Sistemas Operativos

Segundo Cuatrimestre de 2008

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo práctico

Grupo 8

Abstract

Investigación e implementación de distintas funcionalidades de un Ubuntu JeOS corriendo sobre una máquina virtual (Virtual BOX). Threads, pipes, fork, join, acceso al kernel mediante módulos. Manejo de la consola y organización de archivos y carpetas del Sistema Operativo linux.

Palabras clave

Ubuntu JeOS, Módulo del kernel, Threads

Integrante	LU	Correo electrónico
Matías López y Rosenfeld	437/03	matiaslopez@gmail.com
Francisco Gimenez	324/05	dc.francisco@gmail.com
Facundo Ciccioli	399/03	facundofc@gmail.com

Índice

1. Consignas

Antes de empezar, ejecute:

```
sudo apt-get install man-db manpages manpages-dev
```

De esta manera tendrá acceso a ayuda en línea ejecutando:

```
man <comando>
```

Por ejemplo:

```
man cp
```

Puede además instalar la versión en castellano de la ayuda ejecutando:

```
sudo apt-get install manpages-es
```

Para acceder a la ayuda en castellano ejecute por ejemplo:

```
man -L es cp
```

Tenga presente que no todos los comandos poseen ayuda en castellano.

`sudo` permite a usuarios normales ejecutar comandos que requieren permisos de administrador. Al ejecutar un comando con `sudo` el sistema le pedirá su password, y no el password del administrador (llamado `root` en Linux, siguiendo la tradición de Unix). Esto sucede ya que el sistema permite que ciertos usuarios (que deberían corresponderse con usuarios "privilegiados" del sistema) puedan utilizar `sudo` ingresando solamente su propio password. El usuario por defecto creado en una instalación de Ubuntu tiene este permiso y por lo tanto en Ubuntu no es necesario una cuenta de administrador o `root`.

`apt-get` es el manejador de paquetes de la distribución Ubuntu. Permite instalar, actualizar y desinstalar programas. Más adelante lo utilizaremos para instalar las herramientas necesarias para compilar programas en Linux.

Si se encontrara detrás de un proxy, antes de utilizar `apt-get` debe configurar el proxy. Ejecute el siguiente comando:

```
sudo echo "Acquire::http::Proxy \"http://proxy.uba.ar:8080\";" > /etc/apt/apt.conf
```

1.1. Comandos básicos de Unix

1. `pwd` Indique qué directorio pasa a ser su directorio actual si ejecuta:

   ```
   /home/mlopez
   ```

   ---

   a) `cd /usr/bin`

   ```
   /usr/bin
   ```

   ---

   b) `cd`

   ```
   /home/mlopez
   ```

   ---

   c) ¿Cómo explica el punto anterior?

   ```
   Al utilizar el comando cd sin par'ametros, si existe la variable
   HOME entonces se hace un chdir a ese path.
   ```

   ---

2. `cat` ¿Cuál es el contenido del archivo `/home/<usuario>/.profile`?

```
# ~/.profile: executed by the command interpreter for login shells.
# This file is not read by bash(1), if ~/.bash_profile or ~/.bash_login
# exists.
# see /usr/share/doc/bash/examples/startup-files for examples.
# the files are located in the bash-doc package.

# the default umask is set in /etc/profile
#umask 022

# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
. "$HOME/.bashrc"
    fi
fi

# set PATH so it includes user's private bin if it exists
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin:$PATH"
fi
```

3. `find` Liste **todos** los archivos que comienzan con `vmlinuz`.

```
/boot/vmlinuz-2.6.24-19-virtual
/vmlinuz
```

Estos archivos son imágenes del kernel Linux.

4. `mkdir` Genere un directorio `/home/<usuario>/tp`.

```
mkdir tp
```

5. `cp` Copie el archivo `/etc/passwd` al directorio `/home/<usuario>/tp`.

```
cp /etc/passwd tp/
```

6. `chgrp` Cambie el grupo del archivo `/home/<usuario>/tp/passwd` para que sea el suyo.

```
El archivo ya es mio, pero en caso de tener que hacerlo: chgrp mlopez passwd.
```

7. `chown` Cambie el dueño del archivo `/home/<usuario>/tp/passwd` para que sea su usuario.

```
El archivo ya es mio, pero en caso de tener que hacerlo: chown mlopez passwd.
```

8. `chmod` Cambie los permisos del archivo `/home/<usuario>/tp/passwd` para que:

```
Antes de empezar el pr'oximo paso: chmod 000 passwd para limpiar los permisos originales.
```

- el propietario tenga permisos de lectura, escritura y ejecución

```
chmod u+r+w+x passwd
```

- el grupo tenga sólo permisos de lectura y ejecución

```
chmod g+r+x passwd
```

- el resto tenga sólo permisos de ejecución

```
chmod o+x passwd
```

9. `grep`

   Muestre las líneas que tienen el texto "localhost" en el archivo `/etc/hosts`.

```
127.0.0.1 localhost
::1     ip6-localhost ip6-loopback
```

   Muestre todas las líneas que tengan el texto "POSIX" de todos los archivos (incluyendo subdirectorios) en `/etc`. Evite los archivos binarios y aquellos archivos y directorios que no tienen permiso de lectura para su usuario.

```
grep -R -I -s POSIX /etc/
/etc/init.d/glibc.sh:     echo WARNING: POSIX threads library NPTL requires kernel version
/etc/alternatives/updatedb:\# What shell shoud we use?  We should use a POSIX-ish sh.
/etc/security/limits.conf:\# - msgqueue - max memory used by POSIX message queues (bytes)
```

10. `passwd` Cambie su password.

```
(current) UNIX password:
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

11. `rm` Borre el archivo `/home/<usuario>/tp/passwd`

12. `ln`

    Enlazar el archivo `/etc/passwd` a los archivos `/tmp/contra1` y `/tmp/contra2`.

    Hacer un `ls -l` para ver cuantos enlaces tiene `/etc/passwd`.

