

Práctica 2. Sobre demonios y tuberías.

In []:

```
%%bash
rm -rf *.h *.c *.o *.elf Makefile
/*
DANIEL LEDESMA VENTURA
BADR GUAITOUNE AKDI
*/
```

Funcionamiento general

En esta práctica crearéis un servicio de tiempo. Dicho demonio escuchará peticiones por tuberías con nombre y responderá a la solicitud por otra tubería con nombre.

Demonio de notificación de tiempo (hasta 4 puntos)

- Se creará un proceso demonio que programará una alarma (señal `SIGALARM`) con un determinado período.
- Existirá un fichero de configuración (`.tiempo.conf`) que incluirá una línea en la que se indicará cada cuántos segundos se recibirá la señal tipo `SIGALARM`. El fichero de configuración deberá situarse en el directorio `HOME` del usuario que ejecute el demonio. **Importante:** utilizad `getenv()` para construir dicha ruta en vuestro código.
- Se creará un fichero testigo (`.tiempo.lock`) en el directorio `HOME` para indicar que el demonio está en ejecución. Si se ejecuta el demonio y ya existe el fichero testigo, se terminará la ejecución notificando el error por la salida de error estándar. Dicho fichero contendrá el `PID` del proceso que ejecutará el demonio.
- Cuando llegue la señal de alarma, se realizará un log usando `syslog()` incluyendo la hora actual. Puedes consultar documentación y ejemplos en este [enlace](#).
- En caso de recibir la señal `SIGHUP`, se leerá nuevamente el fichero de configuración (`.tiempo.conf`) por si ha habido actualizaciones.
- En caso de recibir la señal `SIGUSR1` se forzarán un log en ese mismo momento, aunque no haya llegado la señal de alarma.
- En caso de recibir la señal `SIGTERM` se finalizará el proceso de manera ordenada: cerrando ficheros abiertos, liberando memoria si es preciso y borrando el fichero testigo.
- Se bloquearán todas las señales no pertinentes.

Creación servidor (hasta 6 puntos)

El comportamiento del servidor (demonio) deberá seguir estas consideraciones:

- Inicialmente, el demonio escuchará si llega algún nuevo cliente mediante la tubería con nombre `/tmp/newclient`.
- Cuando un nuevo cliente se quiera conectar al servidor, el cliente deberá enviar por esa tubería una cadena de caracteres con su `PID`. Asimismo, el cliente creará dos tuberías con nombre a partir de su `PID`. Así, si el `PID` del proceso cliente es `1513`, creará las tuberías `/tmp/toServer1513` y `/tmp/toClient1513`.
- El servidor pasará a escuchar tanto a la tubería `/tmp/newclient` como a todas las tuberías `toServer` que se hayan creado. Para ello usará `select()`, `poll()` o `epoll()`.
- Todo cliente puede escribir en su tubería `toServer` para solicitar la hora actual al servidor. Para ello enviará la cadena `times`. Cuando el servidor lea dicha cadena de cualquiera de las tuberías `toServer` en las que escucha, escribirá en la tubería `toClient` correspondiente un número interno con los segundos transcurridos desde el 1/1/1970.
- Asimismo, el demonio escribirá la hora actual (expresada con un número entero) en todas las tuberías `toClient` que tenga abiertas cada vez que reciba la señal `SIGALARM`.
- El servidor estará preparado para recibir la señal `SIGPIPE` y cerrará la conexión con el cliente cuya tubería generó dicha señal.

Creación clientes (hasta 2 puntos).

Se creará una aplicación que se comporte como un cliente del servidor del apartado anterior. Al arrancar, se conectará al servidor (enviando su `PID` por la tubería correspondiente), creará las 2 tuberías indicadas en el apartado anterior y entrará en un bucle de `n` iteraciones (siendo `n` un número aleatorio entre 1 y 10). En las iteraciones pares solicitará la hora al servidor y luego se quedará leyendo de la tubería `toClient`; en las impares, sólo leerá de la tubería `toClient`.

