Sistemas Operativos 1/2025
Laboratorio 1: Desafío de Comunicación y Elección de Líderes

Profesores:
Cristóbal Acosta (cristobal.acosta@usach.cl)
Fernando Rannou (fernando.rannou@usach.cl)
Miguel Cárcamo (miguel.carcamo@usach.cl)
Ayudantes:
Ricardo Hasbun (ricardo.hasbun@usach.cl)
Daniel Calderón (daniel.calderon.r@usach.cl)

April 3, 2025

1 Introducción y Contexto de las Señales en Linux

Las señales son un mecanismo de comunicación asíncrona en sistemas Unix y Linux que permiten a los procesos y al sistema operativo notificar eventos especiales o de error. Una señal es, en esencia, un mensaje que se envía a un proceso para informarle de que debe realizar una acción (por ejemplo, terminar su ejecución o ejecutar un manejador específico).

En Linux, las señales se utilizan en distintos contextos, tales como:

- Notificar la finalización o la interrupción de un proceso.
- Indicar errores como divisiones por cero o violaciones de segmento.
- Implementar mecanismos de temporización, por ejemplo, con la función `alarm()`.
- Permitir la comunicación y sincronización entre procesos, lo cual es especialmente útil en programas concurrentes.

Este laboratorio, que constituye el primer ejercicio de Sistemas Operativos en el curso, tiene como objetivo introducir a los estudiantes en el manejo de señales en Linux, mostrando tanto la teoría como la práctica a través de un ejemplo en C.

2 Funciones Importantes en el Manejo de Señales

A continuación se detallan algunas de las funciones clave utilizadas en el ejemplo de señales:

- `fork()`
  Uso: Crea un nuevo proceso (proceso hijo) duplicando el proceso actual (proceso padre).
  En el ejemplo: Permite generar un proceso hijo al que se le enviará la señal.

- `sigaction()`
  Uso: Permite instalar un manejador de señales. Es más robusta y flexible que `signal()`.
  En el ejemplo: Se utiliza para definir el manejador de `SIGUSR1`, que recibirá la señal y procesará el token enviado.

- `sigprocmask()`
  Uso: Permite bloquear o desbloquear señales, modificando la máscara de señales del proceso.
  En el ejemplo: Se bloquea `SIGUSR1` antes de crear el proceso hijo, para evitar que la señal se entregue prematuramente.

- `sigsuspend()`
  Uso: Cambia temporalmente la máscara de señales y suspende la ejecución del proceso hasta que se reciba una señal.
  En el ejemplo: Se utiliza en el proceso hijo para esperar de forma segura la señal sin riesgo de perderla.

- `sigqueue()`
  Uso: Envía una señal a un proceso, permitiendo adjuntar información adicional (un entero, en este caso) mediante la unión `sigval`.
  En el ejemplo: El proceso padre utiliza `sigqueue()` para enviar `SIGUSR1` al hijo junto con el token.

- `wait()`
  Uso: Permite al proceso padre esperar a que los procesos hijos terminen su ejecución.
  En el ejemplo: Se utiliza para esperar la terminación del proceso hijo después de que reciba la señal.

3 Descripción del Desafío

El desafío del laboratorio consiste en implementar un programa en C donde:

1. El proceso principal crea un conjunto de procesos hijos organizados en forma de anillo, donde se especificará la cantidad de hijos a crear por medio de la línea de comando con la opción `-p`.

2. Se inicializa un token (un número entero) con un valor especificado desde la línea de comandos mediante la opción `-t`.

3. El primer proceso en el anillo inicia el desafío decrementando el token en un valor aleatorio (obtenido en el rango de 0 a $M - 1$, donde M se pasa como argumento con la opción `-M`).

4. Cada proceso, al recibir el token, realiza lo siguiente:

   (a) Resta un valor aleatorio (entre 0 y $M - 1$) al token.