```
-rw-r--r-- 3 root root 981 2008-10-08 01:11 /etc/passwd
```

    Estos enlaces se llaman "hardlinks". Cada nuevo enlace referencia el mismo espacio ocupado del disco rígido, y por lo tanto cada hardlink es igual de representativo de esos bytes ocupados del disco rígido. El espacio ocupado solamente se liberará cuando todos los enlaces hayan sido borrados.

    Ahora enlace el archivo `/etc/passwd` de manera "soft" al archivo `contra3`.

Verifique con `ls -l` que no aumentó la cantidad de enlaces de `/etc/passwd`.

```
mlopez@esparrago:~/tp$ ln -s /etc/passwd /tmp/contra3
mlopez@esparrago:~/tp$ ls -l /etc/passwd
-rw-r--r-- 3 root root 981 2008-10-08 01:11 /etc/passwd
mlopez@esparrago:~/tp$ ls -l /tmp/c*
-rw-r--r-- 3 root   root   981 2008-10-08 01:11 /tmp/contra1
-rw-r--r-- 3 root   root   981 2008-10-08 01:11 /tmp/contra2
lrwxrwxrwx 1 mlopez mlopez  11 2008-11-15 18:38 /tmp/contra3 -> /etc/passwd
```

Estos enlaces se llaman “softlinks” y apuntan no a los bytes del disco rígido sino a la ruta del archivo a ser enlazado. Operar sobre el softlink es igual que operar sobre el archivo, sin embargo los softlinks no cuentan en la cantidad de enlaces (ya que no apuntan a los bytes ocupados del disco rígido) y pueden ser borrados sin afectar al archivo original, aunque si se borra el archivo original el softlink quedará huérfano y no apuntará a nada.

13. `mount`

    Monte el CD-ROM de instalación de Ubuntu JeOS y liste su contenido.

    Para hacer esto deberá especificar la ISO de instalación de Ubuntu JeOS como CD-ROM de la máquina virtual. Si bien puede hacer esto como lo hizo para instalar el sistema, si la máquina virtual está corriendo debe hacer click derecho en el ícono con forma de CD-ROM en la esquina inferior derecha de la máquina virtual, y seleccionar **CD/DVD-ROM Image...** (ver figura **??**). En la ventana que aparece seleccione la ISO de instalación (ver figura **??**).

    Presente los filesystems que tiene montados.

```
mlopez@esparrago:~/tp$ sudo mount /dev/cdrom /media/cdrom
mount: block device /dev/scd0 is write-protected, mounting read-only
mlopez@esparrago:~/tp$ mount
/dev/sda1 on / type ext3 (rw,relatime,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
/sys on /sys type sysfs (rw,noexec,nosuid,nodev)
varrun on /var/run type tmpfs (rw,noexec,nosuid,nodev,mode=0755)
varlock on /var/lock type tmpfs (rw,noexec,nosuid,nodev,mode=1777)
udev on /dev type tmpfs (rw,mode=0755)
devshm on /dev/shm type tmpfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
share on comp/ type vboxsf (uid=1000,gid=1000,rw)
```

14. `df` ¿Qué espacio libre tiene cada uno de los filesystems montados?

```
mlopez@esparrago:~/tp$ df -h
Filesystem            Size  Used Avail Use% Mounted on
/dev/sda1             494M  427M   42M  92% /
varrun                 62M   32K   62M   1% /var/run
varlock                62M     0   62M   0% /var/lock
udev                   62M   40K   62M   1% /dev
devshm                 62M     0   62M   0% /dev/shm
df: ‘comp/’: No such file or directory
/dev/scd0             100M  100M     0 100% /media/cdrom0

comp es una carpeta compartidad con la otra m’aquina
```

15. `ps` ¿Cuántos procesos de usuario tiene ejecutando? Indique cuántos son del sistema.

```
ps
  PID TTY          TIME CMD
 3946 tty1     00:00:00 bash
 5528 tty1     00:00:07 vi
 5747 tty1     00:00:00 bash
 5748 tty1     00:00:00 ps
```

