# *Prácticas de Sistemas operativos*

David Arroyo Guardeño

Escuela Politécnica Superior de la Universidad Autónoma de Madrid

## Quinta semana: señales

# *Segunda práctica*

1. Hilos: ejercicios 3 y 4
2. Señales: ejercicios 6, 8 y 10

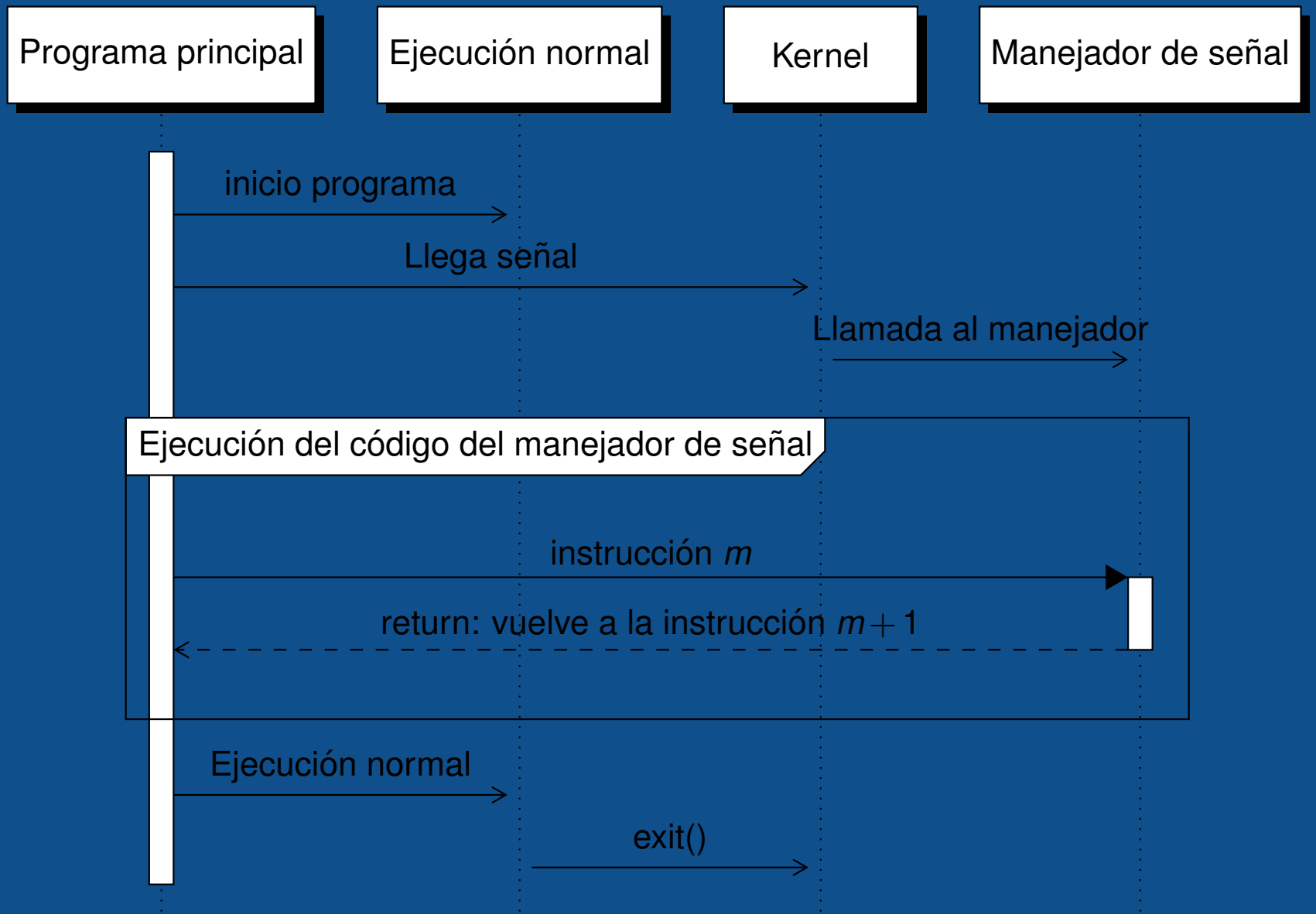
Jueves 5 de marzo

# *Entregas*

- Ejercicio3
- Ejercicio 4

# ¿Qué es una señal?

- ✓ Notificación que recibe un proceso sobre la ocurrencia de un evento: Interrupción software



