

TRABAJOS PRÁCTICOS DE DISEÑO Y ADMINISTRACIÓN DE SISTEMAS OPERATIVOS

71013012

**Asignatura Obligatoria del 1^{er} semestre del 3^{er} Curso del
Grado en Ingeniería Informática de la UNED**

Curso 2016-2017



Fecha de entrega Primer trabajo: **hasta el 5 de Diciembre de 2016**

Fecha de entrega Segundo trabajo: **hasta el 9 de enero de 2017**

Contenido

INFORMACION GENERAL	3
Objetivo de los trabajos.....	3
Carácter de los trabajos	3
Requisitos.....	3
Formato y fecha de entrega de los trabajos	4
Informe	4
Evaluación de los trabajos.....	5
TRABAJO I: Monitorizando procesos.....	6
TRABAJO II: Comunicando procesos.....	13

INFORMACION GENERAL

Objetivo de los trabajos

Los trabajos prácticos de la asignatura Diseño y Administración de Sistemas Operativos se dividen en dos tareas (Trabajo I y Trabajo II).

El objetivo de estos trabajos es que el estudiante practique algunos de los conceptos básicos de la asignatura haciendo uso del **lenguaje C** y el lenguaje de **script Bash** usando como plataforma el sistema operativo **Linux**.

Carácter de los trabajos

Tal y como se señala en la guía de la asignatura, los trabajos prácticos **no son necesarios** para superar la asignatura, pero si son muy recomendables y cuentan un 20% de la nota final. Por ello, se recuerda al estudiante la obligación e importancia de hacer los trabajos por sí mismo **sin copiarlas** de otros compañeros, dado que ello repercutirá en perjuicio del propio estudiante, quien no adquirirá el grado formativo adecuado.

Por otro lado, se recomienda al estudiante que comience a hacer estos trabajos una vez se haya dado ya un **primer repaso** a los temas de la asignatura en los que se basa cada trabajo, y se hayan asimilado los conceptos básicos.

Requisitos

Debido a la existencia de una gran variedad de distribuciones de Linux y compiladores de C y para evitar problemas de compatibilidad de plataformas y arquitecturas, el equipo docente pone a disposición del estudiante una **máquina virtual** con todo lo necesario para la realización de los trabajos así como una **plantilla de ejemplo** que define la estructura y el formato del trabajo que el estudiante debe entregar. Tanto la máquina como la plantilla de referencia se encuentran en la misma página donde ha podido descargar este documento: <http://unedlabs.dia.uned.es/DyASO/>

Se usará únicamente la máquina virtual en el estado en el que se encuentra disponible para su descarga en la corrección de los trabajos y **no se valorarán** trabajos que no puedan compilarse o ejecutarse dentro de dicho entorno o que no se ajusten al formato de entrega.

Por este motivo el estudiante debe comenzar por instalar la máquina virtual. Las instrucciones de descarga e instalación de dicha máquina virtual se detallan en el **Apéndice A** de este documento.

Formato y fecha de entrega de los trabajos

Para asegurar la homogeneidad en la entrega se dispone de una plantilla denominada “Plantilla_DyASO.tar” dentro de la cual aparece un ejemplo de cómo deben entregarse los trabajos. Dicha plantilla debe descomprimirse con permisos de **SUPERUSUARIO** (**sudo tar xf Plantilla_DyASO.tar**) con el fin de que se preserven los propietarios y los permisos de los archivos que contiene.

Los trabajos deberán ser entregados a través de la plataforma del curso virtual aLF en el apartado reservado para tal efecto en la planificación del curso. Cada uno de los dos trabajos deberá enviarse dentro de un fichero **.tar** con la misma estructura que el ejemplo de referencia así como con el informe de cada trabajo en formato PDF. El nombre de los ficheros debe ajustarse a la siguiente estructura:

```
DyASO_PED1_Apellido1_Apellido2_Nombre.tar
DyASO_PED2_Apellido1_Apellido2_Nombre.tar
```

Donde **Apellido1** es el primer apellido del alumno, **Apellido2** el segundo apellido y **Nombre** el nombre del alumno, por ejemplo

```
DyASO_PED1_Chaos_Garcia_Dictino.tar
```

Informe

Además de los archivos y ficheros fuente que componen cada trabajo, el estudiante deberá entregar un **informe** de cada trabajo en formato PDF que describa el trabajo realizado. Para ello debe seguirse la plantilla del informe proporcionada.

