Trabajo de Diseño y Administración de Sistemas Operativos

Alumno: Isabel Valentina Manzaneque Núñez
DNI: 53902577-F
Centro Asociado: Londres
Teléfono de contacto: +447762347351
Email: imanzaneq3@alumno.uned.es

Primera PED

Introducción

En los sistemas operativos UNIX existe una clase especial de procesos conocidos como "procesos demonio", críticos para realizar tareas de administración del sistema. En esta práctica, exploramos el funcionamiento de estos procesos a través de la implementación de un gestor de procesos consistente en una interfaz de entrada de comandos y un proceso demonio asociado a ella.

La interfaz de entrada, Fausto.sh, recibirá órdenes por la línea de comandos y lanzará una serie de procesos que el proceso demonio, Demonio.sh, se encargará de gestionar, monitorizando su estado y realizando ciertas acciones cuando sea necesario. La comunicación y sincronización entre ambos se producirá a través de listas y ficheros los que se registrarán las diferentes acciones que se van realizando: las listas “procesos”, “procesos_servicio” y “procesos_periodicos” registrarán los correspondientes tipos de procesos en ejecución; el directorio “Infierno” contendrá archivos que identificaran aquellos procesos que se desea finalizar; el fichero “SanPedro” permitirá la sincronización entre Fausto y el Demonio; la aparición del fichero “Apocalipsis” indicará que se debe finalizar la ejecución del sistema; todas las acciones se irán recogiendo en la “Biblia.txt”

Implementación

Se va a dividir esta sección en las dos partes en las que se divide el sistema

Fausto.sh

Shell script en bash cuya tarea es recibir órdenes por la línea de comandos e invocar al demonio con el fin de realizar las acciones necesarias.

Fausto comienza comprobando si el proceso Demonio está en ejecución utilizando pgrep para buscar coincidencias con su nombre. Si el proceso Demonio no está en ejecución, realizará las siguientes acciones:

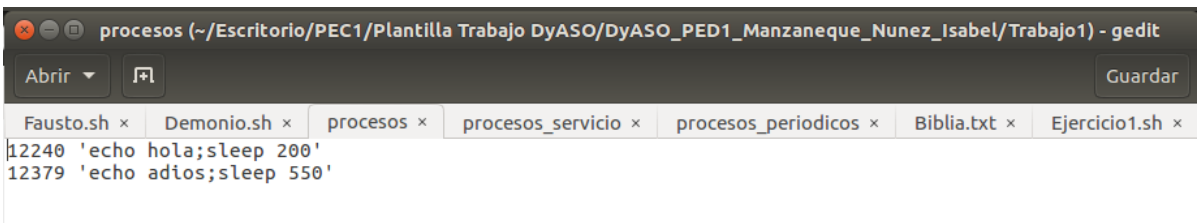
- Borrar los ficheros y carpetas residuales que puedan haber quedado en la ruta donde se encuentra Fausto.sh de ejecuciones anteriores. El borrado se realiza utilizando la opción -f (--force) ya que así no nos dará error si los archivos que se intentan borrar no existen. Esto es muy conveniente en caso de que, más que comprobar si están, lo que queremos es asegurarnos de que no están.
- Vuelve a crear los ficheros y directorios vacíos en la misma que se encuentra Fausto.sh para que estén listos para la próxima ejecución. El fichero Apocalipsis es el único que no se vuelve a crear, ya que esto significaría que se quiere concluir la ejecución del gestor de procesos.
- Lanza el proceso Demonio en segundo plano utilizando nohup, que lo desasociará del terminal, cerrará su entrada estándar y redirigirá la salida y error a /dev/null.
- Crea la entrada génesis en la biblia

```
# Comprueba si proceso Demonio existe
if ! pgrep -x "Demonio.sh" > /dev/null
then
    echo "----------Comenzando----------"
    # Borra ficheros y carpetas residuales y los vuelve a crear vacios
    rm -f procesos procesos_servicio procesos_periodicos Biblia.txt Apocalipsis SanPedro
    rm -fr Infierno
    touch procesos procesos_servicio procesos_periodicos Biblia.txt SanPedro
    mkdir Infierno
    # Lanza el Demonio en segundo plano
    nohup ./Demonio.sh >/dev/null 2>&1 &
    # Entrada genesis en la biblia
    flock SanPedro -c "{
                        echo \"$(date +%H:%M:%S) ---------------Génesis---------------\"
                        echo \"$(date +%H:%M:%S) El demonio ha sido creado\"
                       } >> ./Biblia.txt"

fi
```

Tras realizar las comprobaciones iniciales, Fausto lee la orden y parámetros dados por el usuario y deberá realizar una serie de acciones en consecuencia. Esto se ha llevado a cabo con una sentencia case en la que se comparará el primer argumento ($1) con una serie de casos que se verán a continuación. En todos estos casos, se comprobará que el número de parámetros proporcionados por el usuario es correcto y, en caso contrario, se le dejará saber que se ha producido un error en los parámetros introducidos.

