TRABAJOS PRÁCTICOS
DE DISEÑO Y ADMINISTRACIÓN DE SISTEMAS OPERATIVOS

71013012

Asignatura Obligatoria del 1er semestre del 3er Curso del Grado en Ingeniería Informática de la UNED

Curso 2024-2025

Fecha de entrega Primer trabajo: hasta el **2 de diciembre de 2024**

Fecha de entrega Segundo trabajo: hasta el **13 de enero de 2025**

Contenido

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 el curso virtual.

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 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…)* 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, la nota media de los trabajos se mantendría si fuese necesario 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. Tampoco se permite el intercambio de código a través de los foros de la asignatura u otros medios y el código serán revisado para detectar similitudes.

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 calificar con cero ambas PED, en caso de reincidencia habrá que ponerlo en conocimiento del Servicio de Inspección de la UNED.

TRABAJO I: Pacto con el Diablo

En los sistemas operativos de tipo UNIX existe una categoría de procesos denominados procesos *demonio*. Se trata de procesos que no están asociados a ningún usuario (administrativamente el usuario al que pertenecen es root) que se emplean para realizar tareas periódicas de administración del sistema, como por ejemplo la gestión de procesos. Tanto el proceso *init* como *systemd* en distribuciones modernas son un ejemplo de este tipo de procesos.

Objetivo:

En este trabajo vamos a crear un sencillo gestor de procesos que consistirá en una interfaz de entrada de comandos CLI (*Common Language Infrastructure*) codificado en `Fausto.sh` y un proceso Demonio (`Demonio.sh`) que se encargará de gestionar los procesos en ejecución lanzados por Fausto, monitorizando su estado y reiniciándolos cuando sea necesario. Ambos procesos se comunicarán mediante el uso de tres ficheros que actuarán como listas: `procesos`, `procesos_servicio` y `procesos_periodicos` así como el directorio `Infierno` y los ficheros `SanPedro` y `Apocalipsis` que se describirán más adelante. Además, registrarán sus acciones en la `Biblia.txt`.

A continuación, describimos las dos partes del sistema:

Fausto.sh

Se trata de un shell script en bash cuya tarea es recibir órdenes por la línea de comandos e invocar al demonio (de ahí su nombre Fausto) con el fin de ejecutar los comandos adecuados.

Invocación del demonio. Cuando Fausto sea invocado y antes de realizar ningún comando comprobará la existencia de un proceso Demonio en ejecución. **Si dicho proceso no existe, reiniciará todas las estructuras, la Biblia y lanzará un proceso Demonio.**

Para ello en primer lugar borrará (si existen) los ficheros `procesos`, `procesos_servicio`, `procesos_periodicos`, `Biblia.txt`, `Apocalipsis` y `SanPedro` así como la carpeta `Infierno`.

Y volverá a crear los ficheros y carpetas vacíos con la única excepción de `Apocalipsis`. Todos los ficheros se crearán en la misma ruta que contiene al fichero `Fausto.sh`.

Para crear un proceso demonio se realizará un "*doble fork*" y se desasociará del terminal. Es decir, se creará un proceso bash que ejecutará un comando desasociado del terminal (ver `nohup`), se cerrará su entrada estándar y se redirija la salida y error a `/dev/null`.

El comando para ejecutar será simplemente `./Demonio &` que lanzará el proceso Demonio en segundo plano, al terminar el shell su ejecución sin esperar al proceso demonio, éste quedará adoptado por el proceso "*init*" (en la MV el que hace las funciones de *init* es *systemd*).

Para comprobar que el Demonio se ha creado correctamente podemos hacer

```
$ ps gl | grep [D]emonio
0   1000    33519     1817  20    0   12116   3348 do_wai S     pts/2
0:00 /bin/bash ./demonio.sh
```

Y conocido su pid (33519) podemos comprobar que efectivamente su padre 1817 es systemd

```
$ pstree -s 33519
systemd───systemd───demonio.sh───sleep
```

NOTA: Normalmente los procesos demonio están asociados a root, pero en este caso, por simplicidad y seguridad de ejecución no nos interesa usar sudo, podemos considerarlo un "demonio del usuario sistemas". Además, los procesos demonio suelen desasociarse del sistema de ficheros y otras complicaciones que por simplicidad no nos interesa tratar en esta práctica.

Una vez que el demonio haya sido lanzado se escribirá una entrada en la biblia indicando la hora de creación del mundo, del siguiente modo:

```
11:29:24 ---------------Génesis---------------
11:29:24 El demonio ha sido creado
```

Observe que cada entrada de la Biblia va siempre precedida de la marca temporal `hora:minuto:segundo` de tal forma que sea sencillo trazar los eventos que se han ido produciendo en el sistema.

