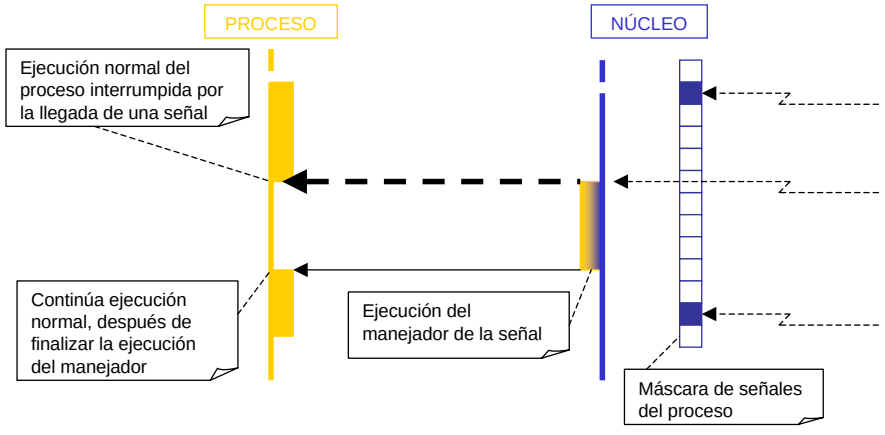
Manejo de Señales

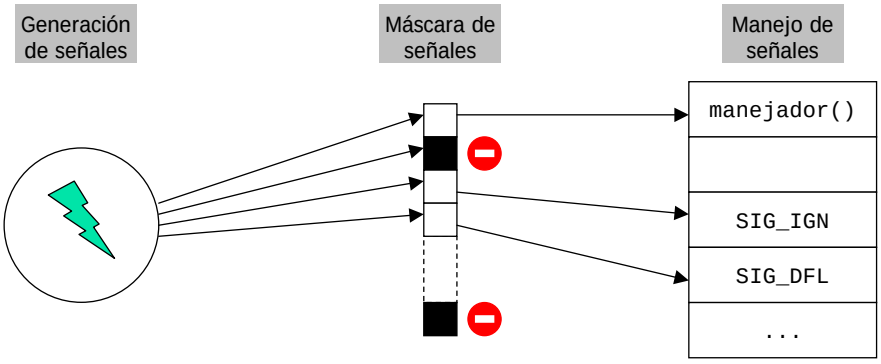
Sistemas Operativos (prácticas)
E.U. Informática en Segovia
Universidad de Valladolid

Señales

- Concepto
  - Evento que interrumpe la ejecución normal de un proceso
  - La acción por defecto suele ser terminar el proceso que la recibe
  - Pueden manejarse (es decir, programarlas para obtener un comportamiento diferente)
  - La mayoría de las señales pueden filtrarse mediante una máscara



Señales



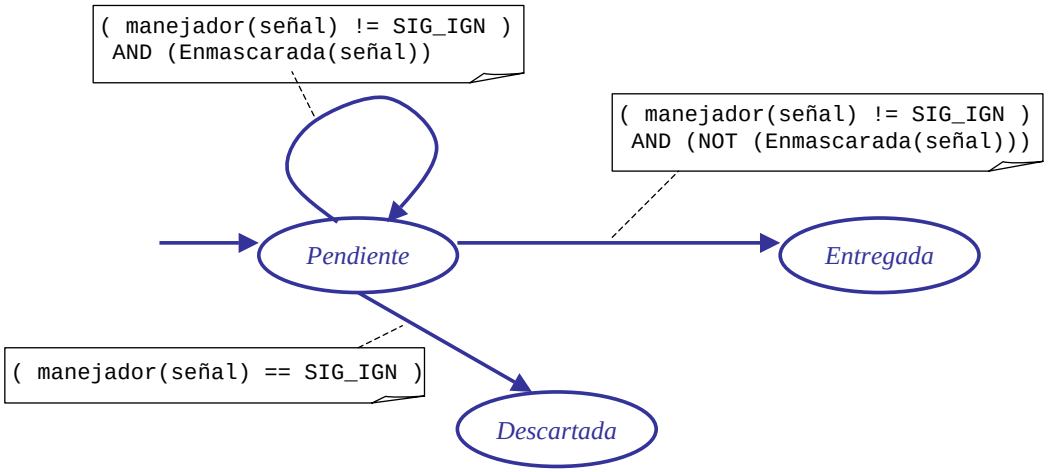
- Eventos externos (terminal, terminación hijo, ...)
- Errores
- Señales software
  - `kill()`
  - `raise()`
  - `alarm()`

