

# Arquitectura de Sistemas

## Práctica 4: Procesos

---

Gustavo Romero López

Updated: 22 de marzo de 2023

Departamento de Ingeniería de Computadores, Automática y Robótica

- ⊙ Aprender a gestionar procesos:
  - creación: `fork()`.
  - cambio de la imagen: `exec()`.
  - terminación: `exit()`.
  - comunicación: `exit()` + `wait()`.
  - identificador de proceso: `getpid()`.
- ⊙ Medición de tiempos.
- ⊙ Comunicación entre procesos
  - padre/hijo.
  - memoria compartida.
  - paso de mensajes.

```
ATT = $(EXE:=.att)
DAT = $(EXE:=.data) $(EXE:=.data.old)
EXE = $(basename $(SRC))
LOG = $(EXE:=.log)
SRC = $(wildcard *.c *.cc)

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

all: $(EXE)

att: $(ATT)

clean:
    -rm -fv $(ATT) $(DAT) $(EXE) $(LOG) core.* *~

exe: $(EXE)

medir.nulo.gb: CFLAGS+=-lbenchmark

%.att: %
    objdump -Cd $< > $@
```

# Creación de un proceso nuevo

- ⊙ Mediante `fork()` podemos crear un nuevo proceso.
- ⊙ El nuevo proceso es una copia idéntica del proceso original, indistinguible salvo por el identificador de proceso. Este se puede consultar mediante `getpid()`.
- ⊙ Esta llamada la sistema tiene la peculiaridad de que es ejecutada desde un único proceso pero retorna a dos: el original, denominado **proceso padre** y, si todo ha ido bien, otro nuevo proceso denominado **proceso hijo**.
- ⊙ El valor de retorno de `fork()` es:
  - **-1** en caso de error.
  - **0** en el proceso hijo.
  - **El identificador del proceso hijo** en el proceso padre.
- ⊙ Pruebe el siguiente programa de ejemplo y busque el código que se ejecuta en realidad.

```
5  #include <unistd.h>
6  #include <sys/wait.h>
7  #include <iostream>
8
9  //-----
10
11  int main()
12  {
13      switch(fork())
14      {
15          case -1: std::cout << "fallo en fork()!";      break;
16          case 0: std::cout << "hijo";                  break;
17          default: wait(nullptr); std::cout << "padre"; break;
18      }
19      std::cout << "\t [" << getpid() << "]" << std::endl;
20  }
```

# Medición de tiempos (1)

Métodos para medir tiempo:

- ⊙ `std::clock::high_resolution_clock`  $\implies$  precisión: nanosegundos.
- ⊙ `clock_gettime`  $\implies$  precisión: nanosegundos.
- ⊙ `gettimeofday`  $\implies$  precisión: microsegundos.
- ⊙ `getrusage`  $\implies$  precisión: microsegundos.
- ⊙ `clock`  $\implies$  precisión: ticks del reloj.
- ⊙ `time`  $\implies$  precisión: milisegundos.

## Medición de tiempos (2)

- ⊙ Modifique el programa anterior de forma que pueda medir el tiempo que se tarda en ejecutar un proceso nulo.
- ⊙ El proceso nulo es tan sencillo como: `'int main() {}'`.
- ⊙ Además de `fork()`, ahora deberá utilizar las llamadas a `wait()` y `exec()`.
- ⊙ Ahora el proceso padre deberá esperar a que acabe el proceso hijo con `wait()`.
- ⊙ El proceso hijo cambiará su imagen para ejecutar el proceso nulo mediante `execl()`.

¿Cómo hacerlo mejor? → siguiente práctica

- ⊙ Goole Benchmark
- ⊙ perf / valgrind

## Comunicación entre procesos: ejemplo padre/hijo

- ⦿ La comunicación entre procesos más básica es el valor que un proceso hijo devuelve a su proceso padre.
  - El proceso hijo devuelve el valor mediante el retorno de `main()` o utilizando `exit()`.
  - El proceso padre recupera el valor devuelto por el hijo mediante `wait()` o `waitpid()`.
- ⦿ Modifique el programa de ejemplo original `fork.cc` para que el proceso padre reciba un valor desde el proceso hijo.
- ⦿ El valor recibido lleva más información que el valor que se desea comunicar. Para descartar el resto podemos utilizar la macro `WEXITSTATUS()`.



- ⊙ Busque información sobre las funciones `shmget()` y `shmat()` de la cabecera `#include <sys/shm.h>`.
- ⊙ Escriba un programa que envíe un mensaje entre dos procesos haciendo uso de memoria compartida.
- ⊙ El proceso es sencillo combinando el uso de:
  - `shmget()`: para conseguir memoria compartida.
  - `shmat()`: para obtener acceso a ella.
- ⊙ Podemos utilizar `fork()` opcionalmente si deseamos una relación padre/hijo entre los dos procesos.

# Comunicación entre procesos mediante paso de mensajes

- ⦿ Busque información sobre las funciones `msgctl()`, `msgget()`, `msgsnd()` y `msgrcv()` de la cabecera `#include <sys/msg.h>`.
- ⦿ Escriba un programa que envíe un mensaje entre dos procesos haciendo uso de este tipo de paso de mensajes.

- ⊙ Para estudiar y practicar la comunicación entre procesos vamos a implementar el programa ping/pong:
  - Dos programas deben enviar y recibir mensajes...
  - Servidor: espera recibir ping y responde con pong.
  - Cliente: envía ping y espera recibir pong.
- ⊙ Mecanismos de implementación (**sin ejemplo**):
  - Memoria compartida entre procesos: mmap.
  - Colas de mensajes (System V): msgget/msgsnd.
  - **Colas de mensajes (POSIX): mq\_open/mq\_send/mq\_receive.**
  - Tuberías: pipe.
  - Memoria compartida entre hebras: pthreads/C++11.
  - **Sockets: socket/bind/listen/read/write/close.**
- ⊙ Estudie los códigos de muestra y cree su propio programa siguiendo el mismo esquema: <https://pccito.ugr.es/~gustavo/as/practicas/ping-pong>.

# Comunicación entre procesos mediante sockets