```
ps aux

USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  1.3   2844  1692 ?        Ss   12:03   0:02 /sbin/init
root         2  0.0  0.0      0     0 ?        S<   12:03   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S<   12:03   0:00 [migration/0]
root         4  0.0  0.0      0     0 ?        S<   12:03   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   12:03   0:00 [watchdog/0]
root         6  0.0  0.0      0     0 ?        S<   12:03   0:10 [events/0]
root         7  0.0  0.0      0     0 ?        S<   12:03   0:00 [khelper]
root        36  0.0  0.0      0     0 ?        S<   12:03   0:00 [kblockd/0]
root        39  0.0  0.0      0     0 ?        S<   12:03   0:00 [kacpid]
root        40  0.0  0.0      0     0 ?        S<   12:03   0:00 [kacpi_notify]
root        80  0.0  0.0      0     0 ?        S<   12:03   0:00 [kseriod]
root       115  0.0  0.0      0     0 ?        S    12:03   0:00 [pdflush]
root       116  0.0  0.0      0     0 ?        S    12:03   0:00 [pdflush]
root       117  0.0  0.0      0     0 ?        S<   12:03   0:00 [kswapd0]
root       160  0.0  0.0      0     0 ?        S<   12:03   0:00 [aio/0]
root      1216  0.0  0.0      0     0 ?        S<   12:03   0:00 [ata/0]
root      1219  0.0  0.0      0     0 ?        S<   12:03   0:00 [ata_aux]
root      1239  0.0  0.0      0     0 ?        S<   12:03   0:00 [scsi_eh_0]
root      1241  0.0  0.0      0     0 ?        S<   12:03   0:00 [scsi_eh_1]
root      2207  0.0  0.0      0     0 ?        S<   12:03   0:00 [kjournald]
root      2346  0.0  0.5   2220   668 ?        S<s  12:03   0:03 /sbin/udevd --daemon
root      2536  0.0  0.0      0     0 ?        S<   12:03   0:00 [kpsmoused]
dhcp      3484  0.0  0.4   2436   548 ?        S<s  12:04   0:00 dhclient3 -e IF_METRIC=100
root      3813  0.0  0.4   1716   508 tty4     Ss+  12:04   0:00 /sbin/getty 38400 tty4
root      3816  0.0  0.4   1716   508 tty5     Ss+  12:04   0:00 /sbin/getty 38400 tty5
root      3820  0.0  0.9   2568  1208 tty2     Ss   12:04   0:00 /bin/login --
root      3824  0.0  0.4   1716   508 tty3     Ss+  12:04   0:00 /sbin/getty 38400 tty3
root      3826  0.0  0.4   1716   504 tty6     Ss+  12:04   0:00 /sbin/getty 38400 tty6
root      3847  0.0  0.1   1844   236 ?        Ss   12:04   0:00 /usr/sbin/vboxadd-timesync
syslog    3890  0.0  0.5   1936   648 ?        Ss   12:04   0:00 /sbin/syslogd -u syslog
root      3909  0.0  0.4   1868   532 ?        S    12:04   0:00 /bin/dd bs 1 if /proc/kmsg
klog      3911  0.0  1.4   2896  1784 ?        Ss   12:04   0:00 /sbin/klogd -P /var/run/kl
root      3926  0.0  0.3   1764   396 ?        Ss   12:04   0:01 /usr/sbin/gpm -m /dev/inpu
root      3945  0.0  0.9   2568  1204 tty1     Ss   12:04   0:00 /bin/login --
mlopez    3946  0.0  1.8   4804  2280 tty1     S    12:04   0:00 -bash
mlopez    3958  0.0  1.7   4788  2228 tty2     S+   12:17   0:01 -bash
mlopez    5528  0.2  2.6   7060  3304 tty1     S+   17:50   0:08 vi svn/informe/consignas.t
mlopez    5802  0.0  0.9   3756  1152 tty1     S+   18:53   0:00 /bin/bash -c (ps -aux) >/t
mlopez    5803  0.0  0.7   2644  1004 tty1     R+   18:53   0:00 ps -aux
```

16. `umount` Desmonte el CD-ROM de instalación de Ubuntu JeOS.

```
sudo umount /media/cdrom
```

17. `uptime` ¿Cuanto tiempo lleva ejecutando su máquina virtual?

```
18:48:33 up  6:45,  2 users,  load average: 0.00, 0.00, 0.00
```

18. `uname` ¿Qué versión del kernel de Linux está utilizando?

```
uname -a:  Linux esparrago 2.6.24-19-virtual \#1 SMP Wed Jun 18 15:52:10 UTC
2008 i686 GNU/Linux
```

1.2. Salida estándar y pipes

1. STDOUT

   a) Conserve en el archivo `/home/<usuario>/tp/config` la salida del comando `ls` que muestra todos los archivos del directorio `/etc` y de los subdirectorios bajo `/etc`.

```
ls -R /etc/ > /home/mlopez/tp/config
```

   b) Presente cuantas líneas, palabras y caracteres tiene `/home/<usuario>/tp/config`.

```
mlopez@esparrago:~/tp$ wc -l config
730 config
mlopez@esparrago:~/tp$ wc -w config
654 config
mlopez@esparrago:~/tp$ wc -m config
8422 config
```

   c) Agregue el contenido, ordenado alfabéticamente, del archivo `/etc/passwd` al final del archivo `/home/<usuario>/tp/config`.

```
sort /etc/passwd >> /home/mlopez/tp/config
```

   d) Presente cuantas líneas, palabras y caracteres tiene `/home/<usuario>/tp/config`.

```
mlopez@esparrago:~/tp$ wc -l config
754 config
mlopez@esparrago:~/tp$ wc -w config
684 config
mlopez@esparrago:~/tp$ wc -m config
9403 config
```

2. Pipes

   a) Liste en forma amplia los archivos del directorio `/usr/bin` que comiencen con la letra “a”. Del resultado obtenido, seleccione las líneas que contienen el texto `apt` e informe la cantidad de caracteres, palabras y líneas.

   Está prohibido, en este ítem, usar archivos temporales de trabajo.

