Programación Concurrente y Distribuida

Práctica 2 Curso 2016/17 3º Curso

Grafos de Precedencia

Introducción

Los grafos de precedencia son grafos sin ciclos donde cada nodo representa una única sentencia o un conjunto de instrucciones que deben ejecutarse secuencialmente. Expresan la concurrencia permitida entre el conjunto de sentencias de sus nodos estableciendo relaciones de precedencia entre ellas. Es decir, cada transición del grafo expresa una restricción a la concurrencia indicando que una sentencia o grupo de sentencias debe preceder a la otra.

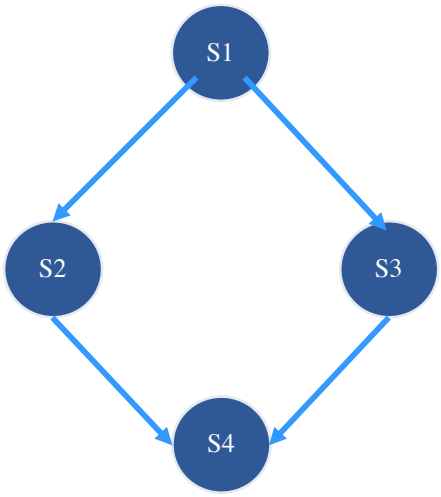

Figura 1.

Por ejemplo, según el grafo de precedencia de la Figura 1 son posibles las siguientes trazas de ejecución:

i) `S1; S2; S3; S4;` y ii) `S1; S3; S2; S4;`

Objetivos

Esta práctica tiene como objetivos fundamentales:

- Familiarizarse con la creación de procesos concurrentes utilizando las llamadas al sistema *fork, wait* y *execl*.
- Comprender la concurrencia expresada por un grafo de precedencia.
- Utilizar los mecanismos de sincronización derivados de la jerarquía de procesos.

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).

Sincronización de procesos hijo

La estructura jerárquica de procesos de Linux permite sincronizar un proceso con la finalización de sus procesos hijo. Por una parte, el sistema operativo envía la señal `SIGCHLD` a un proceso cuando uno de sus procesos hijo finaliza. Además, mediante la función `wait()` un proceso puede suspenderse hasta que finalice alguno de sus procesos hijo, o incluso suspenderse hasta que finalice un proceso hijo en concreto utilizando la función `waitpid()`. Por su parte, los procesos hijo permanecen en un estado *zombie* desde que finalizan hasta que su proceso padre "captura" su finalización mediante `wait()` o `waitpid()`.

Ejercicio 1 (trabajo previo a la sesión de Laboratorio):

Consulte el *man* de `fork()`.

Escriba un proceso `p1` que cree tres procesos hijo y espere en un bucle a recibir una señal utilizando la función `pause()`. Incluya un `sleep` en los procesos hijo para que finalicen unos segundos después de ser creados. El proceso padre deberá imprimir un mensaje en pantalla cada vez que finalice un proceso hijo, y finalizar cuando lo hayan hecho todos sus hijos.

Ejercicio 2 (trabajo previo a la sesión de Laboratorio):

Consulte el *man* de `wait() y exit()`.

Repita el Ejercicio 1 utilizando la función `wait()` en vez de `pause()` y sin capturar la señal `SIGCHLD`. Incluya la función `exit()` para finalizar cada proceso hijo. Cada vez que finalice un proceso hijo, el proceso padre deberá imprimir en pantalla el `pid` del proceso finalizado y el código de finalización utilizado en el `exit()` por dicho proceso hijo.

Ejercicio 3 (trabajo previo a la sesión de Laboratorio):

Consulte el *man* de `waitpid()` y del comando `ps`.

Repita el Ejercicio 2 utilizando la función `waitpid()`[1] en vez de `wait()`. Compruebe, utilizando el comando `ps`, cómo aparecen procesos *zombie* que no aparecían en el Ejercicio 2. Razone acerca de las ventajas/inconvenientes de utilizar `waitpid()` en vez de `wait()`.

[1] Utilice la función `waitpid()` para que se suspenda a la espera de un proceso concreto. Es decir, con el parámetro `pid > 0`.

Ejercicio 4 (trabajo previo a la sesión de Laboratorio):

Consulte el *man* de `execl()`.

Repita el Ejercicio 2 escribiendo un nuevo programa `p2` para los procesos hijo que imprima en pantalla la lista de parámetros que reciba en la línea de comandos. Utilice `execl()` para invocar a dicho programa desde el código de los procesos hijo.