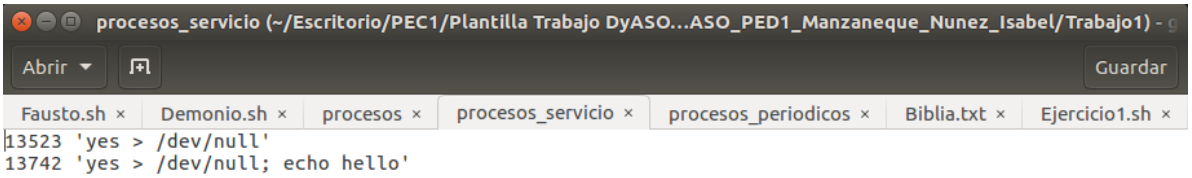
- **run**: Ejecuta una orden una única vez lanzando una instancia de bash en segundo plano y pasándole como argumento el comando ($2). Al lanzar esta orden, podremos conseguir el PID del proceso asociado ($!), que utilizaremos para realizar dos actualizaciones en las listas: Primero, se crea una nueva entrada en la lista de procesos con el PID y el comando, con el formato: *PID 'comando'*. Seguidamente, se escribe una nueva entrada en la biblia que indica que el proceso ha nacido: *HH:MM:SS El proceso PID 'comando' ha nacido.* Ambas escrituras se realizan previo bloqueo del fichero SanPedro, de manera que ningún otro proceso podrá conseguir el bloqueo de este hasta que la actualización actual haya terminado y sea liberado automáticamente. Todas las actualizaciones en las listas de procesos y la biblia se llevarán a cabo de esta manera para sincronizar a Fausto y el Demonio e impedir condiciones de carrera u otras inconsistencias.

```
case "$1" in
    "run")
        if [ "$#" -eq 2 ]
        then
            # Crea una nueva entrada en la lista de procesos y en la biblia
            comando="$2"
            bash -c "$comando" &
            pidBash="$!"

            flock SanPedro -c "echo \"$pidBash '$comando'\" >> ./procesos"
            flock SanPedro -c "echo \"$(date +%H:%M:%S) El proceso $pidBash '$comando' ha nacido.\" >> ./Biblia.txt"
        else
            echo "Error! $1 admite un parametro"
        fi
        ;;
```



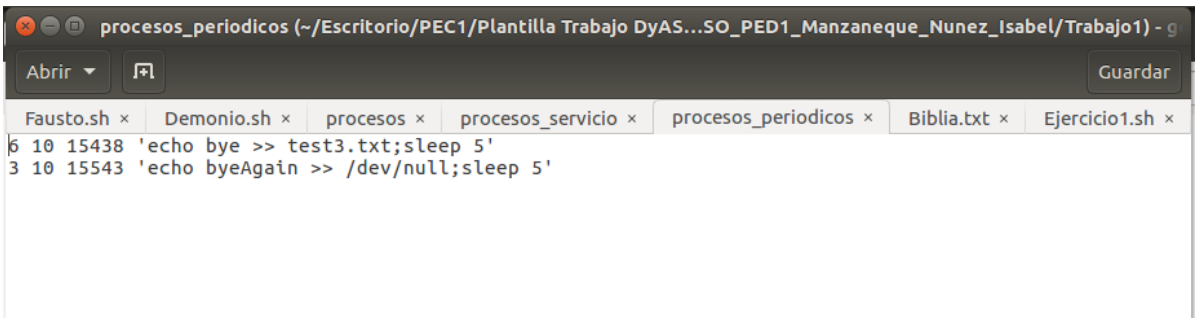
- **run-service**: Ejecuta un proceso como servicio. La implementación de este tipo de órdenes es como la de run, con la única diferencia de que esta vez se actualiza la lista de procesos_servicio y la Biblia, ya que cuando el demonio itere por las listas de procesos realizará diferentes acciones en función de en qué lista se encuentre dicho proceso. El formato de actualización de las listas es el mismo que el anterior y dichas actualizaciones realizan también bloqueos sobre el fichero SanPedro.

```
"run-service")
    # Crea una nueva entrada en la lista de procesos servicio y en la biblia
    if [ "$#" -eq 2 ]
    then
        comando="$2"
        bash -c "$comando" &
        pidBash="$!"

        flock SanPedro -c "echo \"$pidBash '$comando'\" >> ./procesos_servicio"
        flock SanPedro -c "echo \"$(date +%H:%M:%S) El proceso $pidBash '$comando' ha nacido.\" >> ./Biblia.txt"
    else
        echo "Error! $1 admite un parametro"
    fi
    ;;
```

