# Arquitectura de Sistemas

## Práctica 8: Sincronización mediante señales

---

Gustavo Romero López

Updated: 9 de mayo de 2025

Departamento de Ingeniería de Computadores, Automática y Robótica

# Objetivos

1. Aprender a utilizar señales POSIX tanto desde el intérprete de órdenes como desde un programa en C/C++.
2. Escribir un programa en el que varias hebras sincronicen su funcionamiento mediante señales.

# makefile

```
ATT = $(EXE:=.att)
EXE = $(basename $(SRC))
SRC = $(wildcard *.c *.cc)

CFLAGS = -g -march=native -O3 -pthread -Wall
CXXFLAGS = $(CFLAGS)

all: exe

att: $(ATT)

clean:
    -rm -fv $(ATT) $(EXE) core.* *~
    -find -mindepth 2 -maxdepth 2 -name makefile -execdir make
        $@ \;

exe: $(EXE)
```

# Señales UNIX en procesos

```
#include <sys/types.h> // pid_t
#include <unistd.h>    // getpid
#include <signal.h>    // kill raise signal

// instala "manejador" como manejador de señal para
// la señal "signum"
void (*signal(int signum, void (*manejador)(int)))(int);

// envia la señal "signum" a el proceso "pid"
// o a un grupo de procesos
int kill(pid_t pid, int signum);

// envía la señal "signum" a el proceso actual
// equivalente a kill(getpid(), sig)
int raise(int signum);
```

# Señales UNIX en hebras (pthread y C++)

## pthread

```
#include <pthread.h> // pthread_t
#include <signal.h>  // pthread_kill signal

// envía la señal "signum" a la hebra "hebra"
int pthread_kill(pthread_t hebra, int signum);
```

## C++

```
#include <csignal>  // pthread_kill std::signal
#include <thread>   // std::thread

std::thread t(...);
pthread_kill(t.native_handle(), SIGINT);
std::raise(SIGINT);
```

# Lista de señales UNIX (POSIX.1): `kill -l`

| Signal | Value | Action | Comment (Value = alpha/sparc,i386/ppc/sh,mips) |
|---|---|---|---|
| SIGHUP | 1 | Term | Hangup detected on controlling terminal or death of controlling process |
| SIGINT | 2 | Term | Interrupt from keyboard |
| SIGQUIT | 3 | Core | Quit from keyboard |
| SIGILL | 4 | Core | Illegal Instruction |
| SIGABRT | 6 | Core | Abort signal from abort(3) |
| SIGFPE | 8 | Core | Floating point exception |
| SIGKILL | 9 | Term | Kill signal |
| SIGSEGV | 11 | Core | Invalid memory reference |
| SIGPIPE | 13 | Term | Broken pipe: write to pipe with no readers |
| SIGALRM | 14 | Term | Timer signal from alarm(2) |
| SIGTERM | 15 | Term | Termination signal |
| SIGUSR1 | 30,10,16 | Term | User-defined signal 1 |
| SIGUSR2 | 31,12,17 | Term | User-defined signal 2 |
| SIGCHLD | 20,17,18 | Ign | Child stopped or terminated |
| SIGCONT | 19,18,25 | | Continue if stopped |
| SIGSTOP | 17,19,23 | Stop | Stop process |
| SIGTSTP | 18,20,24 | Stop | Stop typed at tty |
| SIGTTIN | 21,21,26 | Stop | tty input for background process |
| SIGTTOU | 22,22,27 | Stop | tty output for background process |

# Señales desde el intérprete de órdenes

- Podemos enviar señales desde el intérprete de órdenes mediante la órden `kill`.
- Sintaxis (copiado de "man kill"):
  - listar señales: `kill -l`.
  - enviar señales: `kill -s señal identificador_de_proceso`.
- También podemos enviar señales mediante combinaciones de teclas:
  - SIGINT: control+c.
  - SIGSTOP: control+z.
- Los procesos detenidos temporalmente mediante la señal SIGSTOP pueden volver a ejecutarser enviandoles la señal SIGCONT. Esto puede hacerse de forma cómoda a través de las órdenes...
  - `fg`: reanuda un proceso y lo pasa a primer plano.
  - `bg`: reanuda un proceso y lo pasa a segundo plano.

# Ejemplo: gestión de procesos mediante señales

1000.sh

```
!/usr/bin/env bash

for (( i=0; i<1000; ++i )) do
    echo -n "$i "
    sleep 0.250
done
```

- Ejecute el siguiente script.
- Pruebe a detenerlo y reanudarlo desde el intérprete de órdenes de las diferentes formas vistas hasta ahora.

# Ejemplo: alarma.c I

```c
const int N = 10; // número de alarmas

const int ESPERA = 1; // segundos

void atrapar_alarma(int); // declaración anticipada

//-------------------------------------------------------------

void instalar_alarma()
{
    signal(SIGALRM, atrapar_alarma); // instalar manejador de señal
}

//-------------------------------------------------------------

void atrapar_alarma(int i) // manejador de señal
```

# Ejemplo: alarma.c II

```
{
    // instalar_alarma();     // reinstalar tras cada alarma?
    printf("¡¡¡ALARMA!!!\n");
}

//------------------------------------------------------------

int main()
{
    instalar_alarma(); // instalar alarma por primera vez
    for (int i = 0; i < N; ++i)
    {
        alarm(ESPERA);
        pause();
    }
}
```

# Ejemplo: alarma2.cc I

```
void alarma(int)
{
    std::cout << "alarma" << std::endl;
}

int main()
{
    signal(SIGALRM, alarma);
    alarm(1); // 1s

    std::cout << "hola" << std::endl;
    pause();
    std::cout << "adios" << std::endl;
}
```

# Sincronización entre hebras mediante señales

- Escriba un programa con dos hebras, o procesos, que deberán llamarse **cliente** y **servidor**... en realidad con uno basta.
- El trabajo de la hebra **cliente** consiste en **enviar** la señal **SIGUSR1** a la hebra servidor.
- El trabajo de la hebra **servidor** consiste en **enviar** la señal **SIGUSR2** a la hebra cliente cada vez que reciba desde esta la señal SIGUSR1.
- La hebra cliente debe mostrar dos mensajes por pantalla:
  - "`[cliente ]: petición enviada`"
  - "`[cliente ]: respuesta recibida`"
- La hebra servidor debe mostrar dos mensajes por pantalla:
  - "`[servidor]: petición recibida`"
  - "`[servidor]: respuesta enviada`"
- Cliente y servidor deben finalizar al cabo de 1 segundo.

# Responda a las siguientes preguntas...

- ¿Los ejemplos **funcionan** correctamente?
- ¿Se puede prescindir de la **alarma** para finalizar los programas?... `timeout`.
- ¿Afecta el mecanismo de finalización la comparativa?
- ¿Qué versión cree **mejor** y por qué?