SISTEMAS OPERATIVOS: PROCESOS

Señales

ADVERTENCIA

- Este material es un simple guión de la clase: no son los apuntes de la asignatura.
- El conocimiento exclusivo de este material no garantiza que el alumno pueda alcanzar los objetivos de la asignatura.
- Se recomienda que el alumno utilice los materiales complementarios propuestos.

Contenido

- Señales.
- Temporizadores.
- Excepciones.
- Entorno de un proceso

Señales. Concepto

- Las señales son un mecanismo para comunicar eventos a los procesos

Son excepciones asíncronas.

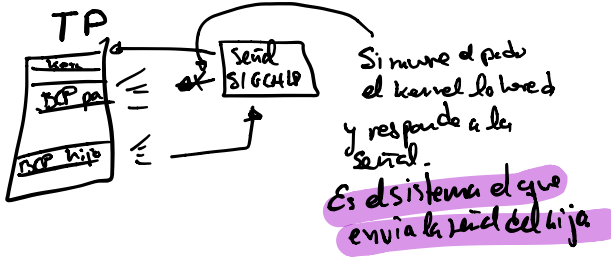
- Cuando un proceso recibe una señal, la procesa inmediatamente.
- Cuando un proceso recibe una señal puede:
  - Ignorar a la señal, cuando es inmune a la misma. SIGIGNORE
  - Invocar la rutina de tratamiento por defecto,
  - Invocar a una rutina de tratamiento propia

Lo general → ctrl+q +c

normalmente es muerte si no esperaba esa señal.

Señales. Concepto

- Permite avisar a un proceso de la ocurrencia de un evento.
  - Y reaccionar a dicho evento
- Ejemplos:
  - Un proceso padre recibe la señal SIGCHLD cuando termina un proceso hijo.
  - Un proceso recibe una señal SIGILL cuando intenta ejecutar una instrucción máquina ilegal.



Son un mecanismo propio de los sistemas UNIX

Señales: interrupciones al proceso

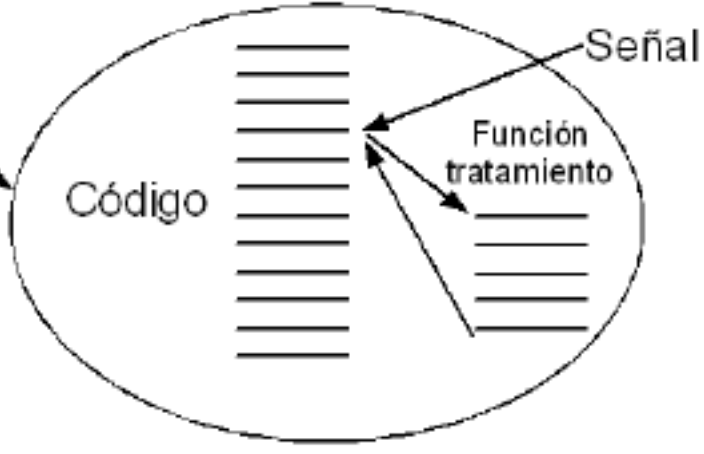
- Las señales son interrupciones al proceso
- Envío o generación
  - Proceso- Proceso (dentro del grupo) con el kill
  - SO - Proceso

Pueden llegar en cualquier momento, por lo que la debe tratar este como este el proceso, ya que si no muere.
No es como una func. a la que se llama.

Si esta ejecutando normal la rutina se hace en el siguiente ciclo. Y si se captura, después continua

Si esta en un bucle, lo hace en ese momento ya que le puede pedir 10 o 20 ciclos.

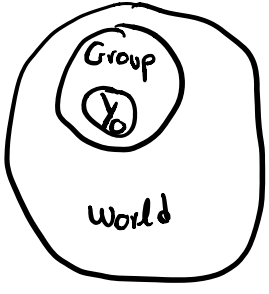
Las señales se envían sabiendo el pid del proceso.



Señales. Tratamiento

- El SO las transmite al proceso
  - El proceso debe estar preparado para recibirla
    - Especificando un procedimiento de señal con sigaction.
    - Dirección de la rutina de tratamiento (nombre)
- Enmascarando la señal con sigprogmask.
- Ignorando la señal
  - Trtamiento con SIG_IGN
- Si no está preparado → acción por defecto (SIG_DFL)
  - El proceso, en general, muere.
  - Hay algunas señales que se ignoran o tienen otro efecto.