# *Recepción de una señal: posibles efectos I*

- ✓ La señal es ignorada: el kernel la descarta $\rightarrow$ no efecto sobre el proceso
- ✓ El proceso es eliminado (*killed*)
  - $\sim$ Abnormal process termination
  - ✗ Terminación normal: exit()
- ✓ Se genera un fichero *core dump* y se termina el proceso
  - ✓ Imagen de la memoria virtual del proceso
  - $\Rightarrow$ Posterior depuración
- ✓ Se para el proceso (*stopped*): proceso en suspensión

# *Recepción de una señal: posibles efectos II*

- ✓ Se restablece el proceso (*resumed*) al estado previo a su suspensión

# *Configurar la respuesta de un proceso a una señal: setting the disposition of a signal*

- ✓ Se ejecuta la acción por defecto
- ✓ La señal es ignorada
- ✓ Se usa una rutina específica que hace las veces de manejador de señal

# *¿Qué es un manejador de una señal (signal handler)?*

- ✓ Una función creada por el programador para generar una respuesta determinada cuando se produce un cierto evento
  - $\Rightarrow$ Shell: *Control* $-$ *C* $\rightarrow$ SIGINT $\rightarrow$ para el proceso en ejecución y se vuelve al bucle de entrada principal (*prompt* de la shell)
- ✓ Instalar o establecer un manejador de señal
  - ✗ Notificación al kernel del programa a ejecutar en caso de que se reciba la señal relativa
- ✓ Ejecución del manejador tras recibir la señal
  - ✗ La señal está siendo tratada, *manejada* (*handled*) o capturada (*caught*)

# *Envío de señales*

```
#include <signal.h>
int kill(pid_t pid, int sig);
```

- ✓ $pid > 0 \rightarrow$ proceso al que enviamos la señal
- ✓ $pid = 0 \rightarrow$ la señal se envía a todos los procesos $\in$ mismo grupo
- ✓ $pid = -1 \rightarrow$ se envía a todos los procesos | identificador real $=$ identificador efectivo del proceso que la envía
- ✓ $pid < -1 \rightarrow$ procesos identificador de grupo $=$ valor absoluto de pid

```
uid_t getuid(void);
uid_t geteuid(void);
gid_t getgid(void);
gid_t getegid(void);
int setpgid(pid_t pid, pid_t pgid);
pid_t getpgid(pid_t pid);
int setpgrp(void);
pid_t getpgrp(void);
```

# *Tratamiento de señales*

```c
#include <signal.h>
void (* signal(int sig, void(*handler)(int)))(int);

#include <signal.h>
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t handler);
```

# Ejemplo 1

```c
#include <stdio.h>
#include <signal.h>
#include <stdlib.h>

int main(int argc, char *argv[],char *env[])
{
  void manejador_SIGINT();

  if(signal (SIGINT,manejador_SIGINT)==SIG_ERR){
    perror("signal");
    exit(EXIT_FAILURE);
  }
  while(1){
    printf("En espera de Ctrl+C\n");
    sleep(100);
  }
}

void manejador_SIGINT(int sig)
{
  printf("Señal número %d recibida \n",sig);
}
```

- ✓ stty -a
- ✓ man stty

# Ejemplo2

```c
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <stdio.h>
#define NUM_HIJOS 3
/* Handle SIGCHLD signals. */
void handle_sigchld(int sig){
  pid_t pid;
  int status;

  pid = wait(&status);

  printf("Hijo totalmente eliminado %d\n",pid);
  sleep(1);
}


int main(){
  int i;
  signal(SIGCHLD, handle_sigchld);

  for(i=0;i < NUM_HIJOS;i++){
    if(fork()==0){
      printf("Desde el hijo, hola %d\n",getpid());
      sleep(5);
      return 0;
    }
```