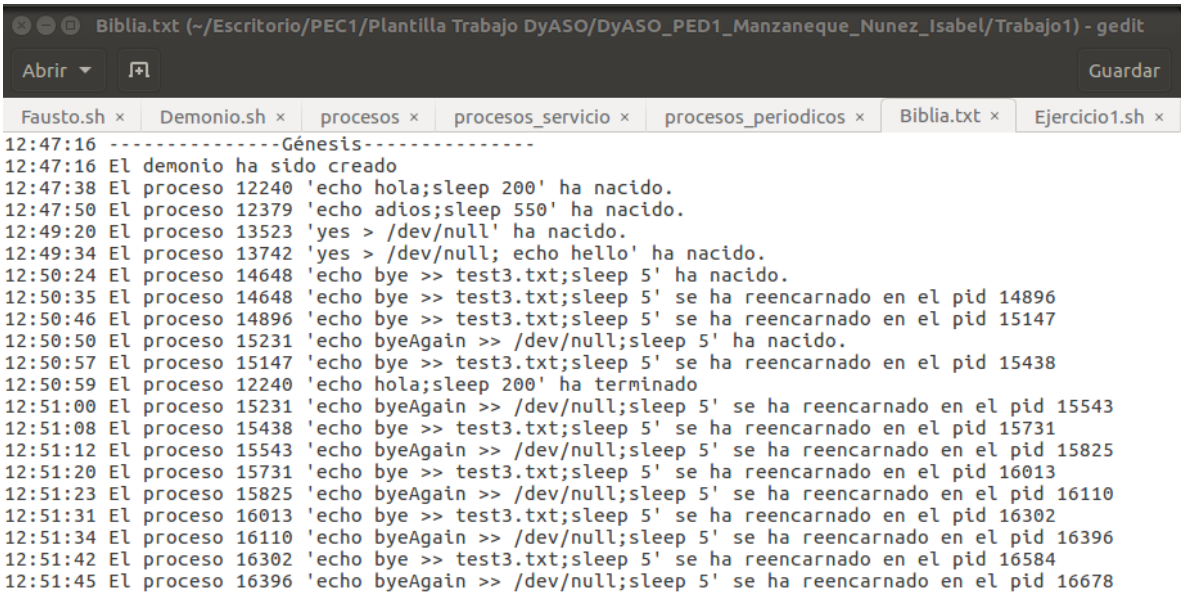


- **run-periodic**: Ejecuta una orden con un reinicio periódico, que el usuario debe especificar como parámetro. Utilizaremos el periodo T especificado por el usuario ($2) y el comando ($3) para actualizar la lista de procesos_periodicos, a la que además añadiremos un 0 como primer elemento de la entrada, que será el contador del tiempo que lleva el proceso en ejecución. La nueva entrada que se creará en la lista tendrá el siguiente formato: *0 T PID 'comando'*. Al igual que en los casos anteriores, también se creará una entrada en la biblia siguiendo el formato anterior y ambas actualizaciones se realizarán con sincronización realizando un bloqueo sobre el fichero SanPedro.

```
    ;;
"run-periodic")
    # Crea una nueva entrada en la lista de procesos periodicos y en la biblia
    if [ "$#" -eq 3 ]
    then
        T="$2"
        comando="$3"
        bash -c "$comando" &
        pidBash="$!"

        flock SanPedro -c "echo \"0 $T $pidBash '$comando'\" >> ./procesos_periodicos"
        flock SanPedro -c "echo \"$(date +%H:%M:%S) El proceso $pidBash '$comando' ha nacido.\" >> ./Biblia.txt"
    else
        echo "Error! $1 admite un parametro"
    fi
    ;;
```



Podemos ver todas estas entradas en la biblia:



- **list**: List mostrará por consola una lista con los procesos actualmente en ejecución

```
"list")
    # Muestra una lista de los procesos creados
    if [ "$#" -eq 1 ]
    then
        echo "***** Procesos normales *****"
        cat procesos
        echo "***** Procesos servicio *****"
        cat procesos_servicio
        echo "***** Procesos periodicos *****"
        cat procesos_periodicos
    else
        echo "Error! $1 admite un solo parametro"
    fi
    ;;
```

```
sistemas@DyASO:~/Escritorio/PEC1/Plantilla Trabajo DyASO/DyASO_PED1_Manzaneque_Nunez_Isabel/Trabajo
1$ ./Fausto.sh list
***** Procesos normales *****
12379 'echo adios;sleep 550'
***** Procesos servicio *****
13523 'yes > /dev/null'
13742 'yes > /dev/null; echo hello'
***** Procesos periodicos *****
9 10 16867 'echo bye >> test3.txt;sleep 5'
6 10 16961 'echo byeAgain >> /dev/null;sleep 5'
sistemas@DyASO:~/Escritorio/PEC1/Plantilla Trabajo DyASO/DyASO_PED1_Manzaneque_Nunez_Isabel/Trabajo
1$
```

- **help**: La opción help muestra al usuario la sintaxis de los comandos disponibles

```
"help")
    # Muestra los comandos disponibles
    echo "Sintaxis:"
    echo "./Fausto.sh run comando"
    echo "./Fausto.sh run-service comando"
    echo "./Fausto.sh run-periodic T comando"
    echo "./Fausto.sh list"
    echo "./Fausto.sh help"
    echo "./Fausto.sh stop PID"
    echo "./Fausto.sh end"
    ;;
```

```
sistemas@DyASO:~/Escritorio/PEC1/Plantilla Trabajo DyASO/DyASO_PED1_Manzaneque_Nunez_Isabel/Trabajo
1$ ./Fausto.sh help
----------Comenzando----------
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
sistemas@DyASO:~/Escritorio/PEC1/Plantilla Trabajo DyASO/DyASO_PED1_Manzaneque_Nunez_Isabel/Trabajo
1$
```

- **stop**: A la opción stop se le pasará como parámetro el PID de un proceso que se desea finalizar. Utilizando grep, se comparará este PID con cada entrada de cada lista de procesos y, si existe alguna coincidencia, se creará un nuevo fichero en la carpeta Infierno de nombre el PID del proceso que se desea eliminar. El Demonio se encargará del resto (más detalles en la implementación del Demonio)

```
"stop")
    # Si existe el proceso, crea un archivo con su pid en Infierno
    if [ "$#" -eq 2 ]
    then
        pid="$2"
        if grep -q "$pid" "procesos" || grep -q "$pid" "procesos_servicio" || grep -q "$pid" "procesos_periodicos"
        then
            touch ./Infierno/"$pid"
        else
            echo "Error! No existe el proceso $pid. Consulte la lista de procesos con './Fausto.sh list'"
        fi
    else
        echo "Error! $1 debe recibir un parametro"
    fi
    ;;
```