Niveles



Param grupo es con: uid y gid (uid) (groupid)

- Para enviar una señal a un proceso:
  - int kill(pid_t pid, int sig);

~Ojo no es el kill normal, pero si no está preparado muere. Se usa cuando le dejas en background.

- El servicio **pause()** para el proceso hasta que recibe una señal

El proceso espera hasta que le llegue una señal de reanudar. Sirve para que se ejecuten en orden.

Señales. Envío y espera

- Para enviar una señal a un proceso:

  int kill(pid_t pid, int sig);

  - CTRL-C -> SIGINT
  - CTRL-Z -> SIGSTOP

- Un proceso se puede enviar una señal a sí mismo

  #include <signal.h>
  int raise(int sig);

- El servicio **pause()** bloquea el proceso hasta que recibe una señal cualquiera

Lista de señales

- El sistema operativo cuenta con un conjunto definido de señales
  - Archivo en "include": signal.h
- Señales importantes:
  - SIGILL instrucción ilegal
  - SIGALRM vence el temporizador
  - SIGKILL mata al proceso
  - SIGSGEV violación segmento memoria
  - SIGUSR1 y SIGUSR2 reservadas para el uso del programador.

En realidad son números, pero por comodidad usan esos nombres.

Señales predefinidas


```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR      31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1
36) SIGRTMIN+2  37) SIGRTMIN+3  38) SIGRTMIN+4  39) SIGRTMIN+5
40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8  43) SIGRTMIN+9
44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13
52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9
56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6  59) SIGRTMAX-5
60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2  63) SIGRTMAX-1
64) SIGRTMAX
```

Servicios POSIX para la gestión de señales

- `int kill(pid_t pid, int sig)`
  - Envía al proceso "pid" la señal "sig".
  - Casos especiales:
    - pid=0 → Señal a todos los procesos con gid igual al gid del proceso.
    - pid < -1 → Señal a todos los proceso con gid igual al valor absolute de pid.

- `int sigaction(int sig, struct sigaction *act, struct sigaction *oact)`
  - Permite especificar la acción a realizar como tratamiento de la señal "sig"
  - La configuración anterior se puede guardar en "oact".

Puede almacenar parametros

La estructura sigaction

Los parametros del sigaction que va en estas struct

```
struct sigaction {
  void (*sa_handler)(); /* Manejador */
  sigset_t sa_mask; /* Señales bloqueadas */
  int sa_flags;     /* Opciones */
};
```

- Manejador:
  - SIG_DFL: Acción por defecto (normalmente termina el proceso).
  - SIG_IGN: Ignora la señal.
  - Dirección de una función de tratamiento.
- Máscara de señales a bloquear durante el manejador.
- Opciones normalmente a cero.

Conjuntos de señales

- int sigemptyset(sigset_t * set);
  - Crea un conjunto vacío de señales.
- int sigfillset(sigset_t * set);
  - Crea un conjunto lleno con todas la señales posibles.
- int sigaddset(sigset_t * set, int signo);
  - Añade una señal a un conjunto de señales.
- int sigdelset(sigset_t * set, int signo);
  - Borra una señal de un conjunto de señales.
- int sigismember(sigset_t * set, int signo);
  - Comprueba si una señal pertenece a un conjunto.

Ejemplo

- Ignorar la señal SIGINT
  - Se produce cuando se pulsa la combinación de teclas Ctrl+C

```
struct sigaction act;
act.sa_handler = SIG_IGN;
act.flags = 0;
sigemptyset(&act.sa_mask);
Sigaction(SIGINT, &act, NULL);
```

Servicios POSIX para la gestión de señales

- `int pause(void)`
  - Bloquea al proceso hasta la recepción de una señal.
  - No se puede especificar un plazo para desbloqueo.
  - No permite indicar el tipo de señal que se espera.
  - No desbloquea el proceso ante señales ignoradas.

- int sleep(unsigned int sec)
  - Suspende un proceso hasta que vence un plazo o se recibe una señal.