In []:

```
%%writefile common.h
/*****
NADA
*****/
```

In []:

```
%%writefile common.c
/*****
NADA
*****/
```

In []:

```
%%writefile servidor.h
/*****

//-----MAX TUBERIAS-----//
#define MAX_TUB 1000

//tipos para de tuberias
typedef struct {
    int pipe[MAX_TUB];
    int index;
}tPipeLines;

//----MENSAJE QUE SE RECIBE DESDE EL CLIENTE----//
const char MSG_times[20] = "times";

void blockSigns();//Bloqueamos la señales que no vamos a usar
void initSigns();//Iniciamos para cada señal su correspondiente handler
void * iniHandler(int signal, void *handler);//Enlazamos la señal con su handler

//----HANDLERS PARA CADA SEÑAL----//
void handler_SIGTERM();
void handler_SIGUSR1();
void handler_SIGHUP();
void handler_SIGALARM();
void handler_SIGPIPE();

void openlogG(char *text);//Genera los log del programa
void config();//Configuracion de la señal alarm mediante su archivo
*****/
```

In []:

```
%%writefile servidor.c
/*****
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <errno.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include <sys/time.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <time.h>
*****/
```

```

#include <stdlib.h>

#include "servidor.h"

extern int errno;

//-----TUBERIAS-----//
tPipeLines readers;
tPipeLines writers;

//-----DESCRIPTOR DEL FICHERO-----//
int fdLock;

//-----ESTRUCTURAS NECESARIAS PARA LA SEÑAL SIGALARM---//
struct itimerval it;

//-----CONJUNTO DE DESCRIPTORES PARA SELECT()-----//
fd_set setReadFiles;

//-----CONJUNTO DE SEÑALES QUE BLOQUEAREMOS-----//
sigset_t signs;

//-----VARIABLES QUE INDICA QUE UN DESCRIPTOR SE HA CERRADO Y CUAL-----//
int SIG_EPIPE = 0;
int indexPP;

void openlogG(char *text){
    setlogmask (LOG_UPTO (LOG_NOTICE));
    openlog("DEMONIO", LOG_CONS | LOG_PID | LOG_NDELAY, LOG_DAEMON);
    syslog(LOG_NOTICE, "%s",text);
    closelog();
}

void config(){
    int fd;
    char buffer[20];
    if((fd = open("/home/tiempo.conf", O_RDWR)) == -1){
        openlogG("NO EXISTE -> tiempo.conf -> /home/\n");
        for (int i = 0; i < readers.index; i++) {if (readers.pipe[i] != -1) close(readers.pipe[i]);}
        for (int i = 0; i < writers.index; i++) {if (writers.pipe[i] != -1) close(writers.pipe[i]);}
        remove("/tmp/newclient");
        remove("tiempo.lock");
        exit(EXIT_FAILURE);
    }
    printf("config()\n");
    read(fd, buffer, sizeof(int));
    close(fd);
    it.it_value.tv_sec = atoi(buffer);
    it.it_value.tv_usec = 0;
    it.it_interval.tv_sec = atoi(buffer);
    it.it_interval.tv_usec = 0;
    setitimer(ITIMER_REAL, &it, NULL);
}

void handler_SIGALARM() {
    char hour[30];
    time_t t = time(NULL);
    struct tm *tm = localtime(&t);
    strftime(hour, 100, "%H : %M : %S", tm);
    for (int i = 1; i < writers.index; i++) {
        if (writers.pipe[i] != -1) {
            if (((write(writers.pipe[i],hour, sizeof(hour))) == -1) && (errno == EPIPE)) {
                SIG_EPIPE = 1;
                indexPP = i;
            }
        }
    }
}

void handler_SIGHUP() {
    openlogG("LEYENDO -> tiempo.conf -> handler_SIGHUP()\n");
    config();
}