- Definición máscara
  - `sigemptyset()`
  - `sigfillset()`
  - `sigaddset()`
  - `sigdelset()`
  - `sigismember()`
- Establecimiento máscara
  - `sigprocmask()`

- Establecimiento manejador
  - `sigaction()`
- Espera de señales
  - `pause()`
  - `sigsuspend()`

Señales

- Estado de una señal
  - El núcleo del SO memoriza el estado de una señal, dependiendo de si la señal está enmascarada o no y si se ignora o no



Señales: llamadas al sistema

Señales	
`kill`	Enviar una señal
`alarm`	Generar una alarma (señal de reloj)
`sigemptyset`	Iniciar una máscara de señales vacía
`sigfillset`	Iniciar una máscara de señales con todas las señales
`sigaddset`	Añadir una señal específica a una máscara
`sigdelset`	Eliminar una señal específica de una máscara
`sigismember`	Comprobar si una señal pertenece o no a una máscara
`sigprocmask`	Consultar/establecer la máscara de señales para un proceso
`sigaction`	Capturar/manejar señales
`sigsuspend`	Esperar por una señal

Lista de señales (representativas)

Nº	Señal	Descripción	Acción por defecto
1	**SIGHUP**	Colgado/muerte terminal de control	exit
2	**SIGINT**	Interrupción (`Ctrl-C`)	exit
3	**SIGQUIT**	Interrupción con *core* (`Ctrl-\`)	core
6	**SIGABRT**	Terminación anormal	core
9	**SIGKILL**	Terminación. Ni manejable, ni enmascarable	exit
13	**SIGPIPE**	Escritura en una tubería sin lector	exit
14	**SIGALRM**	Alarma temporizada, programada por `alarm()`	exit
15	**SIGTERM**	Terminación por software	exit
--	**SIGUSR1**	Señal 1 definida por el usuario	exit
Control de trabajos			
17	**SIGSTOP**	Alto (no puede capturarse, ni enmascararse)	stop
18	**SIGTSTP**	Alto de teclado (`Ctrl-Z`)	stop
19	**SIGCONT**	`Ctrl-C` de teclado (interrupción)	continue
20	**SIGCHLD**	Terminación/suspensión de un proceso hijo	ignore

Señales: `kill`

- **`kill`**: enviar una señal

```
#include <sys/types.h>
#include <signal.h>

int signal(pid_t pid, int sig)
```

- Descripción
  - Envía la señal **`sig`** al proceso o grupo de procesos **`pid`**
  - Si `(pid==0)` la señal se envía al grupo de procesos del emisor (salvo a sí mismo)
  - Si `(pid==-1)` y el `eUID` del emisor es `root`, la señal se envía a todos los proceso excepto los del sistema
  - El `eUID` del proceso que envía la señal debe coincidir con el `eUID` del que la recibe o ser `root`.
- Valor de retorno
  - 0 si la señal es enviada y –1 si hay error
- Errores
  - Número de señal no válido
  - Permisos

Señales: `alarm`

- **`alarm`**: establecer un temporizador

```
#include <unistd.h>

unsigned int alarm(unsigned int sig)
```