- **end**: Esta opción creará un nuevo fichero llamado Apocalipsis en la ruta en la que se encuentra Fausto y el Demonio se encargará del resto (más detalles en la implementación del Demonio)

```
    "end")
        # Crea el fichero Apocalipsis
        touch Apocalipsis
        ;;
    *)
        echo "Error! No existe la orden '$1'. Consulte las órdenes disponibles con ./Fausto.sh help"
        exit 1
        ;;
esac
```

Como se puede apreciar en la captura de pantalla anterior, si el usuario proporciona una orden que no se encuentra entre las anteriores, se mostrará un mensaje de error y se indicará al usuario que consulte las órdenes disponibles con el comando help

```
sistemas@DyASO:~/Escritorio/PEC1/Plantilla Trabajo DyASO/DyASO_PED1_Manzaneque_Nunez_Isabel/Trabajo
1$ ./Fausto.sh cocinarPasta "echo hola"
Error! No existe la orden 'cocinarPasta'. Consulte las órdenes disponibles con ./Fausto.sh help
sistemas@DyASO:~/Escritorio/PEC1/Plantilla Trabajo DyASO/DyASO_PED1_Manzaneque_Nunez_Isabel/Trabajo
1$
```

Demonio.sh

La implementación del demonio se compone de dos partes: la primera es un bucle principal en el que realizará una serie de acciones y del que no saldrá mientras no detecte el fichero Apocalipsis; la segunda comienza cuando se detecta el fichero Apocalipsis y se deberán realizar acciones para dejar el sistema en un estado consistente antes de finalizar.

Bucle Principal

Como ya se ha mencionado, la condición de salida del bucle principal es que se detecte el fichero Apocalipsis en la misma ruta en la que se encuentran Demonio.sh y Fausto.sh. Mientras no se detecte este fichero, realiza las siguientes acciones:

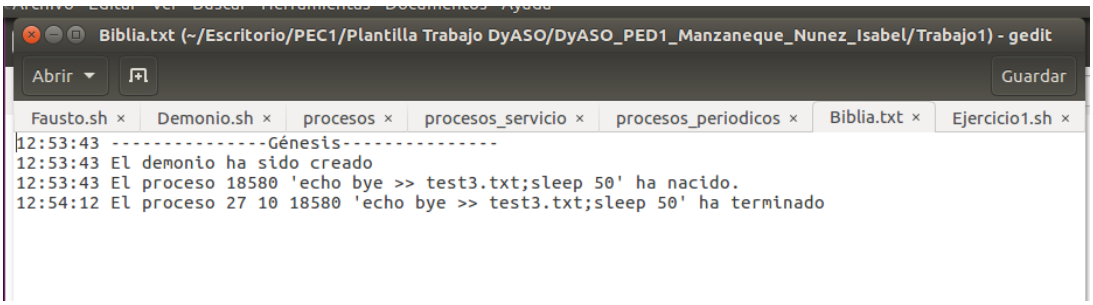
Itera por las 3 listas de procesos: procesos, procesos_servicio y procesos_periodicos y va a igualar una variable n a 1 si está iterando por las primeras dos listas y a 3 si está iterando por los procesos periódicos. Esta variable simboliza la posición del PID en cada entrada de las listas.

```
#!/bin/bash

archivos=("procesos" "procesos_servicio" "procesos_periodicos")

# BUCLE PRINCIPAL
while [ ! -f "Apocalipsis" ]
do


    # Itera por cada lista
    for archivo in "${archivos[@]}"
    do
        n=1
        if [ "$archivo" == "procesos_periodicos" ]
        then
            n=3
        fi
```

Por cada entrada de la lista, e independientemente de la lista en la que se encuentre, lo primero que hace que hace es comprobar si existe un fichero en la carpeta Infierno con nombre el PID de la entrada actual. Esto es independiente de la lista porque siempre que un fichero se encuentre en la carpeta infierno se realizan las mismas acciones:

- Terminará todo el árbol del proceso asociado a ese PID, es decir, el proceso y todos sus posibles descendientes. Esto se hará siguiendo la recomendación del enunciado, utilizando pstree con la opción -p y pasando como argumento el PID del proceso a eliminar para mostrar la jerarquía de procesos relacionada con el PID en cuestión. Después, se utiliza grep para extraer los PIDs de los procesos que componen esa jerarquía y finalizarlos uno a uno.
- Elimina la entrada correspondiente de la lista y el fichero de la carpeta Infierno. Además, añade una entrada en la Biblia indicando que el proceso se ha terminado. Las actualizaciones de la lista y la Biblia se realizan todas bloqueando el fichero SanPedro previamente para evitar condiciones de carrera y otras inconsistencias.

```
        # Itera por cada entrada de la lista
        cat "$archivo" | while read line
        do
            pid=$(echo $line | awk -v N=$n '{print $N}')

            # el proceso esta en el infierno, todos realizan la misma accion
            if [ -e "./Infierno/$pid" ]
            then

                # terminar arbol del proceso
                arbolProcesos=$(pstree -p "$pid" | grep -o '[0-9]\+')
                for proceso in $arbolProcesos
                do
                    kill -15 "$proceso"
                done
                # eliminar entrada de la lista y fichero del infierno
                flock SanPedro -c "sed -i \"\~$line~d\" \"$archivo\""
                rm -f "./Infierno/$pid"
                flock SanPedro -c "echo \"$(date +%H:%M:%S) El proceso $line ha terminado\" >> ./Biblia.txt"
```



