

21.1. Objetivos

Practicar con algunos sistemas RAID importantes. Haga la práctica en la máquina virtual servidora (como `root`).

21.2. RAID software

Los discos pueden combinarse para mejorar capacidad, rendimiento y fiablidad. Aquí vamos a usar el soporte RAID por software de Linux.

- Con la máquina virtual parada instale tres discos más de 40M en el controlador SATA. Arranque. Los verá como:

  ```
  /dev/sdc /dev/sdd /dev/sde
  ```

- Instale el paquete `mdadm`. Dígale al instalador que no va a usarse nigún RAID en la raíz del sistema de ficheros y que detecte los RAID al arrancar.

21.3. RAID-1 o espejo

- Cree un RAID-1 (mirror) con los tres discos, cree un sistema de ficheros dentro, móntelo, verifique su tamaño (debe ser el más pequeño de los tres si son distintos) y su estado, copie algunos ficheros dentro:

  ```
  mdadm --create  /dev/md/md0   --level=1 --raid-devices=3  /dev/sdc  /dev/sdd /dev/sde
  mkfs.ext4 /dev/md/md0
  mkdir /mnt/raid
  mount /dev/md/md0  /mnt/raid
  df /mnt/raid
          mdadm --detail  /dev/md/md0
  cp /bin/b* /mnt/raid
  sum /mnt/raid/*
  ```

- Simule una avería y copie más ficheros dentro. Funciona:

  ```
          mdadm --set-faulty  /dev/md/md0  /dev/sdc
          mdadm --detail  /dev/md/md0
  cp /bin/c* /mnt/raid
  sum /mnt/raid/*
  ```

- Simule otra avería. Funciona:

  ```
          mdadm --set-faulty  /dev/md/md0  /dev/sdd
  cp /bin/d* /mnt/raid
  sum /mnt/raid/*
  ```

- Simule reemplazar un disco averiado. Funciona:

  ```
  mdadm --remove  /dev/md/md0  /dev/sdc
  mdadm --add  /dev/md/md0  /dev/sdc
          mdadm --detail  /dev/md/md0
  cp /bin/e* /mnt/raid
  sum /mnt/raid/*
  ```

- El programa **mdadm** no permite simular averías que hagan el array inoperante. Podría parar la máquina, quitarle uno de los discos funcionales, reiniciar y ver que se puede montar y tiene los ficheros. Puede parar la máquina otra vez, quitarle el otro disco funcional, reiniciar y ver que el array ha desaparecido. No obstante, al quitar y poner discos, cambia su nombre, y no vamos a complicar la práctica.
- Desmonte y pare.

```
umount /mnt/raid.
mdadm --stop  /dev/md/md0
```

21.4. RAID-5

- Cree un RAID-5 (redundancia repartida) con los tres discos, cree un sistema de ficheros dentro, móntelo, verifique su tamaño (debe ser 2/3 de la suma de los tres discos o el doble del más pequeño de los tres si son distintos) y su estado, copie algunos ficheros dentro y verifique que tolera un fallo. Luego desmonte y pare.

```
mdadm --create  /dev/md/md0   --level=5 --raid-devices=3 /dev/sdc /dev/sdd  /dev/sde
mkfs.ext4 /dev/md/md0
mount /dev/md/md0  /mnt/raid
df /mnt/raid
        mdadm --detail  /dev/md/md0
cp /bin/b* /mnt/raid
etc.
```

21.5. RAID-0

- Cree un RAID-0 (stripping) con los tres discos, cree un sistema de ficheros dentro, móntelo, verifique su tamaño (debe ser el triple del más pequeño de los tres si son distintos) y su estado, copie algunos ficheros dentro y verifique que no tolera fallos. Luego desmonte y pare.

```
mdadm --create  /dev/md/md0   --level=0 --raid-devices=3 /dev/sdc /dev/sdd  /dev/sde
etc.
```

Práctica 22. Administración de volúmenes lógicos (opcional)

22.1. Objetivos

Muchas veces es un incordio el uso de particiones fijas y discos concretos para volúmenes. Podemos tener problemas si se llenan y es complicado aprovechar espacio de particiones vacías. Para solventarlo podemos crear volúmenes lógicos.

22.2. Volúmenes lógicos simples

- Con la máquina virtual parada instale dos discos más de 120M cada uno en el controlador SATA. Arranque. Los verá, si no ha desconectado ningún disco de los RAID, como:

  ```
  /dev/sdf /dev/sdg
  ```

- Instale el paquete `lvm2`.
- Créele un única partición al primero de ellos de tipo LVM (8E) con **cfdisk `/dev/sdf`**. Se verá como `/dev/sdf1`.
- Regístrela como volumen físico para LVM, cree un grupo de volúmenes con el volumen físico creado y dentro cree un par de volúmenes lógicos que quepan, dejando algo para extender el que se necesite:

  ```
  pvcreate /dev/sdf1
  vgcreate gr1 /dev/sdf1
  lvcreate -L 60M -n vol1 gr1
  lvcreate -L 40M -n vol2 gr1
  lvdisplay
  ```

- Cree sistemas de ficheros en los volúmenes lógicos, móntelos en sendos directorios y mida los sistemas de ficheros:

  ```
  mkfs.ext4 /dev/gr1/vol1
  mkfs.ext4 /dev/gr1/vol2
  mkdir /mnt/vol1 /mnt/vol2
  ls /mnt
  mount -t ext4 /dev/gr1/vol1 /mnt/vol1
  mount -t ext4 /dev/gr1/vol2 /mnt/vol2
  df /mnt/vol*
  ```