NOTA: Antes de ponerse a trabajar y para evitar descuidos es muy recomendable que el estudiante cambie los datos del ejemplo de referencia (*número del trabajo, nombre, apellidos, DNI, centro asociado en el que se ha matriculado y teléfono de contacto*) por los suyos propios en todas las partes donde aparezcan (nombre de los archivos y carpetas así como portada del informe de cada práctica).

NOTA2: No deben usarse **en ningún caso** ficheros **.zip** para realizar la entrega, ya que los ficheros .zip no almacenan los permisos de ejecución de los archivos en Linux, haciendo que el funcionamiento de la práctica pueda ser incorrecto.

Evaluación de los trabajos

Los trabajos, suponen un 20% de la nota final de la asignatura en caso de que el alumno **tenga como mínimo un 4.5 en el examen**. En caso contrario se habrá suspendido la asignatura. No obstante, la nota media de los trabajos se mantiene para la convocatoria de septiembre.

En la valoración del trabajo se tendrá en cuenta en primer lugar el funcionamiento de los programas. En caso de que no se ejecuten o no compilen el trabajo práctico no se calificará. Tampoco se evaluarán los trabajos que no presenten informe de prácticas en PDF.

¡¡Aviso importante!!, Las prácticas se realizan **individualmente**, no se aceptan grupos de trabajo. Tanto los profesores-tutores como el equipo docente se reservan el derecho de ponerse en contacto en caso de duda con el estudiante y realizarle diferentes cuestiones relativas a las prácticas para verificar que efectivamente es el autor de las mismas y no las ha copiado. La detección de una práctica copiada obligará al equipo docente a ponerlo en conocimiento del Servicio de Inspección de la UNED. Tampoco se permite el intercambio de código a través de los foros de la asignatura u otros medios.

TRABAJO I: Monitorizando procesos

En los sistemas operativos de tipo UNIX existe un comando llamado `top` que permite monitorizar la ejecución de otros procesos. Si se ejecuta `top` en un terminal se muestra una pantalla como la siguiente:

```
top - 17:51:25 up 1 day, 54 min,  2 users,  load average: 0.01, 0.03, 0.05
Mem:    507544k total,  478888k used,    28656k free,    17708k buffers
Swap:      0k total,      0k used,      0k free,   156292k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 23302 root        20   0   238m  74m  16m  S   5.3  15.1   4:24.40 Xorg
 28949 sistemas  20   0  90852  15m  11m  S   3.3   3.1   0:02.69 gnome-terminal
 23484 sistemas  20   0  51828 3432 1172  S   0.3   0.7   0:01.54 gnome-session
 23580 sistemas  20   0   128m 5740 2608  S   0.3   1.1   0:27.22 metacity
 23599 sistemas  20   0   115m 6248 2504  S   0.3   1.2   0:01.77 nm-applet
 23605 sistemas  20   0  95108 3712  760  S   0.3   0.7   0:01.00 bluetooth-apple
 23606 sistemas  20   0  58040 2204  244  S   0.3   0.4   0:00.90 gnome-fallback-
 24049 sistemas  20   0  72220 8388 4580  S   0.3   1.7   0:02.40 notify-osd
 29014 sistemas  20   0   2856 1160  888  R   0.3   0.2   0:01.67 top
     1 root        20   0   3668 1420  728  S   0.0   0.3   0:00.51 init
     2 root        20   0        0     0     0  S   0.0   0.0   0:00.00 kthreadd
     3 root        20   0        0     0     0  S   0.0   0.0   0:04.02 ksoftirqd/0
     5 root        20   0        0     0     0  S   0.0   0.0   0:00.22 kworker/u:0
     6 root        RT   0        0     0     0  S   0.0   0.0   0:00.00 migration/0
     7 root        RT   0        0     0     0  S   0.0   0.0   0:01.59 watchdog/0
     8 root         0 -20        0     0     0  S   0.0   0.0   0:00.00 cpuset
     9 root         0 -20        0     0     0  S   0.0   0.0   0:00.00 khelper
    10 root        20   0        0     0     0  S   0.0   0.0   0:00.00 kdevtmpfs
    11 root         0 -20        0     0     0  S   0.0   0.0   0:00.00 netns
```