- Descripción
  - Programa la señal **`SIGALRM`** para que ocurra al cabo de un cierto número de segundos (los especificados por el parámetro `seconds`)
  - Un proceso sólo puede tener una petición de alarma pendiente. Las peticione ssucesivas de alarmas no se encolan, cada nueva petición anula la anterior
  - `alarm(0)` cancela las alarmas
- Valor de retorno
  - Tiempo restante para que venciese la alarma anterior

Señales: tratamiento de máscaras

- Llamadas al sistema para definir una máscara de señales

```
#include <signal.h>
int sigemptyset (sigset_t *set)
int sigfillset  (sigset_t *set)
int sigaddset   (sigset_t *set, int signum)
int sigdelset   (sigset_t *set, int signum)
int sigismember (const sigset_t *set, int signum)
```

- Descripción
  - **sigemptyset**: inicia una máscara para que no tenga señales seleccionadas
  - **sigfillset**: inicia una máscara para que contenga todas las señales
  - **sigaddset**: pone una señal específica en un conjunto de señales
  - **sigdelset**: quita una señal específica de un conjunto de señales
  - **sigismember**: consulta si una señal pertenece a un conjunto de señales

Señales: **sigprocmask**

- **sigprocmask**: examinar/modificar máscara de señales

```
#include <signal.h>
int sigprocmask (int how, const sigset_t *set,
                 sigset_t *o_set)
```

- Descripción
  - Examina o modifica la máscara de señales activa para un proceso
  - Si una señal está bloqueada por la máscara, no es procesada hasta el momento en que deja de estar bloqueada por la máscara (se memoriza que llegó)
  - Parámetro **how**
    - **SIG_BLOCK**: añade un conjunto de señales a las que se encuentran bloqueadas en la máscara actual
    - **SIG_UNBLOCK**: elimina un conjunto de señales de las que se encuentran bloqueadas en la máscara actual
    - **SIG_SETMASK**: especifica un conjunto de señales que serán bloqueados
  - Parámetro **set**: conjunto de señales que serán utilizadas para la modificación (puede ser **NULL**)
  - Parámetro **oset**: máscara previa a la modificación (puede ser **NULL**)
- Valor de retorno
  - 0 si funciona y –1 si error (activa **errno**)

Señales: **sigprocmask** (ejemplo)

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

int main(int argc, char *argv[]) {
   double y;
   sigset_t intmask;
   int i, ncalc;

   if (argc!=2) {
      fprintf (stderr, "Uso: %s factor\n",
               argv[0]);
      return;
   }
   ncalc= 1000000*atoi(argv[1]);
   sigemptyset(&intmask);
   sigaddset(&intmask, SIGINT);
   for (;;) {
      sigprocmask(SIG_BLOCK, &intmask,
                  NULL);
      fprintf (stdout, "señal SIGINT
               bloqueada\n");
      for (i=0; i<ncalc; i++)
         y= sin((double)i);
      fprintf (stdout, "Finalizado cálculo
               con señal bloqueada\n");

      sigprocmask(SIG_UNBLOCK, &intmask,
                  NULL);
      fprintf (stdout, "señal SIGINT
               desbloqueada\n");
      for(i=0; i<ncalc; i++)
         y= sin((double)i);
      fprintf (stdout, "Finalizado cálculo
               con señal desbloqueada\n");
   }
}
```


Señales: **sigaction**

- **sigaction**: capturar/manejar señales