Ejecución del comando: Una vez invocado el demonio es necesario llevar a cabo la ejecución de la orden asociada, los comandos permitidos y su sintaxis son los siguientes:

```
./Fausto.sh run comando
./Fausto.sh run-service comando
./Fausto.sh run-periodic T comando
./Fausto.sh list
./Fausto.sh help
./Fausto.sh stop PID
./Fausto.sh end
```

A continuación, veremos qué hace cada uno de ellos:

./ Fausto.sh run se encarga de *ejecutar un comando una sola vez*, para ello en primer lugar lanzará en segundo una instancia de bash pasándole como argumento el comando y creará una entrada en la lista de procesos con el *PID del proceso bash que ejecuta el comando** y el comando ejecutado. La lista es simplemente un fichero con una entrada por cada línea. Para evitar condiciones de carrera es imprescindible que las lecturas y escrituras de las listas sean atómicas (ver apartado sincronización). Cada vez que se cree un proceso se apuntará el nacimiento en la `Biblia.txt` junto con la hora en la que se produce dicho evento.

Ejemplo, si se ejecuta la orden

```
$ ./Fausto.sh run 'sleep 10; echo hola'
```

Y a continuación

```
$ ps -l
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S  1000  307413  307403  0  80   0 -  3577 do_wai pts/0    00:00:00 bash
0 S  1000  311904    1817  0  80   0 -  3029 do_wai pts/0    00:00:00 Demonio.sh
0 S  1000   10001    1817  0  80   0 -  3029 do_wai pts/0    00:00:00 bash
0 S  1000  312728   10001  0  80   0 -  2675 hrtime pts/0    00:00:00 sleep
0 S  1000  312764  311904  0  80   0 -  2675 hrtime pts/0    00:00:00 sleep
0 R  1000  312766  307413  0  80   0 -  3528 -      pts/0    00:00:00 ps
$ hola
```

Podemos ver que se ejecutará una vez `sleep 10; echo hola`. La primera instrucción (`sleep 10`) duerme durante 10 segundos. Cuando esta instrucción termina `echo` escribe "`hola`" por pantalla. El proceso que se ha lanzado es `bash` con pid `10001`
Y su hijo es sleep (con un pid distinto que no nos interesa).

En este ejemplo Fausto añadirá una entrada al final del fichero `procesos`:

```
10001 'sleep 10; echo hola'
```

Y una entrada en la Biblia.txt

```
13:57:16:  El proceso 10001 'sleep 10; echo hola' ha nacido.
```

**NOTA:* No es posible asignar un PID a un comando ya que cada comando puede lanzar varios procesos simultáneamente (por ejemplo, `ps | grep Demonio`). Por ese motivo el PID que nos interesa guardar es del intérprete asociado.

Hecho esto, Fausto retornará inmediatamente y a los 10 segundos se imprimirá "hola" por pantalla como podemos ver marcado en rojo en la última línea del ejemplo.

./Fausto.sh run-service comando

Ejecuta un *comando como servicio*, la diferencia con el caso anterior es que el comando y el PID *del proceso bash que lo ejecuta* se almacenarán en el fichero `procesos_servicio`. El proceso demonio usará esa información para resucitar el proceso cada vez que este muera.

Por ejemplo

```
$ ./Fausto.sh run-service "yes > /dev/null"
```

Crearía un proceso `yes` cuya salida se redirige a `/dev/null` y cuyo pid podría ser por ejemplo 10002, si por cualquier motivo este proceso fuese destruido el demonio volvería a lanzarlo. También se incluirá una entrada en la Biblia indicando la resurrección del proceso.

./Fausto.sh run-periodic T comando

Ejecuta una orden *con reinicio periódico*. Actúa de forma similar a los comandos anteriores, pero añade el proceso creado al fichero `procesos_periodicos` donde guardara los siguientes campos: `tiempo de arranque`, `periodo de ejecución T` y `comando`

Por ejemplo:

```
$ ./Fausto.sh run-periodic 10 “echo hola”
```

Ejecutará `echo hola` y creará una entrada en la tabla `procesos_periodicos` como la siguiente:

```
0 10 10003 “echo hola”
```

Donde el 0 es el tiempo que lleva el proceso ejecutándose, 10 es el periodo con el que debe ejecutarse el proceso (en segundos), 10003 es el PID del proceso que se ha creado y “echo hola” el comando. También apuntará el nacimiento en la Biblia.

Esta información permitirá al proceso demonio ejecutar el comando de forma periódica, de modo que se ejecutará el comando imprimiendo “hola” y cada 10 segundos el demonio volverá a ejecutarlo (pero no se verá nada ya que el demonio está des-asociado del terminal):

```
$ ./Fausto.sh run-periodic 10 ‘echo hola’
hola
$ retorna y puedo seguir escribiendo...
```

./Fausto.sh list mostrará el contenido de las listas listas `procesos`, `procesos_servicio` y `procesos_periodicos` por salida estándar.