Esta pantalla se actualiza periódicamente con los datos y estadísticas de uso de procesador de cada uno de los procesos. Para terminar la ejecución del comando `top` basta con pulsar la tecla `Q` (quit, o sea salir). Para más opciones de este comando así como para entender cada uno de los campos que se muestran puede consultarse el manual (`man top`)

Objetivo:

El Objetivo de esta práctica es programar un script en `bash` llamado `mitop.sh` que muestre información sobre los procesos de forma similar a la mostrada por comando `top`. **Obviamente el script no podrá utilizar el comando `top` para hacer su trabajo.**

El sistema de ficheros de procesos procfs:

Llegados a este punto hay que hacerse una pregunta, ¿De dónde sacamos la información?

El sistema operativo UNIX expone la mayoría de su funcionalidad a través del servicio de archivos, y esto no es una excepción. Toda la información sobre los procesos, así como el acceso a su memoria y a sus estructuras de control puede hacerse desde el sistema de archivos *procfs* (process file system) que se encuentra montado en la carpeta */proc*.

Para comprobarlo hagamos `cd /proc` y a continuación listemos su contenido con `ls`:

```
1      229      23621  23834  28959  768      driver      partitions
10     23       23629  23843  29023  8        execdomains  sched_debug
1013   23299    23632   23852  29026  811      fb           schedstat
1029   23301    23636   23857  3       816      filesystems  scsi
1071   23302    23638   23872  328     825      fs          self
11     23393    23645   23886  34      826      interrupts  slabinfo
1124   23473    23651   23912  344     828      iomem       softirqs
12     23484    23653   23938  36      837      ioports     stat
1286   23531    23655   23992  37      838      irq         swaps
13     23539    23657   24      39      839      kallsyms    sys
1372   23545    23659   24010  479     858      kcore       sysrq-trigger
14     23547    23663   24049  485     874      key-users   sysvipc
15     23550    23700   25      5       9       kmsg        timer_list
1558   23551    23703   26      593     acpi       kpagecount  timer_stats
16     23572    23714   26189  6       asound     kpageflags  tty
1609   23574    23716   26904  61      buddyinfo  latency_stats uptime
1610   23580    23731   27306  667     bus        loadavg     version
17     23586    23739   27576  670     cgroups    locks       version_signature
18     23591    23750   28351  684     cmdline    mdstat      vmallocinfo
19     23592    23752   28667  694     consoles   meminfo     vmstat
1921   23597    23756   28815  7       cpuinfo    misc        zoneinfo
2      23598    23757   28871  706     crypto     modules
207    23599    23783   28874  707     devices    mounts
208    23602    23790   28947  713     diskstats  mtrr
21     23605    23791   28949  714     dma        net
22     23606    23798   28958  736     dri        pagetypeinfo
```

Pueden apreciarse muchas carpetas con un nombre numérico así como archivos con información general del sistema. Para ver más detalles puede hacerse `ls-l`. A continuación se muestra un extracto de lo que se obtiene:

```
dr-xr-xr-x  8 root      root          0 mar 18 13:37 1
...
```

```

dr-xr-xr-x  8 sistemas  sistemas          0 jun 30 16:18 23299
dr-xr-xr-x  8 sistemas  sistemas          0 mar 18 13:37 23301
...
-r--r--r--  1 root      root              0 jul 22 17:54 cpuinfo
-r--r--r--  1 root      root              0 jul 22 17:54 devices
...
-r--r--r--  1 root      root              0 ene  9  2015 mdstat
-r--r--r--  1 root      root              0 jul 22 17:54 meminfo
...

