Programación Concurrente y Distribuida

Práctica 1 Curso 2021/22 3º Curso

Sincronización mediante señales

Introducción

Las señales son un mecanismo de sincronización basado en interrupciones software. En Linux cada señal se identifica por un número entero mayor que cero, y cada proceso puede decidir qué señales desea recibir, proporcionando una rutina de servicio de atención para cada una de ellas, si bien, existen algunas señales cuya atención es manejada directamente por el sistema operativo y no se pueden enmascarar (por ejemplo `SIGKILL`). Además, existe una rutina de atención por defecto para cada señal que puede cambiarse en tiempo de ejecución, bien sea por otra rutina proporcionada por el programador, o simplemente por una orden que conlleva ignorar la señal correspondiente.

Las señales se pueden utilizar simplemente para terminar procesos (`SIGTERM` o `SIGKILL`) o para realizar procesamientos síncronos dependientes de la aplicación (`SIGUSR1, SIGUSR2`). Por su parte, el sistema operativo utiliza señales para notificar a los procesos determinadas situaciones. Por ejemplo, las señales `SIGBUS` (*bus error*), `SIGSEV` (*segmentation violation*) y `SIGPFE` (*floating point exception*) son enviadas por el sistema operativo a un proceso cuando éste realiza una operación no permitida. La acción por defecto para estas señales consiste en terminar el proceso y generar un fichero *core.* También se utilizan señales para notificar a los procesos acciones de los usuarios (`SIGINT` se envía al proceso cuando el usuario teclea Ctrl+C en el terminal).

Objetivos

Esta práctica tiene como objetivos fundamentales:

- Iniciarse en la programación concurrente.
- Utilizar un mecanismo de sincronización básico como son las señales.

Entorno

Los ordenadores del laboratorio ya están preparados para realizar las prácticas. Si el alumno desea realizar las prácticas en su ordenador deberá tener instalado Linux (en el laboratorio está instalada la distribución OpenSuse 12.3 y es sobre la que se deberá garantizar el correcto funcionamiento de las prácticas).

Enviando Señales

El conjunto de señales definidas en Linux puede consultarse en el fichero de cabecera */usr/include/bits/signum.h*, incluido en <*signal.h*>.

Para enviar una señal a un proceso puede utilizarse `kill` desde la línea de comandos, o la función `kill()` incluida en <*signal.h*>.

Un proceso puede enviarse una señal a sí mismo mediante la función `raise()` o programar el envío de la señal `SIGALRM` dentro de `t` segundos mediante la función `alarm()`.

Ejercicio 1 (mini ejemplos):

Consulte el *man* del comando `kill`.

Escriba un proceso `p1` que espere a recibir una señal utilizando la función `pause()` (consulte el *man* de dicha función). Envíe a dicho proceso, utilizando el comando `kill`, cada una de las señales definidas y compruebe la acción por defecto de cada una de ellas.

Ejercicio 2 (mini ejemplos):

Consulte el *man* de la función `kill()`.

Repita el Ejercicio 1 utilizando la función `kill()` invocada desde un segundo proceso `p2` que reciba como parámetros el número de la señal a enviar y el `pid` del proceso al que se debe enviar la señal.

Capturando Señales

La principal función de manejo de señales es `sigaction()`, incluida también en *<signal.h>*. Mediante dicha función es posible consultar y asignar la rutina de servicio de atención de una señal dada, establecer el conjunto de señales que deben bloquearse, y establecer una serie de *flags* que modifican el comportamiento del manejo de dicha señal.

Ejercicio 3 (mini ejemplos):

Consulte el *man* para obtener la sintaxis y el tipo de parámetros utilizado por la función `sigaction()` (la función `signal()` está obsoleta y es sustituida por `sigaction()`).

Modifique el proceso `p1` del Ejercicio 1 para que capture y procese todas las señales definidas mediante la utilización de la función `sigaction()`. El proceso deberá finalizar al recibir la señal `SIGTERM` e indicar con un vector de *bits* si ha recibido o no cada una de las señales.

Ejercicio 4 (mini ejemplos):

Modifique el proceso `p1` del Ejercicio 1 para que capture únicamente las señales `SIGUSR1` y `SIGUSR2`. El proceso deberá finalizar al recibir la señal `SIGTERM` e imprimir el número de señales de cada tipo recibidas.

Escriba dos versiones de dicho programa utilizando los flags `SA_RESETHAND` y `SA_RESTART` respectivamente y observe las diferencias entre ambos. Compruebe, añadiendo las modificaciones necesarias, el funcionamiento del campo `sa_mask` del tipo de datos `struct sigaction`.

Ejercicio 5:

Implemente un programa que simule el funcionamiento de un `ascensor` básico. Dicho ascensor puede recibir dos órdenes diferentes: i) subir un piso; ii) bajar un piso. Las órdenes se enviarán al proceso ascensor mediante señales en la línea de comandos. Se utilizará la señal `SIGUSR1` para subir y la señal `SIGUSR2` para bajar. El `ascensor` no podrá bajar si se encuentra en el primer piso, y no podrá subir si se encuentra en el último piso (que recibirá como parámetro en la línea de comandos).

El proceso `ascensor` se suspenderá a la espera de una orden. Cuando ésta se produzca, indicará mediante un mensaje el tipo de orden que va a atender. Tras finalizar el movimiento correspondiente a la orden recibida, que tardará un tiempo constante de `T_PISO` segundos, volverá a esperar una nueva orden. El proceso `ascensor` debería poder realizar otras tareas mientras está subiendo o bajando un piso.

El programa finalizará cuando reciba la señal `SIGQUIT`, mostrando el número de veces que ascendió y descendió.

Ejercicio 6:

Implemente un proceso `pulsador` que sea capaz de enviar las órdenes correspondientes al proceso `ascensor` del Ejercicio 5. Dicho proceso `pulsador` mostrará un menú donde el usuario podrá elegir entre la órdenes de `SUBIR, BAJAR o SALIR`, equivalentes al envío de las señales `SIGUSR1, SIGUSR2` y `SIGQUIT`, respectivamente.

Ejercicio 7:

Implemente un proceso `sensor` que simule el funcionamiento de los sensores que indican al `ascensor` que han llegado a una nueva planta. El proceso `sensor` mostrará un mensaje cada vez que se active y recibirá como parámetro en la línea de comandos el piso al que pertenece.

Modifique el proceso `ascensor` del Ejercicio 5 para que utilice los nuevos procesos `sensores`.

Resumen

Los principales resultados esperados de esta práctica son:

- Iniciarse en el hábito de la programación concurrente de procesos.
- Ser capaces de sincronizar procesos concurrentes utilizando señales.

Como trabajo adicional del alumno, se proponen las siguientes líneas:

- Profundizar en el manejo de señales: sigprocmask, sigpending, sigsuspend.
- Implementar grafos de precedencia utilizando señales.