./Fausto.sh help mostrará la lista de comandos disponibles y su sintaxis.

./Fausto.sh stop PID comprobará que el PID se corresponde con un proceso creado por Fausto o por el Demonio, esto es, que está en alguna de las listas. Si no estuviese mostrará un error y sugerirá usar “`./Fausto.sh list`” para obtener la lista de procesos. Si el proceso ha sido creado, indicará al demonio que debe destruirlo, para ello creará un fichero de nombre “`PID`” en la carpeta `./infierno` (path relativo a la ubicación del script).

Por ejemplo, si se ejecuta

```
./Fausto.sh stop 10003
```

Fausto comprobará que el proceso con PID 10003 está en alguna de las listas, en este caso la comprobación es positiva (sería el proceso “echo hola” que está en la lista

`procesos_periodicos`). Hecho esto, creará un fichero vacío `./infierno/10003`. El Demonio ya se encargará del resto.

./Fausto.sh end *iniciará el Apocalipsis*, es decir terminaría la ejecución de todos los procesos que hayan sido lanzados, incluido el Demonio. Para ello creará un fichero `Apocalipsis` en el directorio en el que se encuentra el script, el Demonio será el encargado de finalizar todos los procesos y a sí mismo.

Cualquier otro comando o una sintaxis incorrecta deberá indicar un error y sugerir que se use **./Fausto.sh help** para obtener la lista de comandos disponibles.

Por ejemplo:

```
$ ./Fausto.sh asdf 'echo hola'
Error, orden "asdf" no reconocida, consulte las órdenes disponibles con ./Fausto.sh help
```

Demonio.sh

El funcionamiento del proceso demonio es el siguiente.

En primer lugar, el demonio ejecutará un bucle hasta que llegue el Apocalipsis (es decir hasta que detecte un fichero `Apocalipsis` en el mismo directorio en el que se encuentra el script).

Cuando llegue el Apocalipsis el demonio terminará todos los procesos de todas las listas que todavía estén vivos, borrará las listas, así como el fichero `Apocalipsis` y el directorio `infierno` y terminará su ejecución. Los únicos ficheros que deberán quedar tras el Apocalipsis son los scripts y la `Biblia.txt`.

El Demonio creará una entrada Apocalipsis en la Biblia donde irá registrando cada proceso presente en alguna de las listas que ha terminado (tanto si lo ha terminado el Demonio como si ha finalizado por cualquier otro motivo) y finalizará escribiendo la frase "Se acabó el mundo.", por ejemplo:

```
11:29:36 ---------------Apocalipsis---------------
11:29:36 El proceso X ha terminado
```

```
11:29:36 El proceso Y ha terminado
…
11:29:36 Se acabo el mundo.
```

Mientras no llegue el Apocalipsis, el demonio se encontrará en un bucle que realizara las siguientes acciones:

1) Esperará un segundo
2) Recorrerá todas las listas para llevar a cabo las acciones oportunas

La acción que deberá realizar depende de la lista en particular y se describe a continuación:

`Lista procesos`

El proceso Demonio recorrerá esta lista y por cada entrada que exista en la misma extraerá el PID, si el PID coincide con algún fichero que se encuentre en el directorio `Infierno` terminará *todo el árbol de dicho proceso***, eliminará la entrada de la lista y el fichero correspondiente del infierno. (Nótese que cada fichero del infierno corresponde a un proceso eliminarse ya que ha sido detenido con la orden ./`Fausto.sh stop PID`).

***NOTA:* Recuerde que el pid almacenado en la lista es el del proceso bash que ejecuta el comando, pero este a su vez puede tener muchos hijos, nietos…etc, *hay que eliminar todos los procesos derivados del mismo. Una forma de hacerlo es usar* `pstree` *con la opción* `-p` *para obtener los procesos, filtrar con* `grep` *los números y utilizar enviar la señal* `SIGTERM` *a esa lista de procesos.*

Si el fichero no se encuentra en el infierno, comprobará que dicho proceso continúa en ejecución, si sigue activo no hará nada, en caso contrario lo eliminará de la lista.

Cada vez que un proceso muera (porque su PID está en el infierno o por que el proceso ha terminado) creará una entrada en el fichero `Biblia.txt` indicando que ese proceso ha terminado.

Por ejemplo, cuando termine el proceso 10001 el demonio lo detectará. Eliminará dicha entrada de la lista y escribirá en la Biblia:

```
13:57:26:  El proceso 10001 'sleep 10; echo hola' ha terminado.
```

Lista procesos_servicio

Se procede de forma similar al caso anterior, con la diferencia de que *si el proceso no está en el infierno y tampoco en ejecución se resucitará* y se añadirá una entrada en la Biblia indicándolo, ya que los procesos de servicio deben estar continuamente en ejecución.