```

```

void handler_SIGUSR1() {
    char hour[30], cadena[1024];
    time_t t = time(NULL);
    struct tm *tm = localtime(&t);
    strftime(hour, 100, "%H : %M : %S", tm);
    snprintf(cadena, sizeof(cadena), "SEÑAL handler_SIGUSR1(), HORA ACTUAL = %s\n", hour);
    openlogG(cadena);
}

void handler_SIGTERM() {
    close(fdLock);
    if (remove("tiempo.lock") == -1) {openlogG("NO SE PUEDE ELIMINAR -> tiempo.lock\n");}
    if (remove("/tmp/newclient") == -1) {openlogG("NO SE PUEDE ELIMINAR -> newclient\n"); }
    for (int i = 0; i < readers.index; i++) {if (readers.pipe[i] != -1) close(readers.pipe[i]);}
    for (int i = 0; i < writers.index; i++) {if (writers.pipe[i] != -1) close(writers.pipe[i]);}
    openlogG("EJECUCION FINALIZADA CORRECTAMENTE\n");
    exit(EXIT_SUCCESS);
}

void handler_SIGPIPE() {
    if(SIG_EPIPE == 1){
        FD_CLR((int)readers.pipe[indexPP], &setReadFiles);
        close(writers.pipe[indexPP]); close(readers.pipe[indexPP]);
        writers.pipe[indexPP] = -1; readers.pipe[indexPP] = -1;
        SIG_EPIPE = 0;
        openlogG("TUBERIA CERRADA -> handler_SIGPIPE()\n");
    }
}

void * iniHandler(int signal, void *handler){
    struct sigaction act , old_act;
    act.sa_handler = handler;
    sigemptyset(&act.sa_mask);
    act.sa_flags = SA_RESTART;
    sigaction(signal, &act, &old_act);
}

void initSigns() {
    iniHandler(SIGALRM, handler_SIGALRM);
    iniHandler(SIGHUP, handler_SIGHUP);
    iniHandler(SIGUSR1, handler_SIGUSR1);
    iniHandler(SIGTERM, handler_SIGTERM);
    iniHandler(SIGPIPE, handler_SIGPIPE);
}

void blockSigns() {
    sigfillset(&signs);
    sigdelset(&signs, SIGALRM);sigdelset(&signs, SIGHUP);sigdelset(&signs, SIGUSR1);
    sigdelset(&signs, SIGTERM);sigdelset(&signs, SIGPIPE);
    sigprocmask(SIG_BLOCK, &signs, NULL);
}

void demonio(){
    int pid , cambios, max, fd_toClient, fd_toServer, cmp;
    struct timeval timeout;
    char pathClient[20],pathServer[20],cadena[20];
    FD_ZERO(&setReadFiles);
    readers.pipe[0] = open("/tmp/newclient", O_RDONLY);
    FD_SET(readers.pipe[0], &setReadFiles);
    max = readers.pipe[0];
    writers.pipe[0] = -1;
    readers.index = 1;
    writers.index = 1;
    while (1) {
        FD_ZERO(&setReadFiles);
        for (int i = 0; i <= readers.index ; i++) {
            if (readers.pipe[i] > -1) {
                FD_SET(readers.pipe[i], &setReadFiles);
                max = max < readers.pipe[i]? readers.pipe[i] : max;
            }
        }
        cambios = select(max + 1, &setReadFiles, NULL, NULL, &timeout);
        if (cambios == 0) {
            openlogG("SELECT -> timeout\n");
        }
        else{
            if (FD_ISSET(readers.pipe[0], &setReadFiles)) {