```

En esta salida pueden apreciarse varias cosas:

- 1) En primer lugar vemos que existen varios directorios cuyo nombre es un número. Por ejemplo `23299` es un directorio del usuario `sistemas`. Este directorio contiene información sobre el proceso con `PID=23299` que pertenece al usuario `sistemas` y grupo `sistemas`.

En general existe un directorio numérico por cada proceso que hay en el sistema de modo que haciendo `ls -l` es posible determinar que procesos existen y a que usuario pertenecen.

Para obtener más información sobre un proceso en particular, es posible entrar en el directorio y ver su contenido. Para ello, siguiendo con el ejemplo, puede hacerse `cd 23299` y a continuación `ls`

```

attr      comm      fd      maps      ns      root      stat
autogroup coredump_filter fdinfo    mem      oom_adj    sched      statm
auxv      cpuset      io      mountinfo oom_score  schedstat  status
cgroup     cwd         latency  mounts    oom_score_adj sessionid  syscall
clear_refs environ     limits  mountstats pagemap    smaps      task
cmdline    exe         loginuid net        personality stack       wchan

```

En el interior del directorio existen ficheros con información sobre el proceso, por ejemplo el fichero `stat` cuyo contenido es (`cat stat`):

```

23299 (pulseaudio) S 1 23298 23298 0 -1 4202560 6958 0 605 0 92 1875 0 0 20 0
3 0 3921087 112537600 1082 4294967295 134512640 134588932 3220732064
3220731232 3078296612 0 0 3674112

```

Del contenido del fichero podemos extraer fácilmente, alguna información, el primer dato de `stat` es el `PID` del proceso, el segundo el nombre del comando ejecutado que dio lugar al proceso (`pulseaudio`).

La misma información puede encontrarse en varios sitios con presentaciones diferentes. Por ejemplo es posible ver comando completo con el que fue invocado haciendo `cat cmdline`:

```
/usr/bin/pulseaudio--start--log-target=syslog
```

Y también es posible encontrar esta información en el contenido de `status`, (`cat status`) que aporta la información en un formato más legible:

```
Name: pulseaudio
State: S (sleeping)
Tgid: 23299
Pid: 23299
PPid: 1
TracerPid: 0
...
voluntary_ctxt_switches: 3244
nonvoluntary_ctxt_switches: 4282
/usr/bin/pulseaudio--start--log-target=syslog
```

Además de los directorios que tienen como nombre el PID del proceso que contienen, en `/proc` es posible obtener información general del sistema. Por ejemplo volviendo al dicho directorio (`cd /proc`) y viendo el contenido de `meminfo` (`cat meminfo`):

```
MemTotal:      507544 kB
MemFree:       16924 kB
Buffers:       18240 kB
Cached:        154384 kB
SwapCached:      0 kB
Active:        329936 kB
...
```

Un análisis pormenorizado de todos los datos que se exponen en `procfs` puede encontrarse consultando manual de linux <http://man7.org/linux/man-pages/man5/proc.5.html>

mitop: Para emular la funcionalidad básica del comando top deberá crearse un script ejecutable *Shell scrip* de *Bash* llamado `mitop.sh` que realice las siguientes operaciones:

- 1) Leer el contenido del directorio `/proc` obteniendo los PIDs de todos los procesos presentes en el sistema. Una posible forma de hacerlo es:
 - a. Obteniendo la lista de procesos con `ls -l`.
 - b. Filtrando todas las líneas que no terminen por un número (puede por ejemplo usarse `awk` y una expresión regular).
- 2) Para cada PID obtenido se leerá el contenido de algunos archivos presentes en la carpeta `/proc/PID` para obtener el tiempo de ejecución en modo usuario y núcleo de dicho proceso (Para ello será necesario averiguar previamente usando la documentación presente en el manual de procfs (`man proc`) dónde está cada dato buscado). Se sumarán ambos datos para obtener el tiempo total de uso del procesador desde que se inició el proceso.
- 3) Se esperará un segundo (`sleep`).
- 4) Se volverá a leer el tiempo en modo usuario y núcleo de cada proceso y a calcular el tiempo de uso del procesador. Se comparará con lo obtenido en el paso 2 y partir de la diferencia entre los tiempos que el proceso ha estado ejecutándose (antes y después de la pausa de 1s) puede calcularse el porcentaje de uso del procesador de cada uno de los procesos (ojo con las unidades).
- 5) Se ordenarán los procesos según el porcentaje (en sentido decreciente) de uso de CPU.
- 6) Se mostrará información estadística general y a continuación información acerca de los 10 procesos que tienen mayor utilización del procesador. A continuación (a diferencia del comando top) termina la ejecución del programa sin esperar la pulsación de ninguna tecla.

La información **mínima** que `mitop` debe mostrar para considerarse correcto es:

Cabecera:

Número de procesos, uso total de la cpu, memoria total, utilizada y libre.

Por cada proceso:

PID: Pid del proceso

USER: Usuario que invoca el proceso

PR: Prioridad

VIRT: Tamaño de la memoria virtual del proceso

S: Estado del proceso

%CPU: porcentaje de uso del procesador

%MEM: Porcentaje de uso de la memoria

TIME: tiempo de ejecución del proceso

COMMAND: Nombre del programa invocado

El archivo **Ejercicio1.sh** es el archivo que deberá demostrar el funcionamiento del programa `mitop.sh`, se trata de un script ejecutable escrito en *bash* de nombre `Ejercicio1.sh` que realizará las siguientes acciones:

- 1) Crea un proceso en segundo plano (yes).
- 2) Ejecuta el archivo `mitop.sh` que se encuentra en la carpeta `./Trabajo1`
- 3) Elimina el proceso creado.

No es necesario que el estudiante **modifique** dicho fichero, pero si debe utilizarlo para comprobar el funcionamiento de su implementación de top. Si todo funciona correctamente el proceso “yes” debería aparecer en la posición más alta de top.

Consejo: Puede resultar de ayuda repasar la redirección de entrada y salida y consultar el manual de los comandos `awk`, `cat`, `grep`, `read` y `sleep` y antes de hacer el ejercicio. Existe una gran cantidad de información en Internet acerca de la programación en *bash* que puede ser de ayuda como por ejemplo

<http://www.gnu.org/software/bash/manual/bash.pdf>

<http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html>

https://es.wikibooks.org/wiki/El_Manual_de_BASH_Scripting_B%C3%A1sico_para_Principiantes

http://www.tldp.org/LDP/Bash-Beginners-Guide/html/sect_04_02.html

Datos de entrega:

El archivo `Ejercicio1.sh` se encontrará en la raíz del fichero `.tar` que contiene la práctica junto con el informe en PDF. Dentro de este mismo fichero debe existir un directorio llamado `Trabajo1` que contendrá el fichero con permisos de ejecución `mitop.sh` desarrollado por el estudiante.

Todos estos ficheros se encuentran `Plantilla_DyASO.tar`, el estudiantes tan solo deberá modificar su nombre así como el contenido del fichero `mitop.sh`

Además de la entrega del ejecutable es necesario describir **detalladamente** el funcionamiento del programa entregado en el *informe en PDF*.

NOTA: En cualquier proyecto de software un código organizado, comentado y bien documentado es esencial. Eso es necesario en todos los ámbitos, incluso cuando se crea un

programa para uso personal. Pero resulta **imprescindible** cuando la persona que ha de leer y entender el código no es la misma que la persona que lo ha escrito. Por tanto se agradecerá y valorará positivamente que el código esté *ordenado, limpio, bien estructurado* y sobre todo *bien comentado*.