Por ejemplo, si el usuario sistemas hace

```
$pkill yes
```

terminará el proceso `"yes > /dev/null"` que se había ejecutado como servicio. Cuando el Demonio lo detecte, lo reiniciará actualizando la entrada con el PID nuevo del proceso recién creado y a continuación creará una entrada en la Biblia como la siguiente

```
13:57:30 El proceso 10002 "yes > /dev/null" resucita con pid 10004
```

Lista procesos_periodicos

La operación de esta lista es similar a la de los procesos de servicio con la salvedad de que es necesario *calcular cuánto tiempo lleva el proceso en ejecución para ver si es necesario reiniciarlo o todavía hay que esperar*.

Para ello, una vez comprobado que el proceso no está en el infierno, se incrementará el contador de tiempo (primer campo de la entrada de la lista), si este es *igual o mayor al periodo* de ejecución (segundo campo de la lista) y, además, el proceso *no está en ejecución, se volverá a lanzar el proceso*. Hecho esto se actualizará la tabla con el PID nuevo, poniendo de nuevo a cero el contador, e indicándolo en la Biblia.

Si el proceso todavía se está ejecutando no hará nada con él, simplemente incrementará el contador.

Por ejemplo, cuando se lanzó el proceso "`echo hola`" con una periodicidad de 10 segundos en la tabla se añadió la siguiente entrada:

```
0 10 10003 "echo hola"
```

Al cabo de un segundo el demonio actualiza la entrada:

```
1 10 10003 "echo hola"
```

Otro segundo después volverá a actualizarla

```
2 10 10003 "echo hola"
```

…

A los 9 segundos la entrada será

```
9 10 10003 "echo hola"
```

Finalmente, a los 10 segundos el contador llegaría a 10, entonces se reiniciaría el proceso hola (que ha terminado hace tiempo, casi con total seguridad en el primer segundo) y se actualizaría la entrada

```
0 10 10005 "echo hola"
```

Y se añadirá una entrada en la Biblia

```
13:58:35 El proceso 10003 "echo hola" se ha reencarnado en el pid 10005.
```

Obsérvese que, a diferencia del caso anterior, en el mensaje de la Biblia se usa la palabra reencarnado para distinguirlo de los procesos resucitados de la lista de `procesos_servicio`.

Nota: Este mecanismo de conteo del tiempo no es muy preciso ya que se asume que el demonio ejecuta el bucle exactamente cada segundo, lo cual no es cierto, el script ejecuta cada iteración del bucle en "al menos" un segundo, (tardará algo más en procesar las listas y lanzar o destruir procesos), pero es suficiente para las pretensiones de este trabajo.

Sincronización

Tanto Fausto como el Demonio están leyendo/escribiendo en los mismos ficheros, esto podría dar lugar a condiciones de carrera, para evitarlo se utilizará el fichero `SanPedro`.

Dicho fichero actuará como llave impidiendo que Fausto y el Demonio ejecuten al mismo tiempo el acceso a secciones críticas (recursos comunes).

Para ello cada vez que el demonio realice una actualización de las listas utilizará un bloqueo *lock* usando el fichero `SanPedro` como llave que debe adquirir antes de realizar la acción, es decir:

```
flock SanPedro Comando_que_actualiza_ficheros
```

Del mismo modo cada vez que el proceso Demonio realice el procesamiento de las listas (paso 2 del bucle principal) o se encuentre llevando a cabo el Apocalipsis deberá realizar un bloqueo.

Es importante destacar que **todas las operaciones de actualización de la lista deben de estar en el miso bloqueo** para evitar que mientras se está actualizando la tabla Fausto pueda modificarla. Esto supone realizar un bloqueo de un grupo de (ver `man flock` para más detalles y cuidado con las opciones, `-n` no hace lo que queremos que haga, no copiéis el ejemplo sin entenderlo).

Fichero de ejecución y prueba

El archivo **Ejercicio1.sh** es el fichero que usará para comprobar el funcionamiento de los programas `Fausto.sh`, y `Demonio.sh`. Se trata de un script ejecutable escrito en *bash* de nombre `Ejercicio1.sh` que realizará varias invocaciones de Fausto y pruebas como las propuestas en los ejemplos.