```
#include <signal.h>

int sigaction (int signo, const struct sigaction *act,
               struct sigaction *oact)

struct sigaction {
  void (*sa_handler)();/* SIG_DFL, SIG_IGN o función */
  sigset_t sa_mask;    /* las señales especificadas por
                          la máscara serán bloqueadas durante
                          la ejecución del manejador */
  int sa_flags;        /* indicadores y opciones */
}
```

Señales: **sigaction**

- Descripción
  - Instala los manejadores (*handlers*) de señal de un proceso
  - La señal se especifica en **signo** y los manejadores en **act**, que es una estructura del tipo **struct sigaction**. El manejador anterior se devuelve en **oact**
  - Si act es **NULL**, el manejador no cambia. Si no se necesita **oact**, puede especificarse **NULL**
  - Los manejadores son “permanentes” (no es necesario volverlos a instalar después de la ocurrencia de una señal)
  - A los manejadores no se les pueden pasar parámetros
  - **SIG_DFL**: instala la acción por defecto (exit, core, ...)
  - **SIG_IGN**: se maneja la señal ignorándola (diferente a bloquearla)
- Valor de retorno
  - 0 si funciona y –1 si error
- Errores
  - Señal inválida o no manejable (**SIGKILL**, **SIGSTOP**)

Señales: **sigaction** (ejemplo)

```
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <stdio.h>
#include <math.h>
#include <string.h>

char msg[]="Pulsado Ctrl-C\n";
struct sigaction act;

int main(void) {
   double y; int i;
   capturar_ctrl_c();
   fprintf (stdout, "Capturado CRTL-C
            durante 10 seg\n");
   for (i=0; i<100000000; i++)
      y= sin((double)i);
   ignorar_ctrl_c();
   fprintf (stdout, "Ignorado CTRL-C
            durante 10 seg\n");
   for (i=0; i<100000000; i++)
      y= sin((double)i);
   fprintf (stdout, "Fin programa\n");
}
```

```
void manejador_ctrl_c(int signo) {
   fprintf(stdout, "%s", msg);
   fflush(stdout);
}


int capturar_ctrl_c() {
   act.sa_handler= manejador_ctrl_c;
   sigemptyset(&act.sa_mask);
   act.sa_flags= 0;
   return sigaction(SIGINT, &act, NULL);
}


int ignorar_ctrl_c() {
   if (sigaction(SIGINT, NULL, &act)==-1){
      perror("No se puede recuperar
              manejador anterior SIGINT");
      return -1;
   } else if (act.sa_handler==manejador_ctrl_c) {
      act.sa_handler= SIG_IGN;
      return sigaction(SIGINT, &act, NULL);
   }
}
```

Señales: **sigsuspend**

- **sigsuspend**: esperar señales

```
#include <signal.h>
int sigsuspend(const sigset_t *sigmask)
```

- Descripción
  - Establece la máscara de señales **sigmask** y suspende el proceso hasta que reciba una señal no enmascarada en **sigmask**
  - Cuando retorna, se restablece la máscara de señales previa
  - Si el proceso captura la señal, retorna después de ejecutar el manejador establecido
- Valor de retorno
  - 0 si éxito y –1 si error

Señales: **sigsuspend** (ejemplo)

```
#include <unistd.h>
#include <signal.h>
#include <stdio.h>

char msg1[]="Señal capturada\n";
char msg2[]="Temporizador vencido\n";
struct sigaction act;

int signum, s_recibida = 0;

void handler(int signo) {
   if (signo != SIGALRM)
      fprintf(stdout, "%s", msg1);
   else {
      s_recibida= 1;
      fprint(stdout, "%s", msg2);
   }
   fflush(stdout);
}
```

```
int main(int argc, char *argv[]) {
   sigset_t sigset;
   int i;

   if (arg != 2) {
      fprintf(stderr, "Uso: %s sig\n", argv[0]);
      return;
   }
   signum=atoi(argv[1]);
   act.sa_handler= handler;
   sigfillset(&act.sa_mask);
   act.sa_flags= 0;
   sigaction(SIGALRM, &act, NULL);
   sigaction(signum, &act, NULL);
   sigfillset(&sigset);
   sigdelset(&sigset, signum);
   sigdelset(&sigset, SIGALRM);

   fprintf(stdout, "esperando señal %s...\n",
           argv[1]);
   alarm(15);
   while (s_recibida == 0) sigsuspend(&sigset);
   fprintf(stdout, "Fin programa\n");
}
```