Práctica 3. Paso de mensajes.

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 idéntico al de la práctica 2, pero sustituyendo las tuberías por un mecanismo de paso de mensajes.

Demonio de notificación de tiempo

Esta parte será idéntica a la práctica anterior.

Creación servidor (hasta 6 puntos)

El comportamiento del servidor (demonio) deberá ser similar al de la práctica anterior, pero usando paso de mensajes, en lugar de tuberías y `select()`. Se usarán dos colas de mensajes: una para recibir solicitudes de los clientes y otra para enviar las respuestas (incluyendo los mensajes destinados a todos los clientes, que se enviarán cuando venza la alarma del servidor).

Creación clientes (hasta 4 puntos).

Se creará una aplicación que se comporte como un cliente del servidor del apartado anterior. Al arrancar, 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 cola de mensajes; en las impares, sólo leerá de la cola de mensajes.

In [ ]:

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

In [ ]:

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

In [ ]:

```
%%writefile servidor.h
/*****************************************************************************/
#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 <sys/msg.h>


//--------TAMAÑO MAXIMO DEL MESAJE-----//
#define MAX_MSG 100

//--------MAXIMO DE LCIENTES-----//
#define MAX_CLNT 100

//-----CONSTANTES PARA LOS MENSAJES ftok(...,...);---//
#define QUEUE_CS 'B' //BADR GUAITOUNE AKDI
#define QUEUE_SC 'D'//DANIEL LEDESMA VENTURA
#define KEYFILE "tiempo.lock"//ARCHIVO DEL SISTEMA

//-----ESTRUCTURA DEL MENSAJE QUE SE ENVIA---//
typedef struct{
    long mtype;//TIPO DE MESAJE
    pid_t pid; //PID DEL PROCESO QUE ENVIA EL MENSAJE
    char message[MAX_MSG];//MENSAJE QUE SE ENVIA
}tMsg;

//-----ESTRUCTURA PARA LLEVAR LOS CLIENTES DADOS DE ALTA----//
typedef struct{
    int index;
    pid_t client[MAX_CLNT];
}tClient;

//-----TIPOS DE CONSTANTES PARA LOS MENSAJES----//
#define LONG (sizeof(tMsg) - sizeof (long))


//-----TIPOS DE CONSTANTES PARA LOS MENSAJES----//
const long NEWCLIENT = 1;
const long CLIENT = 2;
const long DELETECLIENT = 3;


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

In [ ]:

```
%%writefile servidor.c
/*****************************************************************************/
#include "servidor.h"

extern int errno;

//-----------COLA DE MENSAJES------------//
int msg_cs, msg_sc;
key_t key;

//-------ESTRUCTURA PARA MANDAR ALARMA A LOS CLIENTES---//
tClient all_client;

//----------DESCRIPTOR DEL FICHERO---------------------//
```

```
int fdLock;

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

//------------CLAVE PARA LA COLA DE MENSJAES-------------//
key_t key;

//------------VARIABLE PARA LAS SEÑALES-------------//
sigset_t signs;


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");
   remove("tiempo.lock");
   exit(EXIT_FAILURE);
     }
  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], cadena[1024];
     tMsg msg;
     time_t t = time(NULL);
  struct tm *tm = localtime(&t);
  strftime(hour, 100, "%H : %M : %S", tm);
  for (int i = 0; i < all_client.index; i++) {
   if (all_client.client[i] != -1) {
             msg.mtype = all_client.client[i];
             msg.pid = getpid();
    strcpy(msg.message,hour);
    if(msgsnd(msg_sc,(struct msgbuf *)&msg,LONG,0) == -1){
                snprintf(cadena, sizeof(cadena), "SEÑAL handler_SIGALARM(), ERROR AL ENVIAR HORA A
CTUAL = %s AL CLIENTE : %d\n", hour,all_client.client[i]);
             openlogG(cadena);
       }
      }
     }
}
void handler_SIGHUP() {
  printf("handler_SIGHUP()\n");
  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("/home/tiempo.lock") == -1) {openlogG("NO SE PUEDE ELIMINAR -> tiempo.lock\n");}
     msgctl(msg_cs,IPC_RMID,NULL);
  msgctl(msg_sc,IPC_RMID,NULL);
  openlogG("EJECUACION FINALIZADA CORRECTAMENTE\n");
  exit(EXIT_SUCCESS);
```

```
}

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_SIGALARM);
  iniHandler(SIGHUP,  handler_SIGHUP);
  iniHandler(SIGUSR1, handler_SIGUSR1);
  iniHandler(SIGTERM, handler_SIGTERM);
}

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




void demonio(){
     pid_t pid;
  tMsg msg;
     char cadena[100];
     int cmp, solu;
     all_client.index = 0;

  key  = ftok(KEYFILE,QUEUE_CS);
     if (key == (key_t)-1){perror("ERROR AL OBTENER CLAVE PARA COLA ->
QUEUE_CS");exit(EXIT_FAILURE);}
  msg_cs = msgget(key,IPC_CREAT|0600);
     if (msg_cs == -1){perror("ERROR AL OBTENER ID PARA COLA CS");exit (EXIT_FAILURE);}

  key  = ftok(KEYFILE,QUEUE_SC);
     if (key == (key_t)-1){perror("ERROR AL OBTENER CLAVE PARA COLA ->
QUEUE_SC");exit(EXIT_FAILURE);}
  msg_sc = msgget(key,IPC_CREAT|0600);
     if (msg_sc == -1){perror("ERROR AL OBTENER ID PARA COLA SC");exit (EXIT_FAILURE);}