```

```

        if(read(readers.pipe[0], &pid, 20) > 0){
            snprintf(pathServer, sizeof(pathServer), "/tmp/toServer%d", pid);
            snprintf(pathClient, sizeof(pathClient), "/tmp/toClient%d", pid);
            fd_toClient= open(pathClient, O_WRONLY);
            fd_toServer = open(pathServer, O_RDONLY);
            if (fd_toServer != -1 && fd_toClient != -1) {
                writers.pipe[writers.index] = fd_toClient;
                writers.index++;
                readers.pipe[readers.index] = fd_toServer;
                readers.index++;
            }
        }
    }
    else{
        for (int i = 1; i < readers.index ; i++) {
            if (FD_ISSET(readers.pipe[i], &setReadFiles)) {
                if(read(readers.pipe[i], cadena, sizeof(cadena)) == -1){perror("ERROR
LECTURA");exit(EXIT_FAILURE);}
                cmp = strcmp(cadena, MSG_times);
                if(cmp == 0){
                    time_t tim = time(NULL);
                    sprintf(cadena, "%d", (int)tim);
                    write(writers.pipe[i], cadena, sizeof(cadena));
                }
            }
        }
    }
}

int main() {
    pid_t pid;
    char pid_char[20];
    mkfifo("/tmp/newclient", 0666);

    pid = fork();/* Forkeamos el proceso padre */
    if (pid < 0) {perror("FALLO EN FORK");exit(EXIT_FAILURE);}
    if (pid > 0) {exit(EXIT_SUCCESS);}/* Cuando tenemos un PID correcto podemos cerrar el proceso pa
dre.*/

    umask(0);/* Cambiamos el modo de la mascara de ficheros para que los fichero generados por el d
emonio sean accesibles por todo el mundo */
    chdir("/home/");/* Por seguridad, cambiamos el directorio de trabajo */
    if (setsid() < 0) {perror("ERROR en NEWSID"); exit(EXIT_FAILURE);}/* Creamos un nuevo SID para q
ue el sistema se haga cargo del proceso huérfano*/
    close(STDIN_FILENO); close(STDOUT_FILENO); close(STDERR_FILENO);/* Cerramos los descriptores sta
ndard, son posible riesgo de seguridad.*/

    if ((fdLock = open("tiempo.lock", O_RDWR | O_CREAT, 0640)) < 0) {perror("ERROR AL ABRIR -> tiemp
o.lock");exit(EXIT_FAILURE);}/**/
    if (lockf(fdLock, F_TLOCK, 0) < 0) {perror("ERROR AL BLOQUEAR ->
tiempo.lock");exit(EXIT_FAILURE);}/*Bloqueando el fichero garantiza unica instancia*/
    sprintf(pid_char, "%d\n", getpid());
    write(fdLock, pid_char, sizeof(pid_char));/*Escribiendo el PID en archivo*/

    config();/*Leemos el archivo de configuracion para ver el periodo*/
    blockSigns();/*Bloquemos las señales que no vamos a usar*/
    initSigns();/*Inicializamos las señales en sus respectivos handlers o manejadores*/

    demonio();
}
/*****

```

In []:

```

%%writefile cliente.h
/*****
NADA
*****/

```

In []:

```

%%writefile cliente.c

```

```

/*****
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <errno.h>
#include <signal.h>
#include <stdlib.h>
#include <string.h>
#include <syslog.h>
#include <sys/time.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <time.h>

const char MSG_times[20] = "times";

int main() {
    fflush(NULL);
    char pathClient[20], pathServer[20], times[64];
    srand(time(NULL));
    pid_t pid = getpid();
    int fd_FIFO, fd_toClient, fd_toServer, i = 0, iter = rand() % (10 - 1 + 1) + 1;//numero aleatori
o entre 1 y 10
    chdir("/home/");

    if ((fd_FIFO = open("/tmp/newclient", O_WRONLY)) == -1) {perror("ERROR AL ABRIR newclient"); exit
t(EXIT_FAILURE);}

    snprintf(pathClient, sizeof(pathClient), "/tmp/toClient%d", pid);
    snprintf(pathServer, sizeof(pathServer), "/tmp/toServer%d", pid);
    if ((mkfifo(pathClient, 0666)) == -1) {perror("ERROR MKFIFO pathclient");exit(EXIT_FAILURE);}
    if ((mkfifo(pathServer, 0666)) == -1) {perror("ERROR MKFIFO pathServer");exit(EXIT_FAILURE);}

    if (write(fd_FIFO, &pid, sizeof(pid)) < 0) {perror("ERROR AL ENVIAR EL PID DEL
PROCESO");exit(EXIT_FAILURE);}

    if ((fd_toClient = open(pathClient, O_RDONLY)) == -1) {perror("ERROR AL ABRIR
toclient");remove(pathClient);remove(pathServer);exit(EXIT_FAILURE);}
    if ((fd_toServer = open(pathServer, O_WRONLY)) == -1) {perror("ERROR AL ABRIR
toServer");remove(pathClient);remove(pathServer);exit(EXIT_FAILURE);}

    printf("COMIENZO DE LA COMUNICACION\n");
    while (i < iter) {
        if (i % 2 == 0) {
            if(write(fd_toServer, MSG_times, sizeof(MSG_times)) < 0){perror("ERROR AL ESCRIBIR ->
fd_toServer");exit(EXIT_FAILURE);}
            printf("PID : %d | ENVIADO -> %s : ",pid,MSG_times);
            if(read(fd_toClient, times, sizeof(times)) < 0){perror("ERROR AL LEER ->
fd_toClient");remove(pathClient);remove(pathServer);exit(EXIT_FAILURE);}
            printf("RECIBIDO -> segundos = %s\n", times);
        }
        else {
            printf("PID : %d | NO SE ENVIA NADA : ",pid);
            if(read(fd_toClient, times, sizeof(times)) < 0){perror("ERROR AL LEER ->
fd_toClient");remove(pathClient);remove(pathServer);exit(EXIT_FAILURE);}
            printf("RECIBIDO -> hora = %s\n", times);
        }
        i++;
    }
    close(fd_toClient);
    close(fd_toServer);
    remove(pathClient);
    remove(pathServer);
    close(fd_FIFO);
    return 0;
}
*****/