```
mlopez@esparrago:~/tp$ ls -lh /usr/bin/a* | grep apt
-rwxr-xr-x 1 root root  63K 2008-04-22 12:21 /usr/bin/apt-cache
-rwxr-xr-x 1 root root  18K 2008-04-22 12:21 /usr/bin/apt-cdrom
-rwxr-xr-x 1 root root 9.9K 2008-04-22 12:21 /usr/bin/apt-config
-rwxr-xr-x 1 root root  22K 2008-04-22 12:21 /usr/bin/apt-extracttemplates
-rwxr-xr-x 1 root root 187K 2008-04-22 12:21 /usr/bin/apt-ftparchive
-rwxr-xr-x 1 root root 139K 2008-04-22 12:21 /usr/bin/apt-get
-rwxr-xr-x 1 root root 2.2M 2008-04-04 06:56 /usr/bin/aptitude
-rwxr-xr-x 1 root root 1.9K 2008-04-04 06:56 /usr/bin/aptitude-create-state-bundle
-rwxr-xr-x 1 root root 3.0K 2008-04-04 06:56 /usr/bin/aptitude-run-state-bundle
-rwxr-xr-x 1 root root 5.0K 2008-04-22 12:20 /usr/bin/apt-key
-rwxr-xr-x 1 root root 2.2K 2008-04-22 12:20 /usr/bin/apt-mark
-rwxr-xr-x 1 root root  26K 2008-04-22 12:21 /usr/bin/apt-sortpkgs
mlopez@esparrago:~/tp$ ls -lh /usr/bin/a* | grep apt | wc -l
12
mlopez@esparrago:~/tp$ ls -lh /usr/bin/a* | grep apt | wc -w
96
mlopez@esparrago:~/tp$ ls -lh /usr/bin/a* | grep apt | wc -m
817
```

1.3. Scripting

Escriba un script de shell que sincronice dos carpetas. El script debe recibir las dos carpetas (origen y destino, en ese orden) desde la línea de comando, o preguntarlas interactivamente al usuario en caso de no recibirlas. Además, debe aceptar un modificador `-r` que indica modo de operación recursivo.

Una vez conocidas las dos carpetas con las que se operará, el script deberá copiar todos los archivos de la carpeta origen que no estén presentes en la carpeta destino, y además también deberá copiar todos los archivos presentes en ambas carpetas que tengan una fecha de modificación posterior en la carpeta origen que en la carpeta destino. De esta manera, al ejecutar el script, estará seguro de que la carpeta destino contiene toda la información de la carpeta origen, excepto lo que fue modificado posteriormente en la carpeta destino.

El modificador `-r` indica al script realizar la sincronización también con los archivos de todos los subdirectorios de la carpetas origen y destino.

1.3.1. Script de sincronización

```
#!/bin/bash

if ([ $# -ne 3 ] && [ $# -ne 2 ]); then
    echo "Es necesario introducir 2 o 3 parmetros: $0 parametro1 directorioOrigen directorioDestino"
    exit -1
fi

case $# in
    3 )
        if ([ $1 != "-r" ] || [ ! -d $2 ] || [ ! -d $3 ] ); then
            echo "Si introduce 3 parmetros deben ser: $0 parametro1 directorioOrigen directorioDesti
            echo "La nica opcin para el parmetro 1 es '-r'"
            exit -1
        fi
        echo "Son parmetros vlidos (3)."
        cp -u -r $2/* $3  #capaz hay que agregar un *
        ;;
    2 )
```

```
        if ([ ! -d $1 ] || [ ! -d $2 ] ); then
            echo "Si introduce 2 parmetros deben ser: $0 directorioOrigen directorioDestino"
            exit -1
        fi
        echo "Son parmetros vlidos (2)."
        cp -u $1/* $2 #capaz hay que agregar un *
        ;;
esac


exit 0
```

Sugerencia Recuerde que generalmente el espíritu de los scripts es proveer la mínima lógica necesaria alrededor de otros programas ya presentes en el sistema que puedan proveer parte de la funcionalidad requerida para resolver un determinado problema.

1.4. Ejecución de procesos en background

Antes de resolver esta sección instale los siguientes paquetes en la máquina virtual:

- `nano`: editor de texto.
- `mc`: manejador de archivos.
- `gcc`: compilador de C.
- `libc6-dev`: biblioteca estándar de C.

Cree el archivo `/home/<usuario>/tp/loop.c`. Compílelo con `gcc`. El programa compilado debe llamarse `loop`.

```
gcc -o loop loop.c
```

1.4.1. loop.c

```
#include <stdio.h>
#define IDGRUPO 8 /* Completar con su numero de grupo */
int main() {
    int i, c;
    while(1) {
        c = 48 + i;
        //printf("%d, --> %d\n", c, i);
        printf("%d", c);
        i++;
        i = i % IDGRUPO;
        //sleep(1);
    }
    return 0;
}
```

1. Correrlo en foreground. ¿Qué sucede? Mate el proceso con `Ctrl-c`.

```
Imprime consecutivamente los n'umeros del 48 al 55.
```

2. Ahora ejecútelo en background: `/usr/src/loop > /dev/null &`. Mate el proceso con el comando `kill`.