     while(1){
         if((solu = msgrcv(msg_cs,(struct msgbuf *)&msg,LONG,NEWCLIENT,IPC_NOWAIT)) > 0){
             all_client.client[all_client.index++] = msg.pid;
         }
         if((solu = msgrcv(msg_cs,(struct msgbuf *)&msg,LONG,CLIENT,IPC_NOWAIT)) > 0){
             cmp = strcmp(msg.message,MSG_times);
    if(cmp == 0){
                 time_t tim = time(NULL);
     sprintf(cadena, "%d", (int)tim);
           strcpy(msg.message,cadena);
                 msg.mtype = msg.pid;
                 msgsnd(msg_sc,(struct msgbuf *)&msg,LONG,0);
    }
   }
         if((solu = msgrcv(msg_cs,(struct msgbuf *)&msg,LONG,DELETECLIENT,IPC_NOWAIT)) > 0){
             int i = 0;
             while(i <= all_client.index){
     if(all_client.client[i] == msg.pid){
      all_client.client[i] = -1;
                     i = all_client.index + 1;
     }
                 i++;
    }
         }
         solu = 0;
     }
}

int main() {
  pid_t pid;
  char pid_char[20];
```

```
  char pid_char[20];

  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("/home/tiempo.lock", O_RDWR | O_CREAT, 0640)) < 0) {perror("ERROR AL ABRIR ->
tiempo.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();/*Bloquemanos 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 "servidor.h"

int main() {
  fflush(NULL);
  chdir("/home/");/* Por seguridad, cambiamos el directorio de trabajo */
     srand(time(NULL));
  tMsg msg;
  int msg_cs, msg_sc , iter = rand() % (10 - 1 + 1) + 1 ,i = 0, solu;
  key_t key;
     pid_t pid = getpid();

  key  = ftok(KEYFILE,QUEUE_CS);
     if (key == (key_t)-1){perror("ERROR AL OBTENER CLAVE PARA COLA -> QUEUE_CS
");exit(EXIT_FAILURE);}
  msg_cs = msgget(key,IPC_CREAT|0600);
     if (msg_cs == -1){perror("ERROR AL OBTENER ID PARA COLA CS");exit (EXIT_FAILURE);}

  key  = ftok(KEYFILE,QUEUE_SC);
     if (key == (key_t)-1){perror("ERROR AL OBTENER CLAVE PARA COLA -> QUEUE_SC
");exit(EXIT_FAILURE);}
  msg_sc = msgget(key,IPC_CREAT|0600);
     if (msg_sc == -1){perror("ERROR AL OBTENER ID PARA COLA SC");exit (EXIT_FAILURE);}


     strcpy (msg.message, "");
     msg.mtype = NEWCLIENT;
     msg.pid = pid;

     if(msgsnd(msg_cs,(struct msgbuf *)&msg,LONG, 0 ) < 0){perror("ERROR CONECTAR CON
SERVIDOR");exit (EXIT_FAILURE);}

     printf("COMIENZO DE LA COMUNICACION\n");
  while (i < iter) {
   if (i % 2 == 0) {
```

```
            strcpy(msg.message,MSG_times); msg.mtype = CLIENT; msg.pid = pid;
            if((solu = msgsnd(msg_cs,(struct msgbuf *)&msg,LONG,0)) < 0){perror("ERROR AL ENVIAR
MSG");exit (EXIT_FAILURE);}
          printf("PID : %d | ENVIADO -> %s : ",pid,MSG_times);
    if((solu = msgrcv(msg_sc,&msg,LONG,pid,0)) == 0){perror("ERROR AL RECIBIR MSG");exit
(EXIT_FAILURE);}
    printf("RECIBIDO -> segundos = %s\n", msg.message);
   }
  else {
            printf("PID : %d | NO SE ENVIA NADA : ",pid);
    if((solu = msgrcv(msg_sc,(struct msgbuf *)&msg,LONG,pid,0)) < 0){perror("ERROR AL RECIBIR MSG"
);exit (EXIT_FAILURE);}
    printf("RECIBIDO -> hora = %s\n", msg.message);
  }
  i++;
 }
    strcpy(msg.message,""); msg.mtype = DELETECLIENT; msg.pid = pid;
 if((solu = msgsnd(msg_cs,(struct msgbuf *)&msg,LONG,0)) < 0){perror("ERROR AL ENVIAR MSG");exit (
EXIT_FAILURE);}
 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..."
 -sudo 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 client tiempod-server

.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/Practica3$
COMIENZO DE LA COMUNICACION
PID : 6902 | ENVIADO -> times : RECIBIDO -> segundos = 1591818749
COMIENZO DE LA COMUNICACION
PID : 6903 | ENVIADO -> times : RECIBIDO -> segundos = 1591818749
COMIENZO DE LA COMUNICACION
PID : 6904 | ENVIADO -> times : RECIBIDO -> segundos = 1591818749
PID : 6902 | NO SE ENVIA NADA : RECIBIDO -> hora = 21 : 52 : 37
PID : 6903 | NO SE ENVIA NADA : RECIBIDO -> hora = 21 : 52 : 37
PID : 6904 | NO SE ENVIA NADA : RECIBIDO -> hora = 21 : 52 : 37
PID : 6902 | ENVIADO -> times : RECIBIDO -> segundos = 1591818757
PID : 6903 | ENVIADO -> times : RECIBIDO -> segundos = 1591818757
```

```
PID : 6904 | ENVIADO -> times : RECIBIDO -> segundos = 1591818757
badr@Badr:~/Escritorio/copiaASO/Practica3$
[3]   Hecho                   sudo ./cliente.elf
[4]-  Hecho                   sudo ./cliente.elf
[5]+  Hecho                   sudo ./cliente.elf
```