El estudiante **no debe modificar** dicho fichero, pero si debe utilizarlo para comprobar el funcionamiento de su implementación, para ello debajo de cada línea se comenta el resultado esperado. Si todo funciona correctamente el resultado de ejecución será similar al siguiente:

```
$./Ejercicio1.sh
**********************************************************
1) Debería de haberse creado el proceso Demonio
la salida esperada es algo así
systemd───systemd───Demonio.sh───sleep
```

```
donde Demonio.sh no debe ser hijo de bash sino de systemd o
similar
************************************************************

systemd---systemd---Demonio.sh---sleep

************************************************************
2) Lanzo algunos comandos y compruebo que se han creado
Debería de haber un proceso normal
'sleep 10; echo hola > test1.txt'
Un proceso servicio 'yes > /dev/null'
y dos periódicos, el normal y el lento
Comparamos los procesos teóricamente lanzados y los que
realmente existen
************************************************************´

Procesos lanzados según Fausto:
./Fausto.sh list
***** Procesos normales *****
7791 'sleep 10; echo hola >> test1.txt'
***** Procesos servicio *****
7808 'yes > /dev/null'
***** Procesos periódicos *****
0 5 7820 'echo hola_periodico >> test2.txt'
0 5 7831 'echo hola_periodico_lento >> test3.txt; sleep 20'

Procesos existentes:
ps -l
F  S     UID     PID    PPID    C  PRI    NI  ADDR  SZ  WCHAN    TTY
TIME CMD
0 S 1000 6447 6440 0 80  0 -  2052 wait  pts/1   00:00:00 bash
0 S 1000 7767 6447 0 80 0-1675 wait pts/1 00:00:00
0 S 1000 7781 1360 0 80 0 - 1684 wait  pts/1 00:00:00 Demonio.sh
0 S 1000 7789 7781 0 80 0 -  1376 hrtime pts/1    00:00:00 sleep
0 S 1000 7791 1360 0  80 0 -  1672 wait   pts/1    00:00:00 bash
0 S 1000 7794 7791 0 80 0 -  1376 hrtime pts/1    00:00:00 sleep
0 S 1000 7808 1360 0 80  0 -  1672 wait   pts/1    00:00:00 bash
```

```
0 R 1000 7810 7808 0 80  0 -  1375 -      pts/1    00:00:00 yes
0 S 1000 7831 1360 0 80  0 -  1673 wait   pts/1    00:00:00 bash
0 S 1000 7834 7831 0 80 0 -  1376 hrtime pts/1    00:00:00 sleep
0 R  1000 7846 7767 0 80  0 -  2191 -      pts/1    00:00:00 ps

**********************************************************
3) Elimino manualmente el proceso yes sin avisar a Fausto.
El Demonio debería detectarlo y reiniciar el proceso:
**********************************************************
pkill yes
bash: línea 1:  7810 Terminado               yes > /dev/null

./Fausto.sh list
***** Procesos normales *****
7791 'sleep 10; echo hola >> test1.txt'
***** Procesos servicio *****
7854 'yes > /dev/null'
***** Procesos periódicos *****
4 5 7820 'echo hola_periodico >> test2.txt'
4 5 7831 'echo hola_periodico_lento >> test3.txt; sleep 20'

**********************************************************
4) Elimino el proceso usando Fausto.
El Demonio NO debe reiniciar el proceso:
**********************************************************
./Fausto.sh stop 241040
241040
./Fausto.sh list
***** Procesos normales *****
7791 'sleep 10; echo hola >> test1.txt'
***** Procesos servicio *****
***** Procesos periódicos *****
7 5 7831 'echo hola_periodico_lento >> test3.txt; sleep 20'
ps -l
F  S     UID     PID    PPID    C  PRI    NI  ADDR  SZ  WCHAN    TTY
TIME CMD
0 S 1000 6447 6440 0 80  0 -  2052 wait  pts/1   00:00:00 bash
```

```
0 S 1000 7767 6447 0 80 0-1675 wait pts/1 00:00:00
0 S 1000 7781 1360 0 80 0 - 1684 wait  pts/1 00:00:00 Demonio.sh
0 S 1000 7789 7781 0 80 0 -  1376 hrtime pts/1    00:00:00 sleep
0 S 1000 7791 1360 0  80 0 -  1672 wait   pts/1    00:00:00 bash
0 S 1000 7794 7791 0 80 0 -  1376 hrtime pts/1    00:00:00 sleep
0 S 1000 7831 1360 0 80  0 -  1673 wait   pts/1    00:00:00 bash
0 S 1000 7834 7831 0 80 0 -  1376 hrtime pts/1    00:00:00 sleep
0 R  1000 7846 7767 0 80  0 -  2191 -      pts/1    00:00:00 ps

************************************************************
5) Error de sintaxis provocado para que Fausto nos avise:
************************************************************
./Fausto.sh asdf
Error, orden "asdf" no reconocida, consulte las órdenes
disponibles con ./Fausto.sh help

************************************************************
6) Siguiendo la sugerencia anterior veríamos la ayuda:
************************************************************
./Fausto.sh help
sintaxis:
./Fausto.sh run comando
./Fausto.sh run-service comando
./Fausto.sh run-periodic T comando
./Fausto.sh list
./Fausto.sh help
./Fausto.sh stop PID
./Fausto.sh end

************************************************************
7) Terminamos la ejecución y vemos los mensajes enviados
por los procesos lanzados en los ficheros test 1, 2 y 3
************************************************************

hola
hola_periodico
hola_periodico
```

```
hola_periodico_lento

************************************************************
8) Comprobamos que no hay procesos sin terminar.
Esperamos que sólo salgan bash Ejercicio1.sh y ps
************************************************************

  PID TTY          TIME CMD
 6447 pts/1    00:00:00 bash
 7767 pts/1    00:00:00 Ejercicio1.sh
 7854 pts/1    00:00:00 bash
 8085 pts/1    00:00:00 ps

************************************************************
9) Comprobamos que no hay ficheros basura.
Solo deben quedar Fausto.sh, Demonio.sh y la Biblia.txt
************************************************************

Biblia.txt  Demonio.sh  Fausto.sh

************************************************************
10) Comprobamos que no hay bloqueos pendientes
no debería de salir nada:
************************************************************


************************************************************
11) Finalmente mostramos la Biblia
************************************************************

11:29:24 ---------------Génesis---------------
11:29:24 El demonio ha sido creado
11:29:24 El proceso 7791 'sleep 10; echo hola >> test1.txt' ha
nacido
11:29:24 El proceso 7808 'yes > /dev/null' ha nacido
11:29:24 El proceso 7820 'echo hola_periodico >> test2.txt' ha
nacido
```

```
11:29:24 El proceso 7831 'echo hola_periodico_lento >>
test3.txt; sleep 20' ha nacido
11:29:25 El proceso 7808 resucita con pid 7854
11:29:28 El proceso 7854 ha terminado
11:29:29 El proceso 7820 se reencarna con pid 7945
11:29:34 El proceso 7791 ha terminado
11:29:36 ---------------Apocalipsis---------------
11:29:36 El proceso 7831 ha terminado
11:29:37 El proceso 7945 ha terminado
11:29:36 Se acabo el mundo.
```