```
mlopez@esparrago:~/tp$ /home/mlopez/tp/loop > /dev/null &
[1] 3880
mlopez@esparrago:~/tp$ ps
  PID TTY          TIME CMD
 3848 tty2     00:00:00 bash
 3880 tty2     00:00:09 loop
 3881 tty2     00:00:00 ps
mlopez@esparrago:~/tp$ kill -9 3880
mlopez@esparrago:~/tp$ ps
  PID TTY          TIME CMD
 3848 tty2     00:00:00 bash
 3882 tty2     00:00:00 ps
[1]+  Killed                  /home/mlopez/tp/loop > /dev/null
```

1.5. IPC y sincronización

1.5.1. Pipes

Muestre con un ejemplo en lenguaje C como realizar exclusión mutua entre dos procesos utilizando un sólo pipe.

Sugerencia Revise la ayuda de la llamada al sistema `pipe` para construir el pipe y de `fork` para crear nuevos procesos.

1.5.2. pipe.c

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

int main(void)
{
        int     fd[2], res;
        pid_t   pidDelHijo;
        char    mensaje[] = "Tom'a este mensaje, o mejor tom'a mate!\n";
        char    loMensajeado[80];

        pipe(fd);

        if((pidDelHijo = fork()) == -1)
        {
                perror("fork");
                exit(1);
        }

        if(pidDelHijo != 0)
```

```
        {
                /* Este es el proceso hijo */
                printf("Hijo en el paso  1\n");
                /* El hijo cierra la entrada del pipe */
                close(fd[0]);
                printf("Hijo en el paso  2\n");
                /* Manda el "mensaje" atreaces de la salida del pipe */
                write(fd[1], mensaje, strlen(mensaje));
                printf("Hijo en el paso  3\n");
                close(fd[1]);
                printf("Hijo en el paso  4\n");
                exit(0);
        }
        else
        {
                /* Este es el proceso padre */
                printf("Padre en el paso 1\n");
                /* El padre cierra la salida del pipe */
                close(fd[1]);

                printf("Padre en el paso 2\n");
                /* Lee la entrada del pipe */
                res = read(fd[0], loMensajeado, sizeof(loMensajeado));
                printf("Padre en el paso 3\n");
                printf("Mensaje recibido: %s\n", loMensajeado);
                printf("Padre en el paso 4\n");

                exit(0)
        }

        return(0);
}
```

1.5.3. Threads

Antes de resolver este ejercicio instale el paquete `glibc-doc`.

Resuelva el problema de productor/consumidor utilizando threads.

Sugerencia Revise la ayuda de `pthreads`, la implementación de threads en Linux, para conocer los mecanismos de creación y destrucción de threads. Además, `pthreads` provee mecanismos de sincronización que le ayudarán a resolver este ejercicio.

1.5.4. productorConsumidor.c

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

pthread_mutex_t buff_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  buff_lleno_cond  = PTHREAD_COND_INITIALIZER;
pthread_cond_t  buff_vacio_cond  = PTHREAD_COND_INITIALIZER;
```

```
#define MAX 3

char buffer[MAX];
int in, out, n;

void *almacenar(void *val);
void *retirar(void *val);

void testAlmacenar(void * v);
void testRetirar(void * v);

void verBuffer();

main()
{
    pthread_t thread1, thread2, thread3, thread4, thread5, thread6,
    thread7, thread8, thread9, thread10, thread11, thread12, thread21,
    thread22, thread23, thread24, thread25, thread26,thread27,
    thread28, thread29, thread30, thread31, thread32;

    n = 0;
    in = 0;
    out = 0;
    char * mensaje1 ="a";
    char * mensaje2 ="b";
    char * mensaje3 ="c";
    char * mensaje4 ="d";
    char * mensaje5 ="e";
    char * mensaje6 ="f";
    char * mensaje7 ="g";
    char * mensaje8 ="h";
    char * mensaje9 ="i";
    char * mensaje10 ="j";
    char * mensaje11 ="k";
    char * mensaje12 ="l";

    char * mensajeS21 = (char *) malloc(sizeof(char));
    char * mensajeS22 = (char *) malloc(sizeof(char));
    char * mensajeS23 = (char *) malloc(sizeof(char));
    char * mensajeS24 = (char *) malloc(sizeof(char));
    char * mensajeS25 = (char *) malloc(sizeof(char));
    char * mensajeS26 = (char *) malloc(sizeof(char));
    char * mensajeS27 = (char *) malloc(sizeof(char));
    char * mensajeS28 = (char *) malloc(sizeof(char));
    char * mensajeS29 = (char *) malloc(sizeof(char));
    char * mensajeS30 = (char *) malloc(sizeof(char));
    char * mensajeS31 = (char *) malloc(sizeof(char));
    char * mensajeS32 = (char *) malloc(sizeof(char));


    pthread_create( &thread21, NULL, (void *)&testRetirar, mensajeS21);

    pthread_create( &thread1, NULL, (void *)&testAlmacenar, mensaje1);
    pthread_create( &thread2, NULL, (void *)&testAlmacenar, mensaje2);
    pthread_create( &thread3, NULL, (void *)&testAlmacenar, mensaje3);
    pthread_create( &thread4, NULL, (void *)&testAlmacenar, mensaje4);
    pthread_create( &thread5, NULL, (void *)&testAlmacenar, mensaje5);
```

```
pthread_create( &thread22, NULL, (void *)&testRetirar, mensajeS22);
pthread_create( &thread23, NULL, (void *)&testRetirar, mensajeS23);
pthread_create( &thread24, NULL, (void *)&testRetirar, mensajeS24);
pthread_create( &thread25, NULL, (void *)&testRetirar, mensajeS25);

pthread_create( &thread6, NULL, (void *)&testAlmacenar, mensaje6);

pthread_create( &thread26, NULL, (void *)&testRetirar, mensajeS26);
pthread_create( &thread27, NULL, (void *)&testRetirar, mensajeS27);

pthread_create( &thread7, NULL, (void *)&testAlmacenar, mensaje7);
pthread_create( &thread8, NULL, (void *)&testAlmacenar, mensaje8);
pthread_create( &thread9, NULL, (void *)&testAlmacenar, mensaje9);
pthread_create( &thread10, NULL, (void *)&testAlmacenar, mensaje10);
pthread_create( &thread11, NULL, (void *)&testAlmacenar, mensaje11);
pthread_create( &thread12, NULL, (void *)&testAlmacenar, mensaje12);