```

In []:

```

%%writefile Makefile
#####
CODECLIENT = cliente.c
CODESERVER = servidor.c

```

```

CONFIGFILE = /home/.tiempo.conf
FLAGS = -Wall -g

all : cliente.elf servidor.elf config

config :
@echo "Generando fichero de configuración..."
@echo 10 >> $(CONFIGFILE)

uninstall : clean
@echo "Desinstalando demonio..."
- rm $(CONFIGFILE)
@echo "Desinstalación completa"

cliente.elf : $(CODECLIENT)
@echo "Compilando..."
gcc $^ $(FLAGS) -o $@

servidor.elf : $(CODESERVER)
@echo "Compilando..."
gcc $^ $(FLAGS) -o $@

clean :
@echo "Borrando archivos antiguos..."
-rm *.elf
-rm $(CONFIGFILE)

.PHONY: all clean install uninstall
#####

```

Este ejercicio hay que ejecutarlo en terminal, pon aquí tu salida, reemplazando la que hay como ejemplo:

```

badr@Badr:~/Escritorio/copiaASO/Practica2$ sudo ./servidor.elf
badr@Badr:~/Escritorio/copiaASO/Practica2$ sudo ./cliente.elf & sudo ./cliente.elf & sudo ./
cliente.elf &

COMIENZO DE LA COMUNICACION
COMIENZO DE LA COMUNICACION
COMIENZO DE LA COMUNICACION

PID : 5058 | ENVIADO -> times : RECIBIDO -> segundos = 1591814416
PID : 5059 | ENVIADO -> times : RECIBIDO -> segundos = 1591814416
PID : 5060 | ENVIADO -> times : RECIBIDO -> segundos = 1591814416

PID : 5058 | NO SE ENVIA NADA : RECIBIDO -> hora = 20 : 40 : 16
PID : 5060 | NO SE ENVIA NADA : RECIBIDO -> hora = 20 : 40 : 16
PID : 5059 | NO SE ENVIA NADA : RECIBIDO -> hora = 20 : 40 : 16

PID : 5058 | ENVIADO -> times : RECIBIDO -> segundos = 1591814416
PID : 5060 | ENVIADO -> times : RECIBIDO -> segundos = 1591814416
PID : 5059 | ENVIADO -> times : RECIBIDO -> segundos = 1591814416

PID : 5058 | NO SE ENVIA NADA : RECIBIDO -> hora = 20 : 40 : 26
PID : 5060 | NO SE ENVIA NADA : RECIBIDO -> hora = 20 : 40 : 26
PID : 5059 | NO SE ENVIA NADA : RECIBIDO -> hora = 20 : 40 : 26

PID : 5058 | ENVIADO -> times : RECIBIDO -> segundos = 1591814426
PID : 5060 | ENVIADO -> times : RECIBIDO -> segundos = 1591814426
PID : 5059 | ENVIADO -> times : RECIBIDO -> segundos = 1591814426


PID : 5058 | NO SE ENVIA NADA : RECIBIDO -> hora = 20 : 40 : 36
PID : 5059 | NO SE ENVIA NADA : RECIBIDO -> hora = 20 : 40 : 36
PID : 5060 | NO SE ENVIA NADA : RECIBIDO -> hora = 20 : 40 : 36

[4] 5058
[5] 5059
[6] 5060
[1] Hecho sudo ./cliente.elf

```

```
[2] Hecho          sudo ./cliente.elf
[3] Hecho          sudo ./cliente.elf
```

```
badr@Badr:~/Escritorio/copiaASO/Practica2$
```

A horizontal scrollbar for the terminal window, with a small black slider on the left and a small black square on the right.