Observe que la mayoría de los test son auto-explicativos pero algunos deben interpretarse con calma.

En particular obsérvese que en la prueba 2) el proceso 7820 (`'echo hola_periodico >> test2.txt'`) se ejecuta inmediatamente y termina, por ese motivo no está en el listado emitido por `ls -l`, eso es normal y dependiendo de la planificación podría, o no, aparecer en la lista. Sin embargo, **el resto de los procesos** (7799, 7808 y 7831) **sí que han de aparecer** en el listado. Además obsérvese que el nombre de dichos procesos no es muy informativo (bash) lo importante es fijarse en los PIDs.

Obsérvese también que dicho proceso **7820 ha de reencarnarse a los 5s** como efectivamente está registrado en la Biblia (salida de la prueba 11 en rojo).

Por otra parte, en **la prueba 3) se termina el proceso 7854, ese proceso ha de resucitarse** ya que es un proceso de servicio, así muestra también la Biblia.

Por otra parte la resucitación del proceso 7808 es el proceso 7854, puesto que este proceso se termina con Fausto a los tres segundos (tal como muestra la Biblia) no vuelve a reencarnarse. **De hecho Ningún otro proceso debe de ser resucitado ni reencarnado** ya que el proceso periódico lento 7831 tarda 20 segundos en terminar y para entonces ya ha empezado el Apocalipsis.

Además, es importante observar que cuando el proceso 7791 termina el Demonio lo detecta y lo registra en la Biblia.

Finalmente, el Apocalipsis debe de terminar todos los procesos que no lo han hecho por sí mismos, es decir todos excepto el 7791 y no debe quedar ningún proceso o fichero basura tras finalizar el Apocalipsis.

Hay que tener encuenta que **la planificación de los procesos en el ejercicio del estudiante puede diferir respecto a este ejemplo**, no se pretende que sea exactamente igual pero sí que pase todos los test y que sea coherente, es decir: que se resuciten los procesos que han de resucitar, que no resuciten los terminados por Fausto, que se reencarnen los procesos periódicos que han terminado transcurrido su tiempo T, que no se resucite un proceso que no está muerto, que se detención los procesos que terminan y que el Apocalipsis termine todos los procesos que quedasen por terminar dejando un entorno limpio.