pthread_create( &thread28, NULL, (void *)&testRetirar, mensajeS28);
pthread_create( &thread29, NULL, (void *)&testRetirar, mensajeS29);
pthread_create( &thread30, NULL, (void *)&testRetirar, mensajeS30);
pthread_create( &thread31, NULL, (void *)&testRetirar, mensajeS31);
pthread_create( &thread32, NULL, (void *)&testRetirar, mensajeS32);

pthread_join(thread1, NULL);
pthread_join(thread2, NULL);
pthread_join(thread3, NULL);
pthread_join(thread4, NULL);
pthread_join(thread5, NULL);
pthread_join(thread6, NULL);
pthread_join(thread7, NULL);
pthread_join(thread8, NULL);
pthread_join(thread9, NULL);
pthread_join(thread10, NULL);
pthread_join(thread11, NULL);
pthread_join(thread12, NULL);
pthread_join(thread21, NULL);
pthread_join(thread22, NULL);
pthread_join(thread23, NULL);
pthread_join(thread24, NULL);
pthread_join(thread25, NULL);
pthread_join(thread26, NULL);
pthread_join(thread27, NULL);

pthread_join(thread28, NULL);
pthread_join(thread29, NULL);
pthread_join(thread30, NULL);
pthread_join(thread31, NULL);
pthread_join(thread32, NULL);

free(mensajeS21);
free(mensajeS22);
free(mensajeS23);
free(mensajeS24);
free(mensajeS25);
free(mensajeS26);
free(mensajeS27);
```

```
    free(mensajeS28);
    free(mensajeS29);
    free(mensajeS30);
    free(mensajeS31);
    free(mensajeS32);

    exit(0);
}


void verBuffer(){
    printf("| ");
    int i;
    for(i=0; i<MAX; i++)
        printf("%c | ", buffer[i]);

    printf("\n");

}

void testAlmacenar(void * v){
    almacenar(v);
}

void testRetirar(void * v){
    retirar(v);
}

void *almacenar(void *val)
{
    pthread_mutex_lock( &buff_mutex );
    if(n == MAX)
    {
        pthread_cond_wait( &buff_vacio_cond, &buff_mutex );
   }
    char *mensaje = (char *)val;
    buffer[in] = *mensaje;
    in++;
    if(in == MAX)
        in = 0;

    n++;

    pthread_mutex_unlock( &buff_mutex );

    verBuffer();

    printf(">>>Mandando Signal a consumidor\n");
    pthread_cond_signal( &buff_lleno_cond );

}

void *retirar(void *val)
{
    pthread_mutex_lock( &buff_mutex );
    if(n == 0)
    {
```

```
        pthread_cond_wait( &buff_lleno_cond, &buff_mutex );
    }
     *((char *)val) = buffer[out];
     buffer[out] = 0;
     out++;
     if(out == MAX)
         out = 0;

     n--;
     pthread_mutex_unlock( &buff_mutex );

     verBuffer();