Si no encuentra el PID en la carpeta Infierno, el Demonio realizará diferentes acciones en función de la lista en la que se encuentre:

- **procesos**: El Demonio comprobará si el proceso se encuentra actualmente en ejecución. Si lo está, no hará nada. Si el proceso no está en ejecución, lo eliminará de la lista ya que esto significa que el proceso ha terminado de ejecutarse. También incluirá una entrada en la biblia con formato: *HH:MM:SS El proceso PID 'comando' ha terminado.* Ambas actualizaciones se realizan tras bloquear el fichero SanPedro.

```
# Procesos: si no se esta ejecutando lo elimina de la lista
if ! kill -0 "$pid" >/dev/null && [ "$archivo" == "procesos" ]
then
    flock SanPedro -c "sed -i \"\~$line~d\" \"$archivo\""
    flock SanPedro -c "echo \"$(date +%H:%M:%S) El proceso $line ha terminado\" >> ./Biblia.txt"
fi
```

- **procesos_servicio**: Este tipo de procesos debe estar continuamente en ejecución. Es por ello que si el Demonio comprueba que no se encuentra actualmente en ejecución (y ya hemos establecido que tampoco se encuentra en el infierno), lo resucitará. Para resucitar el proceso extraerá el comando, lo volverá a lanzar, extraerá el PID del nuevo proceso creado y lo utilizará para actualizar la entrada correspondiente en la lista de procesos servicio. Además, creará una nueva entrada en la biblia indicando que el proceso ha resucitado con un PID nuevo: *HH:MM:SS El proceso PID 'comando' resucita con el pid PIDNUEVO.* Ambas actualizaciones se realizan tras conseguir el bloqueo a SanPedro.

```
# Procesos Servicio: si no se esta ejecutando lo resucita
if ! kill -0 "$pid" >/dev/null && [ "$archivo" == "procesos_servicio" ]
then

    # ejecuta el comando
    comandoProceso=$(echo "$line" | grep -o "'.*'" | sed "s/'//g")
    bash -c "$comandoProceso" &
    pidNuevo="$!"

    # sustituye el pid
    flock SanPedro -c "sed -i -e \"s/$pid/$pidNuevo/g\" \"$archivo\""
    flock SanPedro -c "echo \"$(date +%H:%M:%S) El proceso $line resucita con el pid $pidNuevo\" >> ./Biblia.txt"
fi
```

- **procesos_periodicos**: Este tipo de proceso debe volver a ejecutarse pasado un periodo de tiempo T especificado por el usuario. Tras comprobar que el proceso no se encuentra en el infierno, lo primero que se hará será incrementar su contador asociado, que es el primero elemento de la entrada. El Demonio comprueba entonces si el proceso no se encuentra en ejecución y si además el contador es mayor o igual al periodo T (el segundo elemento de la entrada). Si no se cumple alguna de estas condiciones, se actualiza el contador y no se hace

nada más. En cambio, si se cumplen ambas condiciones, se vuelve a lanzar el proceso y se obtiene su nuevo PID. Se actualiza el PID y el contador se pone a 0 de nuevo. Además, se crea una nueva entrada en la Biblia indicando que el proceso se ha reencarnado: *HH:MM:SS El proceso PID 'comando' se ha reencarnado en el pid PIDNUEVO*

```
                # Procesos periodicos
                if [ "$archivo" == "procesos_periodicos" ]
                then
                    # incrementar contador
                    vectorLine=($line)
                    ((vectorLine[0]++))
                    # si el proceso no se esta ejecutando y el contador es mayor o igual al periodo
                    if ! kill -0 "$pid" >/dev/null && [ "${vectorLine[0]}" -ge "${vectorLine[1]}" ]
                    then

                        # volver a lanzar el proceso
                        comandoProceso=$(echo "$line" | grep -o "'.*'" | sed "s/'//g")
                        bash -c "$comandoProceso" &
                        echo "comandoproceso: $comandoProceso"

                        #poner contador a 0 y sustituye el pid
                        vectorLine[0]=0
                        newLine="${vectorLine[*]}"
                        pidNuevo="$!"

                        flock SanPedro -c "{
                            sed -i \"s~$line~$newLine~g\" \"$archivo\"
                            sed -i \"s~$pid~$pidNuevo~g\" \"$archivo\"
                        }"

                        flock SanPedro -c "echo \"$(date +%H:%M:%S) El proceso $pid '$comandoProceso' se ha reencarnado en el pid $pidNuevo\" >> ./Biblia.txt"

                    else
                        newLine="${vectorLine[*]}"
                        flock SanPedro -c "sed -i \"s~$line~$newLine~g\" \"$archivo\""
                    fi
                fi
            fi
        done
    done

    sleep 1

done
```