Por tanto en la medida de lo posible evítense comandos crípticos, úsense nombres de variables comprensibles, créense funciones si es necesario para dividir el código, elimine el código muerto de versiones anteriores y sobre todo use los comentarios para indicar lo que está haciendo.

TRABAJO II: Comunicando procesos

Este ejercicio consiste en transmitir información entre varios procesos a través de los mecanismos que se muestran en la Figura 1. Para ello deberán implementarse los siguientes archivos de código en C: `fuentes1.c`, `fuentes2.c` y `fuentes3.c`. Además deberá implementarse un fichero de script `Ejercicio2.sh` que compile dichos archivos fuente y genere los ejecutables `Ej1`, `Ej2` y `Ej3` respectivamente.

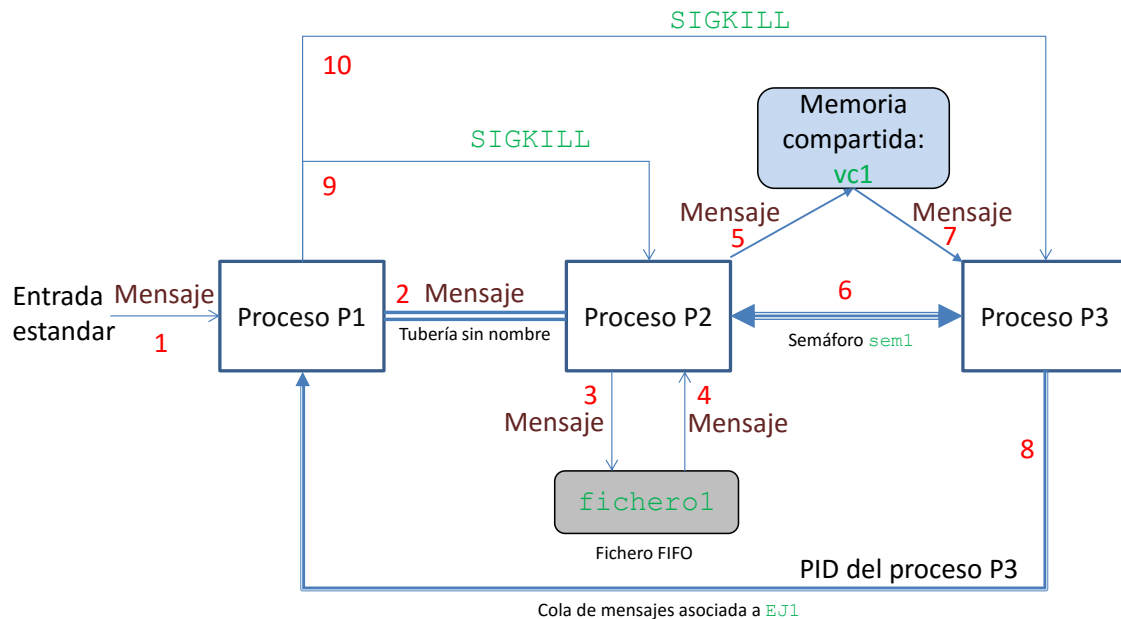


Figura1: Esquema de comunicación y sincronización entre los procesos.

Dichos ejecutables deberán realizar las siguientes acciones:

Ej1: Este archivo ejecutable creará un proceso P1 que usará `fork()` para crear un hijo P2, ambos compartirán una tubería sin nombre. El proceso P1 pedirá al usuario que introduzca un mensaje por teclado y se lo transmitirá al proceso P2 a través de dicha tubería. Una vez hecho esto, el proceso P1 creará una cola de mensajes asociada al archivo `Ej1` bajo una cierta clave y se quedará a la espera de un mensaje.

Cuando llegue un mensaje al proceso P1 a través de dicha cola, extraerá del mismo el PID que contiene (que corresponde al proceso P3 que se describirá posteriormente) y a continuación, enviará una señal `SIGKILL` a su hijo P2 y al proceso cuyo PID ha recibido (P3). Finalmente borrará el fichero `fichero1`, mostrará sus estadísticas de uso de CPU y finalizará su ejecución.