     printf("\t\t\t\t\t\t\t<<<Mandando Signal a productor\n");
     pthread_cond_signal( &buff_vacio_cond );
}
```

1.6. El kernel Linux

Antes de resolver esta sección instale los siguientes paquetes en la máquina virtual:

- `make`: utilidad para mantener grupos de programas.
- `linux-headers-<version>`: headers del kernel de Linux.

Sustituya `<version>` por el resultado del comando `uname -r`.

1.6.1. Funcionamiento del kernel Linux

1. Describa la administración del procesador utilizada por defecto en el kernel Linux.

   La administración del procesador de Linux utiliza el método de round robin. Este sistema permite que varios procesos puedan estar ejecutandose al mismo tiempo (hablando en sentido de procesos concurrentes, a menos de que se cuente con 2 procesadores, nunca se va a poder tener 2 procesos ejecutandose a la vez), lo que quiere decir que la CPU esta dividida en ‘‘partes", cada uno para un proceso particular. Si un de ellos no termina para cuando su porción de tiempo (quantum) expira, un intercambio de procesos puede ocurrir (en caso de que haya alguno en cola de espera). Esta administración utiliza interrupciones por tiempo que se generean cada vez que el quantum transcurrio. Debido a todo esto, no es necesario código adicional dentro de los procesos para que la CPU pueda mantenerlos corriendo concurrementemente.

   La política de administración se basa en enumerar procesos de acuerdo a su prioridad dentro del sistema. Hay algorítmos complicados para definir el orden en el cual los procesos deben ser ejecutados para optimizar el rendimiento, pero siempre se mantiene el hecho de que a cada proceso se le asigna un número que corresponde a que tan apropiado es de ser asignado a la CPU.

   La asignación de prioridades es dinamica dentro de Linux. Cuando a un proceso se le nego por mucho tiempo el uso de la CPU, se le incrementa su prioridad. Lo mismo pasa si un proceso se ejecuta por demasiado tiempo, como penalidad se le decrementa su prioridad. Gracias a esto se evita que un proceso se mantenga inactivo por mucho tiempo o que consuma demasiado los ciclos del procesador.

2. Describa la administración de memoria utilizada por defecto en el kernel Linux.

   Cuando el procesador ejecuta un programa, esta leyendo una instrucción desde la memoria y la decodifica. Por lo general, esa instrucción necesita alguno datos, por

lo que el procesador debera realizar un nuevo acceso a memoria para poder darselos al proceso mismo. De esta manera, el procesador esta siempre accediendo a memoria para traer datos o instrucciones. Utilizando memoria virtual, todas las direcciones a las cuales el procesador accede no son físicas, las cuales permiten que, junto con un set de tablas dentro del sistema operativo, se pueda convertir estas direcciones virtuales en direcciones reales.

Como hay menos memoria física que la memoria virtual, el sistema debe de tener mucho cuidado en no usar la memoria física ineficientemente. Cuando un proceso intenta acceder a una dirección virtual que no se encuentra dentro de las asignadas a él, el procesador no puede encontrar una tabla de pagina cuya entrada sea la direccion pedida. Esto quiere decir que el proceso esta pidiendo una dirección que no le corresponde, por lo que el sistema lo terminara (al proceso).

Linux usa paginación por demanda como sistema de administración de memoria. Carga imagenes ejecutables a la memoria virtual de un proceso para su utilizacion. Cada vez que un comando se ejecuta, el archivo que lo contiene es abierto y sus contenidos son mapeados a la memoria virtual del proceso. Esto se conoce con el nombre de mapeo de memoria. Cuando el proceso se ejecuta, solo una parte de este esta en memoria física, el resto esta dentro del disco (el dispositivo que se utiliza como memoria virtual), por lo que el sistema usara el mapa de memoria para ir dandole al proceso toda la información necesaria para su ejecución. Si un proceso necesita volcar una página virtual dentro de la memoria física, y no hay suficiente espacio en ella, el sistema debe conseguir lugar para esta página, descartando cualquier otra de la misma. A veces, la página que se va a descartar no sufrió cambio alguno, por lo que no debe ser guardada. Sin embargo, si la página fue modificada, el sistema debe preservar los contenidos de esa página para que pueda ser leida mas tarde. Este tipo de página que fue modificada se guarda dentro de un archivo conocido como "swap file". Linux utiliza LRU (Least Recently Used) para definir que página sera quietada para poner otra nueva. Mientras mas sea utilizada una determinada página, tendrá menos posibilidades de ser swapeada que una página que se utiliza menos.

3. Describa el sistema de archivos utilizado en la distribución de Linux que instaló en la máquina virtual. ¿Qué capas existen en el kernel Linux para soportar sistemas de archivos sobre dispositivos de bloques?

   Dentro de los sistemas Linux, se cuenta con un sistema de archivos virtual, que es el encargado de manejar los filesystems dentro del sistema. Éste se conoce mas por sus siglas en ingles: VFS (Virtual File System). El concepto es poder separar los filesystems de bajo nivel con el resto del Kernel, para así no tener que repensar el Kernel para querer cambiar el sistema de archivos sobre el cual esta montado. De esta manera, será posible un manejo de los archivos eficaz sin imortar el sistema que se utilize. Es decir, Linux puede utilizar cualquier filesystem (ext2, ext3, fat, vfat, ufs, etc) gracias a el uso del VFS. El VFS brinda el set de instrucciones necesario y generales para todos los filesystems para poder usarlos a gusto del sistema (o del programador). Las estructuras atravéz de las cuales el VFS funciona son: los index-nodes, que son atravez de los cuales se mantiene un registro de los file; un caché de directorios para acelerar la búsqueda y mapeado de nombres de ficheros a sus correspondientes inodos y el super bloque, que es donde se carga cada filesystem para poder ser usado.

   La distribucion instalada dentro de este trabajo practico contiene a ext3 como su filesystem. Ext3 (third extended filesystem o "tercer sistema de archivos extendido") es un sistema de archivos con registro por diario (journaling). Este concepto se basa en llevar un journal o registro de diario en el que se almacena la información necesaria para restablecer los datos afectados por la transacción en caso de que ésta falle.

1.6.2. Módulos de kernel

El kernel de Linux permite ser ampliado en **runtime** con módulos. Los módulos son objetos de código compilado que pueden ser insertados en runtime al kernel, siendo linkeados contra el kernel al

momento de ser insertados. De esta manera puede ampliarser la funcionalidad del kernel en runtime, sin tener que incluir todo el código en el binario original.

1.6.3. Compilando un módulo de kernel

En esta sección omitimos el enunciado y vamos directo a lo que hicimos.

1.6.4. cambiarLuces.c

```
/* cambiarLuces.c
 *
 * Cambiador de luces del teclado usando un modulo del kernel
 *
 */
/* Headers para modulos de kernel */
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/proc_fs.h>
#include <linux/init.h>
#include <linux/tty.h>          /* fg_console, MAX_NR_CONSOLES*/
#include <linux/vt_kern.h>      /* fg_console, MAX_NR_CONSOLES*/
#include <linux/kd.h>           /* KDSETLED */
#include <linux/console_struct.h>   /* vc_cons*/

#define MODULE_NAME "lasluces"
#define ERROR -1

/* Prototipos de las funciones de inicializacion y destruccion */
static int __init luces_init(void);
static void __exit luces_exit(void);

static struct proc_dir_entry *mi_directorio, *mi_archivo;
struct tty_driver *mi_driver;

/* Informamos al kernel que inicialize el modulo usando luces_init
 * y que antes de sacarlo use luces_exit
 */
module_init(luces_init);
module_exit(luces_exit);

int usandoLaIOCTL(int valor){
/* Definidas en linux/kd.h
 * 0x1 LED_SCR el de Scroll Lock
 * 0x2 LED_NUM el de Numeric lock
 * 0x4 LED_CAP el de Caps Lock
 */
    /* Para los leds*/
    (mi_driver->ioctl) (vc_cons[fg_console].d->vc_tty,
        NULL, KDSETLED,valor);