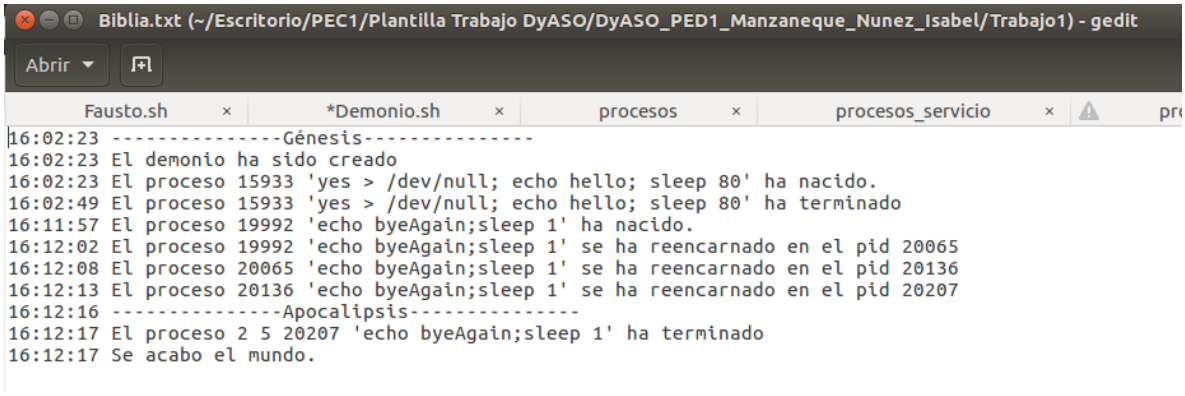
Por último, el Demonio duerme durante un segundo antes de comenzar la siguiente vuelta.

Apocalipsis

Una vez que Fausto detecta la existencia del fichero Apocalipsis, comenzará a realizar las acciones necesarias para finalizar dejando el sistema en un estado consistente. Comenzará creando una entrada Apocalipsis en la Biblia, que se realiza obteniendo el bloqueo sobre SanPedro primero.

A continuación, de manera similar al bucle principal, va a iterar por las diferentes listas y obteniendo el primer o tercer elemento de la entrada (el PID) en función de la lista en la que se encuentre. Por cada entrada, va a comprobar si el proceso se encuentra aún en ejecución y si es así, termina su árbol de procesos. Además, por cada proceso que haya terminado creará una entrada en la Biblia indicando que el proceso ha terminado, como podemos ver a continuación:

```
# APOCALIPSIS
flock SanPedro -c "echo \"$(date +%H:%M:%S) ---------------Apocalipsis---------------\" >> ./Biblia.txt"

# terminar todos los procesos de todas las listas
for archivo in "${archivos[@]}"
do
    n=1
    if [ "$archivo" == "procesos_periodicos" ]
    then
        n=3
    fi

    cat "$archivo" | while read line
    do
       pid=$(echo $line | awk -v N=$n '{print $N}')

       # si el proceso esta en ejecucion, lo termina
       if kill -0 "$pid" >/dev/null
       then
           arbolProcesos=$(pstree -p "$pid" | grep -o '[0-9]\+')
           for proceso in $arbolProcesos
           do
               kill -15 "$proceso"
           done
       fi

       flock SanPedro -c "echo \"$(date +%H:%M:%S) El proceso $line ha terminado\" >> ./Biblia.txt"
    done
done
```



Tras terminar los procesos que quedaban en ejecución, se borrarán todas las carpetas y ficheros a excepción de Fausto.sh, Demonio.sh y Biblia.txt, donde se realizará una última actualización indicando que se ha acabado el mundo.

```
# borra todo menos Fausto, Demonio y Biblia
rm -f procesos procesos_servicio procesos_periodicos Apocalipsis SanPedro
rm -fr Infierno
echo "$(date +%H:%M:%S) Se acabo el mundo." >> ./Biblia.txt
```

Ejecución de ejemplo

En este apartado, se van a repasar y comentar cada uno de los puntos del Ejercicio1.sh para comprobar que la ejecución es coherente y correcta:

Test 1)

```
**********************************************************
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───lightdm───lightdm───upstart───Demonio.sh───sleep
```

Vemos como Demonio.sh no es hijo de Bash, por lo que este apartado se ejecuta correctamente

Test 2)

```
**********************************************************
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 *****
21636 'sleep 10; echo hola >> test1.txt'
***** Procesos servicio *****
21652 'yes > /dev/null'
***** Procesos periodicos *****
0 5 21661 'echo hola_periodico >> test2.txt'
0 5 21669 '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 20319  6358  0  80   0 -  2059 wait   pts/11   00:00:00 bash
0 S  1000 21612 20319  0  80   0 -  1675 wait   pts/11   00:00:00 Ejercicio1.sh
0 S  1000 21624  1376  0  80   0 -  1681 wait   pts/11   00:00:00 Demonio.sh
0 S  1000 21635 21624  0  80   0 -  1375 hrtime pts/11   00:00:00 sleep
0 S  1000 21636  1376  0  80   0 -  1671 wait   pts/11   00:00:00 bash
0 S  1000 21639 21636  0  80   0 -  1375 hrtime pts/11   00:00:00 sleep
0 S  1000 21652  1376  0  80   0 -  1671 wait   pts/11   00:00:00 bash
0 R  1000 21655 21652  0  80   0 -  1374 -      pts/11   00:00:00 yes
0 S  1000 21669  1376  0  80   0 -  1671 wait   pts/11   00:00:00 bash
0 S  1000 21672 21669  0  80   0 -  1375 hrtime pts/11   00:00:00 sleep
0 R  1000 21681 21612  0  80   0 -  2189 -      pts/11   00:00:00 ps
```