# Ejemplo3 I

```
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_HIJOS 3
/* Handle SIGCHLD signals. */
void handle_sigchld(int sig){
  pid_t pid;
  int status;

  while(1){
    pid = waitpid(-1,&status,WNOHANG);
    if(pid<=0) /*Ya no hay zombies*/
      break;
    /*En la practica, se debe evitar imprimir...*/
    printf("Hijo totalmente eliminado %d\n",pid);

  }
  sleep(1);
}


int main(){
```

# *Ejemplo3 II*

```
    int i;
    signal(SIGCHLD, handle_sigchld);

    for(i=0;i < NUM_HIJOS;i++){
      if(fork()==0){
        printf("Desde el hijo, hola %d\n",getpid());
        sleep(5);
        exit(EXIT_SUCCESS);
      }
    }

    while(1){
      pause();
    }
    return 0;
}
```

# *Ejemplo 4*

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h>
#include <signal.h>

int main(int argc, char *argv[]) {
  sigset_t intmask;

  sigemptyset(&intmask);
  sigaddset(&intmask, SIGINT);
  while(1){
    sigprocmask(SIG_BLOCK, &intmask,NULL);
    printf("señal SIGINT bloqueada\n");
    sleep(10);
    printf("10 segundos con señal bloqueada\n");
    sigprocmask(SIG_UNBLOCK, &intmask, NULL);
    printf("señal SIGINT desbloqueada\n");
    sleep(10);
    printf("5 segundos con señal desbloqueada\n");
  }
  return EXIT_SUCESS;
}
```

# Ejemplo 5

```
int interrumpido;
void interrumpe(int d) {
  interrumpido = 1;
}
void mata(char *s) {
  printf("%s\n", s);
  exit(EXIT_SUCCESS);
}
int main() {
  struct sigaction sa;  int n;   char c;

  sa.sa_handler = interrumpe;   sigemptyset(&sa.sa_mask);   sa.sa_flags =
      0;
  if (sigaction(SIGINT, &sa, NULL))
    mata("sigaction-SIGINT");
  sa.sa_flags = SA_RESTART;
  if (sigaction(SIGQUIT, &sa, NULL))
    mata("sigaction-SIGQUIT");

  interrumpido = 0;
  n = read(0, &c, 1);
  if (n == -1 && errno == EINTR)
    printf("read call was interrupted\n");
  else if (interrumpido)
    printf("read call was restarted,c=%c\n",c);
  return 0;
}
```

# *Ejemplo 6 I*

```
void imprime_mascara()
{
  sigset_t sigset;
  int errno_save;

  errno_save = errno;
  if (sigprocmask(0, NULL, &sigset) < 0)
    perror("Sigprocmask");
  printf("sigmask = ");
  if (sigismember(&sigset, SIGINT)) printf("SIGINT ");
  if (sigismember(&sigset, SIGQUIT)) printf("SIGQUIT ");
  if (sigismember(&sigset, SIGUSR1)) printf("SIGUSR1 ");
  if (sigismember(&sigset, SIGUSR2)) printf("SIGUSR2 ");
  if (sigismember(&sigset, SIGALRM)) printf("SIGALARM ");
  if (sigismember(&sigset, SIGABRT)) printf("SIGABRT ");
  if (sigismember(&sigset, SIGCHLD)) printf("SIGCHLD ");
  if (sigismember(&sigset, SIGHUP)) printf("SIGHUP ");
  if (sigismember(&sigset, SIGTERM)) printf("SIGTERM ");
  printf("\n");
  errno = errno_save;
}

int main() {
  sigset_t mascara;
```

# *Ejemplo 6 II*

```
    sigset_t mascara_antigua;

    sigemptyset(&mascara);
    sigaddset(&mascara, SIGQUIT);
    sigaddset(&mascara, SIGUSR2);
    sigprocmask(SIG_BLOCK, &mascara, &mascara_antigua);
    imprime_mascara();
    while(1) pause();
}
```

# *Referencias*


- Francisco M. Márquez. Unix, Programación Avanzada. Editorial: Ra-Ma. 3ª Edición. ISBN: 84-7897-603-5