- Algunos sistemas de ficheros, como `ext4` permiten crecer en caliente, es decir, con el sistema de ficheros montado. Extienda uno de ellos y redimensione el sistema de ficheros para que se adapte al sitio:

  ```
  lvextend -L 70M /dev/gr1/vol1
  lvdisplay
  df /mnt/vol*
  resize2fs /dev/gr1/vol1  # lo adapta al sitio
  df /mnt/vol*
  ```

- Pero `ext4` sólo permite decrecer en frio, es decir, con el sistema de ficheros desmontado. Reduzca el otro, desmontando previamente, y redimensione el sistema de ficheros para que se adapte al sitio que va a haber después, reduzca el volumen, móntelo y observe:

  ```
  umount /mnt/vol2
  resize2fs -fp /dev/gr1/vol2 32M  # lo adapta al sitio que habrá
  lvreduce -L 32M /dev/gr1/vol2
  lvdisplay
  ```

```
mount -t ext4 /dev/gr1/vol2 /mnt/vol2
df /mnt/vol*
```

- Añada el otro disco al grupo de volúmenes y añada al grupo un tercer volumen lógico y extienda el primero:

```
cfdisk /dev/sdg
pvcreate /dev/sdg1
vgextend gr1 /dev/sdg1
lvcreate -L 100M -n vol3 gr1
mkfs.ext4 /dev/gr1/vol3
mkdir /mnt/vol3
mount -t ext4 /dev/gr1/vol3 /mnt/vol3
df /mnt/vol*
lvdisplay
lvextend -L 90M /dev/gr1/vol1
lvdisplay
resize2fs /dev/gr1/vol1  # lo adapta al sitio
df /mnt/vol*
```

22.3. Volúmenes sobre RAID

Para ganar fiabilidad o eficiencia, podemos poner volúmenes lógicos en un RAID.

- Recree el RAID-5 de la práctica Administración de arrays de discos.
- Créele un única partición tipo LVM (8E) con **cfdisk `/dev/md/md0`**. Se verá como `/dev/md/md0p1`.
- Regístrela como volumen físico. Cree un grupo de volúmenes con ese volumen físico. Dentro cree un par de volúmenes lógicos que quepan, dejando algo apara extender el que se necesite. Cree sistemas de ficheros y monte. Compruebe que resiste una avería:

```
pvcreate  /dev/md/md0p1
vgcreate gr2  /dev/md/md0p1
lvcreate -L 20M -n vol4 gr2
lvcreate -L 20M -n vol5 gr2
lvdisplay
mkfs.ext4 /dev/gr2/vol4
mkfs.ext4 /dev/gr2/vol5
mkdir /mnt/vol4 /mnt/vol5
mount -t ext4 /dev/gr2/vol4 /mnt/vol4
mount -t ext4 /dev/gr2/vol5 /mnt/vol5
df /mnt/vol*
mdadm --set-faulty  /dev/md/md0  /dev/sdc
...
```

Práctica 23. Trabajo con directorios

23.1. Objetivos

Manipular directorios, obtener información sobre ficheros y examen de un pequeño programa recursivo. Utilización de un depurador para seguir un programa.

23.2. Requisitos

Estudio de las operaciones de directorio: llamada al sistema (`getdents`) y rutinas de conveniencia (`opendir`, `closedir`, `readdir`, etc), de las de obtener información sobre un fichero (`stat`, `lstat` y `fstat`), y del depurador.

23.3. Especificación

Estudie el programa siguiente, que lista recursivamente los directorios que se le pasan como parámetros.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <limits.h>
#include <dirent.h>
#include <stdlib.h>

int nivel = 0;

void tree (char* path)  {
  DIR* dirp;
  struct dirent* dp;
  if ((dirp = opendir (path)) == NULL)  {
    perror (path);
    return;
  }
  while ((dp = readdir (dirp)) != NULL) {
    struct stat buf;
    char fichero [PATH_MAX];
    int i;
    if ((strcmp (dp->d_name, ".") == 0)
        || (strcmp (dp->d_name, "..") == 0))
      continue;
    sprintf (fichero, "%s/%s", path, dp->d_name);
    for (i = 0; i < nivel; i ++)
      printf ("   ");
    printf ("%s\n", dp->d_name);
    if (lstat (fichero, & buf) == -1) {
      perror(fichero);
      exit(1);
    }
    if ((buf.st_mode & S_IFMT) == S_IFDIR) {
      nivel++;
      tree (fichero);
      nivel--;
    }
  }
```

```
  closedir (dirp);
}

int main (int argc, char* argv []) {
  int i;
  if (argc == 1) tree (".");
  else
    for (i = 1; i < argc; i++)tree (argv[i]);
}
```

- Sitúese en su directorio de trabajo con **cd** y ejecútelo sin parámetros y con diversos parámetros, para comprender su funcionamiento. Puede probar con `.`, `..`, `/bin`, `/boot`, ambos (`/boot /bin`), etc.
- Sígalo con un depurador (**gdb** o **ddd**), pasándole como argumento un directorio con subdirectorios (por ejemplo `/etc`). Sigalo paso a paso y poniendo puntos de parada. Vea el contenido de variables relevantes, incluidas las asignadas automáticamente en la pila en las llamadas recursivas.
- Modifíquelo para que además de nombre del fichero o directorio muestre el número de inodo. Utilice el depurador para ayudarse.

23.4. Ficheros ofrecidos

- Programa `arbol.c`.
- Una solución compilada: `arbol2`.

23.5. Resultados pedidos

- El módulo fuente: `arbol2.c`.