Al utilizar la orden list de Fausto para comprobar los procesos lanzados, vemos que todos ellos parecen haberse lanzado correctamente. Al comprobar los procesos existentes con ps -l, encontramos el proceso 21636, 21652 y 21669. El proceso 21661 se ejecuta inmediatamente y termina, por lo que no se ve reflejado. Este apartado también es correcto.

Test 3)

```
**********************************************************
3) Elimino manualmente el proceso yes sin avisar a Fausto.
El Demonio debería detectarlo y reiniciar el proceso:
**********************************************************

pkill yes

bash: línea 1: 21655 Terminado              yes > /dev/null
./Fausto.sh list
***** Procesos normales *****
21636 'sleep 10; echo hola >> test1.txt'
***** Procesos servicio *****
21698 'yes > /dev/null'
***** Procesos periodicos *****
3 5 21661 'echo hola_periodico >> test2.txt'
3 5 21669 'echo hola_periodico_lento >> test3.txt; sleep 20'
```

Se elimina el proceso 21655 (hijo de 21652) manualmente. En este caso, el proceso debe resucitarse con un PID nuevo y, como puede observarse en la imagen, este es el caso que se da.

Test 4)

```
**********************************************************
4) Elimino el proceso usando Fausto.
El Demonio NO debe reiniciar el proceso:
**********************************************************

./Fausto.sh stop 21698
./Fausto.sh list
***** Procesos normales *****
21636 'sleep 10; echo hola >> test1.txt'
***** Procesos servicio *****
***** Procesos periodicos *****
1 5 21831 'echo hola_periodico >> test2.txt'
6 5 21669 '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 20319  6358  0  80   0 -  2059 wait   pts/11   00:00:00 bash
0 S  1000 21612 20319  0  80   0 -  1675 wait   pts/11   00:00:00 Ejercicio1.sh
0 S  1000 21624  1376  0  80   0 -  1681 wait   pts/11   00:00:00 Demonio.sh
0 S  1000 21636  1376  0  80   0 -  1671 wait   pts/11   00:00:00 bash
0 S  1000 21639 21636  0  80   0 -  1375 hrtime pts/11   00:00:00 sleep
0 S  1000 21669  1376  0  80   0 -  1671 wait   pts/11   00:00:00 bash
0 S  1000 21672 21669  0  80   0 -  1375 hrtime pts/11   00:00:00 sleep
0 S  1000 21864 21624  0  80   0 -  1375 hrtime pts/11   00:00:00 sleep
0 R  1000 21870 21612  0  80   0 -  2189 -      pts/11   00:00:00 ps
```

Ahora vamos a eliminar el proceso utilizando la orden de Fausto. Como estamos enviando el proceso al infierno, no queremos que resucite, sino que debe eliminarse toda su jerarquía. Tras pararlo, comprobamos que el proceso no aparece utilizando la orden list. Además, comprobamos los procesos existentes utilizando ps -l y podemos comprobar que 21698 no aparece en la lista.

Test 5)

```
**********************************************************
5) Error de sintaxis provocado para que Fausto nos avise:
**********************************************************

./Fausto.sh asdf
Error! No existe la orden 'asdf'. Consulte las órdenes disponibles con ./Fausto.sh help

**********************************************************
```

Tal y como esperábamos, al introducir una orden que no existe se hace saber al usuario que ha habido un error y se le sugiere que utilice el comando help para ver las órdenes disponibles.

Test 6)

```
************************************************************
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

************************************************************
```

Los comandos disponibles se muestran también correctamente al utilizar la orden help.

Test 7)

```
************************************************************
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

************************************************************
```

Los mensajes enviados por los procesos lanzados se corresponden con los que vemos en la salida esperada del enunciado.

Test 8)

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

  PID TTY          TIME CMD
20319 pts/11   00:00:00 bash
21612 pts/11   00:00:00 Ejercicio1.sh
21975 pts/11   00:00:00 ps
```

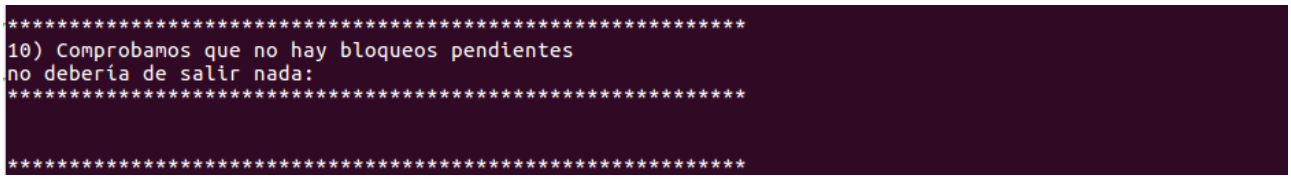
Como se han finalizado todos los procesos y sus jerarquías, no queda ningún proceso sin terminar

Test 9)

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

Biblia.txt  Demonio.sh  Fausto.sh

************************************************************
```