Consejo: Puede resultar de ayuda repasar la redirección de entrada y salida y consultar el manual de los comandos `awk`, `cat`, `cut`, `grep`, `read`, `flock` 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

https://atareao.es/tutorial/scripts-en-bash/

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

https://www-howtogeek-com.cdn.ampproject.org/v/s/www.howtogeek.com/788955/how-to-validate-the-syntax-of-a-linux-bash-script-before-running-it/amp/?amp_js_v=a6&amp_gsa=1&usqp=mq331AQIKAGwASCAAgM%3D#aoh=16493140540983&csi=0&referrer=https%3A%2F%2Fwww.google.com&amp_tf=De%20%251%24s&ampshare=https%3A%2F%2Fwww.howtogeek.com%2F788955%2Fhow-to-validate-the-syntax-of-a-linux-bash-script-before-running-it%2F

https://morningcoffee.io/killing-a-process-and-all-of-its-descendants.html

Datos de entrega:

La carpeta `DyASO_PED1_Apellido1_Apellido2_Nombre` deberá contener el archivo `Ejercicio1.sh` junto con el informe en PDF. Dentro de esta misma carpeta debe existir un directorio llamado `Trabajo1` que contendrá los ficheros con permisos de ejecución `Fausto.sh` y `Demonio.sh` desarrollados por el estudiante. No deberá de haber más ficheros en dicha carpeta.

Toda la carpeta se empaquetará en un `tar` de nombre `DyASO_PED1_Apellido1_Apellido2_Nombre.tar` que deberá entregarse en el curso virtual.

En la plantilla `Plantilla_DyASO.tar` se encuentra la estructura correcta, así como un ejemplo de entrega. El estudiante tan solo tiene que modificar su nombre, el contenido de los ficheros `Fausto.sh` y `Demonio.sh` y redactar *el informe en PDF* donde describa detalladamente el funcionamiento del programa entregado.

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*.
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.

Evite también copiar y pegar fragmentos de código de internet (ya que serán detectados como **plagio**), escriba su propio código desde cero.

TRABAJO II: Combate de procesos

Este ejercicio consiste en un combate entre varios procesos hijos que será arbitrada por el proceso padre.

Para ello deberán implementare dos archivos de código en C: `padre.c` e `hijo.c`. Además deberá implementarse un fichero de script `Ejercicio2.sh` que compile dichos archivos fuente y genere los ejecutables `PADRE` e `HIJO`, cree un fichero FIFO `resultado` y limpie todos los ficheros creados al finalizar el combate.

El funcionamiento de los procesos es el siguiente:

El proceso padre P que se obtiene al ejecutar el ejecutable `PADRE` creará una clave asociada al propio fichero ejecutable (nombre que obtendrá del primer argumento de entrada) y a una letra (por ejemplo '`X`').

A partir de esta clave generará varios mecanismos *IPC*, en particular creará una cola de mensajes `mensajes`, una región de memoria compartida `lista` (que enlazará con una array con capacidad para N PIDs), un semáforo `sem` que usará para proteger el acceso a dicha variable compartida y una tubería sin nombre `barrera`.

A continuación creará `N` procesos hijos ($H_1 \ldots H_N$), donde N es un parámetro que se pasada como entrada al invocarlo desde `Ejecicio2.sh`. Cada hijo realizará un `exec()` del ejecutable `HIJO`.

Cada uno de estos procesos hijo H_i re-establecerá los mecanismos IPC para comunicarse con su padre (es necesario reabrirlos* ya que al hacer `exec()` el código del padre se sobrescribe con el código del hijo). A continuación, los distintos hijos se atacaran por rondas, en cada ronda hay dos fases "preparación" y "ataque".

El proceso padre, justo antes de iniciar cada ronda mantendrá una lista con los PIDs de los procesos "vivos" en el array `lista` ubicado en la región de memoria compartida. El acceso a dicha región por el padre y los hijos deberá protegerse mediante un semáforo `sem` que será usado en cada acceso a la misma.

Para sincronizar el inicio de cada ronda se usará una "barrera" en la que el padre imprimirá "`Iniciando ronda de ataques`" por salida estándar y avisará a los contendientes para

que se preparen enviando K bytes a la tubería barrera (uno por cada hijo que quede vivo). Los hijos esperarán la barrera leyendo de la tubería barrera.

En la fase de preparación, cada proceso hijo decidirá aleatoriamente si "ataca" o "defiende" e iniciará la variable estado que controla el resultado de la contienda a la cadena "OK".

Los procesos que se defiendan instalarán la función defensa en SIGURS1 y dormirán durante 0.2 segundos**, mientras que los atacantes instalarán la función indefenso y esperarán 0.1 segundos antes de iniciar su ataque.

La función defensa, al ser invocada mostrará un mensaje diciendo "El hijo Hi ha repelido un ataque" (donde H_i es su PID) y establecerá la cadena estado al valor "OK".

La función indefenso, por el contrario, al ser invocada escribirá "El hijo Hi ha sido emboscado mientras realizaba un ataque" y establecerá estado a "KO".