Por otra parte, el proceso P2 leerá la tubería sin nombre y una vez que tenga el mensaje creará un fichero FIFO de nombre `fichero1` y escribirá en él dicho mensaje. A continuación ejecutará el programa `Ej2` con la llamada al sistema `exec()` de tal manera que el proceso P2 pasará a ser una instancia del ejecutable `Ej2`.

Ej2: Este programa ejecutable abrirá el fichero FIFO `fichero1` y leerá de él el mensaje que le ha dejado el antiguo proceso P2.

A continuación creará una región de memoria compartida y dentro de ella una variable `vc1` en la cual almacenar el mensaje, así como un semáforo `sem1` para proteger el acceso a dicha región. Tanto `vc1` como `sem1` estarán asociados al `fichero1`.

A continuación se bifurcará mediante `fork()` creando un proceso hijo P3.

El proceso P2 se mantendrá a la espera durante un segundo y después escribirá el mensaje en la variable compartida `vc1`. Posteriormente hará una operación `V()` para abrir el semáforo y suspenderá su ejecución con la llamada al sistema `pause()`.

Por su parte el proceso P3 hará una llamada al sistema `exec()` sobre el ejecutable `Ej3` que pasará a ser el proceso P3.

Ej3: Este ejecutable cargará la región de memoria compartida y la variable asociada `vc1` así como el semáforo `sem1` que ha creado P2 asociados al archivo `fichero1`.

A continuación realizará una operación `P()` sobre el semáforo `sem1` para esperar a que el proceso P2 acabe de escribir en la memoria compartida. Cuando lo haya hecho leerá la región de memoria compartida y mostrará el mensaje por pantalla.

Una vez terminada la operación, abrirá la cola de mensajes asociada al ejecutable `Ej1` y le enviará un mensaje al proceso P1 para indicarle que ha terminado. Dicho mensaje deberá contener su PID. Hecho esto se quedará a la espera mediante la llamada al sistema `pause()`.

Nota: Cada vez que uno de los procesos envíe o reciba un mensaje deberá indicarlo escribiendo en la salida estándar. Dicha escritura indicará el PID del proceso que escribe, el proceso en

curso (P1, P2 ó P3), el ejecutable que da lugar a dicho proceso (Ej1, Ej2 ó Ej3), el origen o destino del mensaje y el mecanismo de transmisión. Por ejemplo:

“El proceso P1 (PID=1247, Ej1) transmite un mensaje al proceso P2 por una tubería sin nombre”

El archivo **Ejercicio2.sh** deberá ser un script ejecutable escrito en el lenguaje de `shell` `bash` que realizará las siguientes acciones:

- 1) Compile usando gcc las fuentes que se encuentran en el directorio `./Trabajo2` creando los ejecutables *Ej1*, *Ej2* y *Ej3*.
- 2) Asigne los permisos necesarios a Ej1, Ej2 y Ej3.
- 3) Ejecute Ej1.
- 4) Borre todos los archivos ejecutables creados (pero no sus fuentes).

Este fichero *si* deberá ser modificado por el estudiante para compilar los ficheros fuente que el mismo ha creado

Datos de entrega:

El archivo `Ejercicio2.sh` se encontrará en la raíz del fichero `.tar` que contiene la práctica junto con el informe en PDF. Dentro de este mismo fichero debe existir un directorio llamado `Trabajo2` que contendrá los ficheros `fuentes1.c`, `fuentes2.c` y `fuentes3.c`. **No deberán entregarse los archivos ejecutables** Ej1, Ej2 y Ej3.

Además de la entrega del código fuente y el script `ejercicio2.sh` que se encarga de la compilación del mismo es necesario describir detalladamente el funcionamiento del programa entregado en el *informe en PDF*.

NOTA: En este caso al igual que en el programa anterior se agradecerá y valorará positivamente que el código esté *ordenado, limpio, bien estructurado* y sobre todo *bien comentado*.