Ejemplo: capturar SIGSEV


```
/*Programa que provoca que se eleve la seneal SIGSEGV
escribiendo en la posicion 0 de memoria la captura. */

#include …
#include <signal.h>

void capturar_senyal(int senyal){
   printf("Error por ocupacion indebida de memoria\n");
   signal(SIGSEGV,SIG_DFL);}
main(void){
   int *p;
   signal(SIGSEGV,capturar_senyal);
   printf ("Ya he colocado el manejador\n");
   p=0;
   printf ("Voy a poner un 5 en la variable\n");
   *p=5; }
```

Contenido

- Señales.
- **Temporizadores.**
- Excepciones.
- Entorno de un proceso

Temporizadores

- El sistema operativo mantiene un temporizador por proceso (caso UNIX).
  - Se mantiene en el BCP del proceso un contador del tiempo que falta para que venza el temporizador.
  - La rutina del sistema operativo actualiza todos los temporizadores.
  - Si un temporizador llega a cero se ejecuta la función de tratamiento.
- En UNIX el sistema operativo envía una señal SIGALRM al proceso cuando vence su temporizador.

Servicios POSIX para temporización

- `int alarm(unsigned int sec)` ticks
  - Establece un temporizador. → sec, µs, ns
  - Si el parámetro es cero, desactiva el temporizador.

llamada al sistema → alarm, se duerme exactamente el tiempo indicado, lo tiene medido

No es llamada al sistema. → sleep, como mínimo duerme el tiempo indicado.

Ejemplo: Imprimir un mensaje cada 10 segundos


```
#include <signal.h>
#include <stdio.h>
void tratar_alarma(void) {
  printf("Activada \n");
}

int main()  {
  struct sigaction act;

  /* establece el manejador para SIGALRM */
  act.sa_handler = tratar_alarma;
  act.sa_flags = 0; /* ninguna acción especifica */
  sigaction(SIGALRM, &act, NULL);

  act.sa_handler = SIG_IGN;             /* ignora SIGINT */
  sigaction(SIGINT, &act, NULL);
  for(;;){      /* recibe SIGALRM cada 10 segundos */
    alarm(10);
    pause();
  }
}
```

Finalización temporizada


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

pid_t pid;
void tratar_alarma(void) {
  kill(pid, SIGKILL);
}

main(int argc, char **argv) {
  int status;
  char **argumentos;
  struct sigaction act;
  argumentos = &argv[1];
  pid = fork();
  switch(pid) {
    case -1:   /* error del fork() */
      perror ("fork");
    exit(-1);
    case 0:    /* proceso hijo */
      execvp(argumentos[0], argumentos);
      perror("exec");
      exit(-1);
    default:   /* padre */
      /* establece el manejador */
      act.sa_handler = tratar_alarma;
      act.sa_flags = 0;
      sigaction(SIGALRM, &act, NULL);
      alarm(5);
      wait(&status);
  }
  exit(0);
}
```

Contenido

- Señales.
- Temporizadores.
- **Excepciones**
- Entorno de un proceso.

Excepciones

- El hardware detecta conditiones especiales:
  - Fallo de página, escritura a página de solo lectura, desbordamientos de pila, violación de segmento, syscall, …
- Transfiere control al SO para su tratamiento, que:
  - Salva contexto proceso
  - Ejecuta rutina si es necesario
  - Envía una señal al proceso indicando la excepción
- Las excepciones son una optimización de rendimiento:
  - Evitan mucho código extra de comprobación en programas

Exceptions

- Many languages (Java, C++, ..) use a mechanism know as *Exceptions* to handle errors at runtime
  - Try {} Catch (exception) {}
- In Java, for example, Exception is a class with many descendants.
  - ArrayIndexOutOfBoundsException
  - NullPointerException
  - FileNotFoundException
  - ArithmeticException
  - IllegalArgumentException

Contenido

- Señales.
- Temporizadores.
- Excepciones.
- **Entorno de un proceso.**

Concepto

Probar en Linux:
$ env
ó
$ getenv

- El entorno de un proceso se hereda del padre los siguientes datos:
  - Vector de argumentos con el que se ejecutó el comando del programa
  - Vector de entorno, una lista de variables <nombre, valor> que el padre pasa al hijo
- El paso de variable de entorno entre padre e hijo
  - Es una forma flexible de comunicar ambos procesos y determinar aspectos de la ejecución del hijo en modo usuario
- El mecanismo de las variables de entorno permite particularizar aspectos a nive de cada proceso particular
  - En lugar de tener una configuración común para todo el sistema

Entorno de un proceso

- Variables de entorno:
  - Mecanismo de paso de información a un proceso.
- ¿Se puede ver? Comando: "env"
- Ejemplo:

```
PATH=/usr/bin:/home/pepe/bin
TERM=vt100
HOME=/home/pepe
PWD=/home/pepe/libros/primero
TIMEZONE=MET
```

Entorno de un proceso

- El entorno de un proceso se coloca en la pila del proceso al iniciarlo.

- Acceso:
  - El sistema operativo coloca algunos valores por defecto (p. ej. PATH).
  - Acceso mediante mandatos (set, export).
  - Acceso mediante API de SO (putenv, getenv).

Paso del entorno

- Un proceso recibe como tercer parámetro de main la dirección de la tabla de variables de entorno.

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

int main(int argc, char** argv, char** envp) {
  for (int i=0;envp[i]!=NULL;i++) {
    printf("%s\n",envp[i]);
  }
  return 0;
}
```

Llamadas de entorno

- `char * getenv(const char * var);`
  - Obtiene el valor de una variable de entorno.
- `int setenv(const char * var, const char * val, int overwrite);`
  - Modifica o añade una variable de entorno.
- `int putenv(const char * par);`
  - Modifica o añade una asignación var=valor

Entorno de un proceso en Windows

- **`DWORD GetEnvironmentVariable(LPCTSTR lpszName, LPTSTR lpszValue, DWORD valueLenght);`**
  - Devuelve el valor de una variable de entorno.
- **`BOOL SetEnvironmentVariable(LPCTSTR lpszName, LPTSTR lpszValue);`**
  - Modifica o crea una variable de entorno.
- **`LPVOID GetEnvironmentStrings();`**
  - Obtiene un puntero a la tabla de variables de entorno.

Puntos a recordar

- Las señales POSIX se pueden ignorar o tratar.
- Los temporizadores tienen distinta resolución de POSIX in Win32.

Lecturas recomendadas

Básica

- Carretero 2007:
  - 3.6. Señales y excepciones.
  - 3.7. Temporizadores.
  - 3.13. Servicios.
  - 3.9 Threads

Complementaria

- Stallings 2005:
  - 4.1 Procesos e hilos.
  - 4.4 Gestión de hilos y SMP en Windows.
  - 4.5 Gestión de hilos y SMP en Solaris.
  - 4.6 Gestión de procesos e hilos en Linux.
- Silberschatz 2006:
  - 4 Hebras.

SISTEMAS OPERATIVOS: PROCESOS

Material complementario

Entorno de un proceso en Windows

- **`DWORD GetEnvironmentVariable(LPCTSTR lpszName, LPTSTR lpszValue, DWORD valueLenght);`**
  - Devuelve el valor de una variable de entorno.
- **`BOOL SetEnvironmentVariable(LPCTSTR lpszName, LPTSTR lpszValue);`**
  - Modifica o crea una variable de entorno.
- **`LPVOID GetEnvironmentStrings();`**
  - Obtiene un puntero a la tabla de variables de entorno.

Listado de variables de entorno en Windows


```
#include <windows.h>
#include <stdio.h>

int main() {
  char * lpszVar;
  void * lpvEnv;

  lpvEnv =GetEnvironmentStrings();
  if (lpvEnv == NULL) {
    exit(-1);
  }
  char * p = lpszVar;
  while (p!=NULL) {
    printf("%s\n",p);
    while (p!=NULL) p++;
    p++;
  }

  printf("\n");
  FreeEnvironmentStrings(lpszVar);

  return 0;
}
```

Temporizadores en Windows

- **`UINT SetTimer(HWND hWnd, UINT nIDEvent, UINT uElapse, TIMERPROC lpTimerFunc);`**
  - Activa un temporizador y ejecuta la funció lpTimerFunc cuando venza el tiempo.
  - La función debe cumplir con:
    - VOID TimerFunc(HWND hWnd, UINT uMsg, UINT idEvent, DWORD dwTime);
- **`BOOL KillTimer(HWND hWnd, UINT uIdEvent);`**
  - Desactiva un temporizador.
- **`VOID Sleep(DWORD dwMilliseconds);`**
  - Hace que el hilo actual se su suspenda durante un cierto tiempo.

Temporización de un mensaje en Windows


```
#include <windows.h>
#include <stdio.h>

VOID Mensaje(HWND,UINT,UINT,DWORD) {
  printf("Tiempo finalizado");
}

int main() {
  tid = SetTimer(NULL,2,10,Mensaje); /* 2 msec */
  realizar_tarea();
  KillTimer(NULL,tid);
  return 0;
}
```

SISTEMAS OPERATIVOS: PROCESOS

Hilos y Procesos