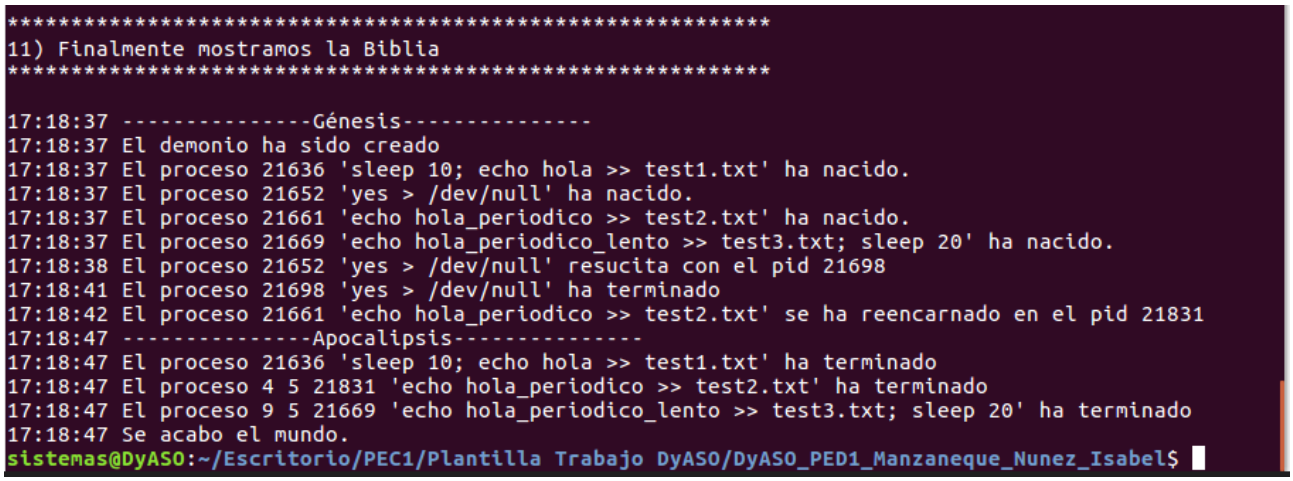
Al final de la ejecución se eliminan todos los ficheros y carpetas residuales, quedando únicamente la Biblia.txt, Demonio.sh y Fausto.sh

Test 10)



No queda ningún bloqueo pendiente, por lo tanto en este test no hay ninguna salida como resultado

Test 11)



Algunas observaciones sobre el resultado en la biblia:

- El proceso 21636 termina 10 segundos después de comenzar (Al coincidir con el apocalipsis, este se ejecuta primero). Se han dado iteraciones en las que la diferencia de tiempo entre Génesis y Apocalipsis era de 11 segundos en lugar de 10 y 21636 ha finalizado justo antes de apocalipsis, por lo que parece una cuestión de diferencias pequeñas entre los tiempos de ejecución.
- El proceso 21661 se reencarna a los 5 segundos
- El proceso 21652 resucita una vez en 21698 y no vuelve a resucitarse, ya que lo termina Fausto.

Es por todo esto que creo que se mantiene la coherencia durante la ejecución de la prueba y que creo que el gestor de procesos opera de manera correcta.

Conclusiones

Si bien ha sido en ocasiones algo frustrante, he aprendido mucho con esta práctica y me alegro de haberla hecho. Me ha costado un poco desenvolverme con Bash, pero creo que es muy importante poner esfuerzo en aprender porque es el intérprete de comandos por defecto en la mayoría de los sistemas operativos basados en Unix, lo que le hace una herramienta esencial para desarrolladores que trabajan en este tipo de entornos.

Me ha parecido muy interesante ver el ciclo de vida de los procesos y como interactúan entre ellos. Me ha parecido especialmente interesante aprender más sobre los procesos en segundo plano, los cuales para mi eran un concepto algo abstracto pero que ahora son una idea algo más tangible.

Como en toda buena práctica de programación, me he quedado atascada muchas veces y he pasado muchas horas delante del ordenador intentando solucionar bugs, pero según he ido adquiriendo más soltura me ha ido ocurriendo menos.

Mi dificultad más grande ha sido debida a la sincronización entre procesos con Flock y también peculiaridades de los comandos relativas a sus opciones y parámetros. Por ejemplo, el comando sed me ha dado muchos problemas por utilizarlo de forma incorrecta o no tener en cuenta detalles como los caracteres especiales al pasarle una variable.

Esta práctica me ha dejado con un buen sabor de boca y ganas de aprender más sobre este intérprete de comandos.

Bibliografía.

Fundamentos del sistema operativo UNIX, José Manuel Díaz - Rocío Muñoz Mansilla - Dictino Chaos García

https://www.baeldung.com/linux/use-command-line-arguments-in-bash-script

https://www.explainshell.com

https://stackoverflow.com

https://superuser.com/

https://www.gnu.org/savannah-checkouts/gnu/bash/manual/bash.html

https://devhints.io/bash

https://ryanstutorials.net/bash-scripting-tutorial/bash-loops.php

https://linuxcommand.org/lc3_wss0080.php

https://linuxconfig.org/advanced-bash-regex-with-examples

https://utcc.utoronto.ca/~cks/space/blog/linux/FlockUsageNotes