(b) Si el token resultante es mayor o igual a cero, lo envía al siguiente proceso activo mediante una señal.

(c) Si el token resultante es negativo, el proceso se considera eliminado y notifica a los demás procesos mediante señales su salida.

5. Al eliminarse un proceso, se activa un mecanismo de elección de líder entre los procesos restantes. El líder (por ejemplo, el proceso con el mayor identificador) reinicializa el token y reanuda el desafío.

6. El desafío continúa hasta que solo queda un proceso, el cual es declarado ganador.

4 Ejemplo Detallado del Envío del Token y Manejo de Señales

A continuación se presenta un ejemplo en C que ilustra cómo utilizar las funciones anteriormente descritas para enviar un token a través de señales de forma segura. Se utiliza `sigqueue()` para enviar la señal y `sigsuspend()` para esperar de forma segura la señal en el proceso hijo.

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>
#include <sys/wait.h>

volatile sig_atomic_t token_recibido = 0;

/*
* Manejador de señales que procesa la señal SIGUSR1.
* Se utiliza SA_SIGINFO para acceder a la información adicional enviada con la señal.
*/
void manejador(int sig, siginfo_t *si, void *context) {
        // Extrae el token enviado a través del campo sival_int de la unión sigval.
        int token = si->si_value.sival_int;
        printf("Proceso %d recibió el token: %d\n", getpid(), token);
        token_recibido = 1;  // Indica que la señal ha sido recibida.
}

int main() {
        struct sigaction sa;
        sigset_t mask, oldmask;

        // Configuración del manejador para SIGUSR1.
        sa.sa_flags = SA_SIGINFO;          // Permite acceder a información extendida.
        sa.sa_sigaction = manejador;        // Asigna el manejador definido.
```

```
27          sigemptyset(&sa.sa_mask);
28          if(sigaction(SIGUSR1, &sa, NULL) == -1) {
29                  perror("sigaction");
30                  exit(EXIT_FAILURE);
31          }
32
33          // Bloquea SIGUSR1 para evitar su entrega prematura.
34          sigemptyset(&mask);
35          sigaddset(&mask, SIGUSR1);
36          if(sigprocmask(SIG_BLOCK, &mask, &oldmask) < 0) {
37                  perror("sigprocmask");
38                  exit(EXIT_FAILURE);
39          }
40
41          pid_t pid = fork();
42          if(pid == -1) {
43                  perror("fork");
44                  exit(EXIT_FAILURE);
45          } else if(pid == 0) {
46                  // Proceso hijo: espera la señal de manera segura usando
                       sigsuspend.
47                  while(!token_recibido) {
48                          // sigsuspend reemplaza temporalmente la máscara
                               actual por 'oldmask',
49                          // permitiendo la recepción de SIGUSR1.
50                          sigsuspend(&oldmask);
51                  }
52                  exit(0);
53          } else {
54                  // Proceso padre: prepara y envía la señal con el token.
55                  union sigval value;
56                  value.sival_int = 42;  // Asigna el token (valor de ejemplo
                       ).
57
58                  if(sigqueue(pid, SIGUSR1, value) == -1) {
59                          perror("sigqueue");
60                          exit(EXIT_FAILURE);
61                  }
62                  wait(NULL);  // Espera a que el proceso hijo termine.
63          }
64          return 0;
65  }
```

Listing 1: Ejemplo de envío y recepción de señales usando `sigsuspend()`

En este ejemplo:

- Se configura un manejador para `SIGUSR1` mediante `sigaction()` que utiliza la bandera `SA_SIGINFO` para acceder al token enviado.

- Se bloquea `SIGUSR1` con `sigprocmask()` antes de llamar a `fork()` para evitar que el hijo reciba la señal antes de estar listo.
- En el proceso hijo se utiliza un bucle junto con `sigsuspend()` para esperar la señal de manera segura; la función reemplaza temporalmente la máscara de señales por `oldmask` (donde `SIGUSR1` no está bloqueada).
- El proceso padre envía la señal utilizando `sigqueue()`, adjuntando el token en el campo `sival_int` de la unión `sigval`.

5 Restricciones y Consideraciones

- **Comunicación vía señales:** La transmisión del token y la notificación de eventos (como la eliminación de un proceso) deben realizarse exclusivamente mediante señales. No se permite el uso de otros mecanismos de IPC (por ejemplo, pipes o memoria compartida).
- **Generación de números aleatorios:** Se puede utilizar la función `rand()` de `stdlib.h` para generar números, asegurándose de inicializar la semilla con `srand()`.
- **Sincronización:** Es fundamental prestar atención a la sincronización de los procesos y al manejo de condiciones de carrera al recibir señales.

6 Ejemplo de Ejecución

Un posible flujo de ejecución podría ser el siguiente:

```
$ ./desafio1 -t 10 -M 10 -p 4
Proceso 0 ; Token recibido: 10 ; Token resultante: 8
Proceso 1 ; Token recibido: 8 ; Token resultante: 5
Proceso 2 ; Token recibido: 5 ; Token resultante: 2
Proceso 3 ; Token recibido: 2 ; Token resultante: -1 (Proceso 3 es
   eliminado)
Proceso 0 ; Token recibido: 10 ; Token resultante: 9
Proceso 1 ; Token recibido: 9 ; Token resultante: 4
Proceso 2 ; Token recibido: 4 ; Token resultante: -2 (Proceso 2 es
   eliminado)
Proceso 0 ; Token recibido: 10 ; Token resultante: 7
Proceso 1 ; Token recibido: 7 ; Token resultante: 1
Proceso 0 ; Token recibido: 1 ; Token resultante: -3 (Proceso 0 es
   eliminado)
Proceso 1 es el ganador
```

Listing 2: Ejemplo de ejecución del programa

7 Entregables

El laboratorio es en parejas. Si se elige una pareja, ésta no podrá ser cambiada durante el semestre. Se descontará 1 punto (de nota) por día de atraso con un máximo de tres días, a contar del cuarto se evaluará con nota mínima. Debe subir en un archivo comprimido ZIP (una carpeta) a USACH virtual con los siguientes entregables:

Comprima y entregue los siguientes archivos:

1. `Makefile`: Archivo para compilar el programa.
2. `desafio1.c`: Archivo principal en C con el código fuente (puede dividirse en varios archivos si es necesario).
3. `README.txt`: Instrucciones para compilar y ejecutar el programa.
4. Trabajos con códigos que hayan sido copiados de un trabajo de otro grupo serán calificados con la nota mínima.

Recuerde que todas las funciones deben estar comentadas, explicadas de forma entendible especificando sus entradas, funcionamiento y salida. Si una función no está explicada se bajará puntaje. Se deben comentar todas las funciones de la siguiente forma:

```
// Entradas: explicar qué se recibe
// Salidas: explicar qué se retorna
// Descripción: explicar qué hace
```

- El archivo comprimido (al igual que la carpeta) debe llamarse:

  RUTESTUDIANTE1_RUTESTUDIANTE2.zip
  Ejemplo 1: 19689333k_186593220.zip

- **NOTA 1:** El archivo debe ser subido a uvirtual en el apartado "Entrega Lab1".
- **NOTA 2:** Cualquier diferencia en el formato del laboratorio que es entregado en este documento, significará un descuento de puntos.
- **NOTA 3:** SOLO UN ESTUDIANTE DEBE SUBIR EL LABORATORIO.
- **NOTA 4:** En caso de solicitar corrección del laboratorio, esta será en los computadores del Diinf, es decir, si funciona en esos computadores no hay problema.
- **NOTA 5:** Cualquier comprimido que no siga el ejemplo 1, significará un descuento de 1 punto de nota.

Fecha de entrega: 17-04-2025, antes de las 23:59 hrs.