    /* Para los flags, no la luz */
//  (mi_driver->ioctl) (vc_cons[fg_console].d->vc_tty,
//      NULL, KDSKBLED,valor);
    return 0;
}
```

```
int escribiendo(struct file *filp, const char __user *buff, unsigned long len, void *data){
    printk(KERN_ALERT "Me escribieron %s - y len es %lu\n", buff, len);
    if(len>2){
        printk(KERN_ALERT "Escribi'o mal las opciones, debe poner un entero entre 0 y 7 - Sale por l
        return len;
    }
    switch (buff[0]){
        case '0':
            printk(KERN_ALERT "Apagando todo\n");
            usandoLaIOCTL(0);
            break;
        case '1':
            printk(KERN_ALERT "LED_SCR on\n");
            usandoLaIOCTL(1);
            break;
        case '2':
            printk(KERN_ALERT "LED_NUM on\n");
            usandoLaIOCTL(2);
            break;
        case '3':
            printk(KERN_ALERT "LED_NUM + LED_SCR on\n");
            usandoLaIOCTL(3);
            break;
        case '4':
            printk(KERN_ALERT "LED_CAP on\n");
            usandoLaIOCTL(4);
            break;
        case '5':
            printk(KERN_ALERT "LED_CAP + LED_SCR on \n");
            usandoLaIOCTL(5);
            break;
        case '6':
            printk(KERN_ALERT "LED_CAP + LED_NUM on\n");
            usandoLaIOCTL(6);
            break;
        case '7':
            printk(KERN_ALERT "LED_CAP + LED_NUM + LED_SCR on\n");
            usandoLaIOCTL(7);
            break;
        default:
            printk(KERN_ALERT "Escribi'o mal las opciones, debe poner un entero entre 0 y 7\n");
            break;
    }

    return len;
}

/* Inicializacion */
static int __init luces_init() {
    int ret = 0;
    printk(KERN_ALERT "Estamos cargando el modulo para usted\n");

    mi_directorio = proc_mkdir(MODULE_NAME, NULL);
    if(mi_directorio == NULL)
    {
        printk(KERN_ALERT "Fallo!, no pude crear el directorio.\n");
```

```
        ret = -ENOMEM;
        goto no_dir;
    } else {
        printk(KERN_ALERT "DONE!, primer paso completado con 'exito\n");
    }

    mi_archivo = create_proc_entry("escribaAqui", 0644, mi_directorio);
    if(mi_archivo == NULL){
        printk(KERN_ALERT "Fallo la creaci'on del archivo\n");
        ret = -ENOMEM;
        goto no_file;
    } else {
        printk(KERN_ALERT "GOLAZO! Pudiste crear el archivo\n");
    }

    mi_archivo->write_proc = escribiendo;
    mi_driver = vc_cons[fg_console].d->vc_tty->driver;

    printk(KERN_ALERT "Si ves esto es que ya podes usarme.\n");
    return 0;

no_file:
    remove_proc_entry("escribaAqui", mi_directorio);
no_dir:
    remove_proc_entry(MODULE_NAME, NULL);
    return ret;
}
/* Destruccion */
static void __exit luces_exit() {
    printk(KERN_ALERT "Desmontando el m'odulo.\n");

    remove_proc_entry("escribaAqui", mi_directorio);
    remove_proc_entry(MODULE_NAME, NULL);
}

MODULE_LICENSE("GPL");
```

Para compilar este código deberá construir una **Makefile**. Este archivo Makefile es utilizado luego por el comando `make` para compilar el módulo con las opciones correctas. En el mismo directorio donde se encuentra `cambiarLuces.c` cree un archivo `Makefile` conteniendo:

1.6.5. Makefile

Luego ejecute:

`make`

Para compilar el módulo. Finalmente, pruebe insertar el módulo usando:

`insmod cambiarLuces.ko` esto también viene facilitado con el siguiente script

1.6.6. ./cargarKernel.sh

```
sudo insmod cambiarLuces.ko
```

Debería ver un mensaje en la consola indicando que se cargo correctamente o que falló.

Para hacer funcionar el módulo escribir como root en `/proc/lasluces/escribaAqui` o, nuevamente, con un script que toma un dato:

1.6.7. ./escribir.sh dato

```
sudo insmod cambiarLuces.ko
```

Para sacar el módulo:

`rmmod cambiarLuces.ko` o su versión script.

1.6.8. ./descargarKernel.sh

```
sudo rmmod cambiarLuces.ko
```

Esto también debería dar un mensaje indicando lo sucedido.

1.6.9. Un módulo propio

Escriba un módulo de kernel que permita controlar los LEDs del teclado **sin necesidad de escribir un programa en lenguaje C**. El módulo deberá permitir prender o apagar cada LED usando simplemente comandos del shell.

Dado que la máquina virtual provista por VirtualBox no muestra el estado de los LEDs, la cátedra provee abajo un programa que lo hace. Compile utilizando:

```
gcc -o check_kbleds check_kbleds.c
```

1.6.10. check_kbleds.c

```
_kbleds.c
```

Sugerencia Haga que su módulo cree un archivo en `/proc` y que las escrituras a ese archivo controlen los LEDs utilizando la IOCTL `KDSETLED` de la consola de Linux.

Referencias

[BUR97] Richard L. Burden, J. Douglas Faires - Numerical Analysis 6th Edition