Transcurridos los 0.1 segundos de la fase de preparación, los procesos que estén en modo de ataque elegirán aleatoriamente un PID válido de la lista, esto es, uno que sea distinto de cero (y del suyo propio por razones obvias). Seguidamente imprimirán por salida estándar "Atacando al proceso H_i", enviarán a dicho proceso una señal SIGURS1 y esperarán otros 0.1 segundos a que termine la fase de ataques.

Terminada la ronda de ataques y defensas cada proceso hijo enviará un mensaje al padre usando la cola de mensajes llamada mensajes, indicando su PID y el resultado de la contienda (cadena estado) y quedara a la espera de una siguiente ronda (esperando en la barrera).

Por su parte, el padre leerá los K mensajes recibidos y comprobará el resultado de la contienda. A cada proceso que esté "KO" le enviará una señal SIGTERM, esperará a que finalice (wait), pondrá a cero su PID en el array lista y actualizará la variable K que controla el número de procesos vivos.

Si después de la ronda quedan dos o más contendientes, el padre iniciará una nueva ronda de ataques, en caso contrario terminarán las rondas.

Al terminar las rondas, si queda un solo hijo con vida, el padre lo finaliza con `SIGTERM` esperando a que termine su ejecución, a continuación escribirá "`El hijo Hi ha ganado`" en el fichero FIFO `resultado` y finalmente liberaría todos los recursos IPC.

Si no quedan hijos vivos (se han matado todos mutuamente) el padre escribirá "`Empate`" en el fichero FIFO `resultado` y liberará los recursos.

Antes de terminar, el padre *demostrará* que los recursos IPC se han liberado utilizando la llamada al sistema `system` para invocar la utilidad `ipcs` con las opciones apropiadas para mostrar *solamente las colas de mensajes y los semáforos* (En general puede haber regiones de memoria compartida en uso, es normal y se cierran solas cuando el proceso termina pero no deben quedar colas de mensajes ni semáforos abiertos).

NOTA: Para aumentar la legibilidad deben agregarse saltos de línea después de cada mensaje. Además, si se estima conveniente, también pueden imprimirse mensajes adicionales para aclarar lo que está ocurriendo en los procesos. Por último, para evitar condiciones de carrera en las escrituras conviene que tras escribir en la salida estándar, se realice un volcado inmediato del buffer de la salida estándar (`fflush`).

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

1) Compilará usando gcc las fuentes `padre.c` e `hijo.c` que se encuentran en el directorio `./Trabajo2` creando los ejecutables `PADRE` e `HIJO`.
2) Creará un FIFO `resultado`
3) Lanzará un proceso `cat` en segundo plano a la espera de leer el resultado del fichero FIFO `resultado`.
4) Ejecutará `PADRE` pasándole como argumento el número `10` (para crear 10 contendientes).
5) Al terminar la ejecución del comando `cat` borrará todos los archivos ejecutables creados y el fichero FIFO `resultado`, dejando solamente los ficheros fuentes `padre.c` e `hijo.c`.

***PISTA:** El lector atento se habrá dado cuenta de que conociendo la clave es fácil recuperar la región de memoria compartida, la cola de mensajes y el semáforo, pero ¿Cómo recuperamos el descriptor de lectura de la tubería?

La respuesta es simple, el descriptor se hereda, basta con saber cuál es su número. Hay muchas formas de hacerlo. Una forma sencilla es indicándoselo al hijo como argumento de entrada.

Otra forma interesante para los más curiosos es redirigir el extremo de lectura de la tubería a la entrada estándar. Para ello basta con cerrar (`close`) el descriptor de la entrada estándar y luego realizar un `dup` o `dup2` sobre `barrera[0]`. Entonces podría leerse de la tubería como si fuese la entrada estándar. Este es el mecanismo usado por el shell para redirigir las entradas y salidas de sus procesos hijos usando tuberías.

****PISTA:** `sleep` no sirve, ya que su argumento de entrada es el número entero en segundos que el proceso ha de dormir. No obstante, buscando en el manual (`man`), puede encontrarse fácilmente una alternativa que permite esperar tiempos más cortos.

Datos de entrega:

La carpeta `DyASO_PED2_Apellido1_Apellido2_Nombre` deberá contener el archivo `Ejercicio2.sh` junto con el *informe en PDF*. Dentro de esta misma carpeta debe existir un directorio llamado `Trabajo2` que contendrá los ficheros `padre.c` e `hijo.c`. *No* deberán entregarse los archivos *ejecutables* `PADRE` e `HIJO` ni los ficheros FIFO.

Toda la carpeta se empaquetará en un `tar` de nombre `DyASO_PED2_Apellido1_Apellido2_Nombre.tar` y que deberá entregarse en el curso virtual.

De nuevo hay que ajustarse al formato de ejemplo que se proporciona en la plantilla. Modificando los ficheros `padre.c` e `hijo.c` y el script `ejercicio2.sh` que se encarga de la compilación del mismo. Además, 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*.