Ejercicio 5:

Escriba un único programa `p3` que implemente el grafo de precedencia de la Figura 2 utilizando las llamadas al sistema `fork` y `wait`[2]. El grupo de sentencias a ejecutar en cada nodo del grafo se simularán mediante la sentencia `printf("str")`, donde `"str''` se corresponde con la cadena de caracteres que contiene cada nodo de la Figura 2[3]. Una traza de ejecución correcta sería:

```
> p3
De Gea Ramos Carvajal Piqué Jordi-Alba Busquets Nolito Iniesta Isco Silva Aspas
```

Compruebe que, aun implementando correctamente la precedencia expresada por el grafo, los resultados no son los esperados: i) aparecen *jugadores* repetidos; ii) aparecen con una *estrategia* no permitida por el grafo. Realice los experimentos necesarios para obtener conclusiones acerca de qué sentencias se repiten y el orden en el que parecen ser ejecutadas.

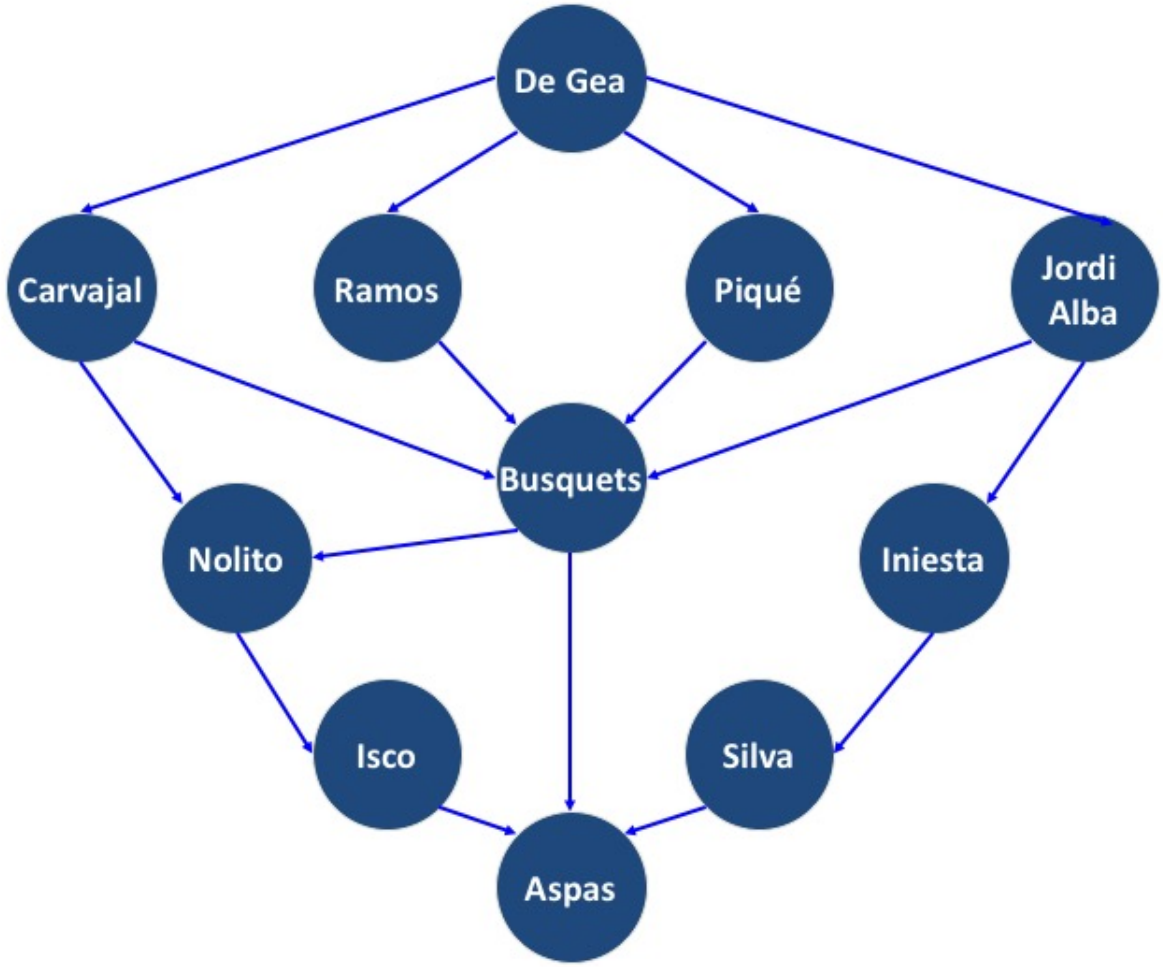

Figura 2.

[2] No se puede utilizar `waitpid`.

[3] Cada *alineación* deberá aparecer en **una única línea**. Es decir, sólo se deberá incluir un `"\n"` en el `printf` correspondiente a la última sentencia del grafo `("Aspas\n")`.

Ejercicio 6:

Repita el Ejercicio 5 utilizando `execl` y el programa `p2` del Ejercicio 4 para los procesos hijo.

Compruebe que, aun implementando correctamente la precedencia expresada por el grafo, los resultados siguen sin ser los esperados. En este caso, no aparecen *jugadores* repetidos pero sí con *estrategias* no permitidas por el grafo. Realice los experimentos necesarios para obtener conclusiones acerca del orden en el que parecen ser ejecutadas las sentencias del grafo.

Resumen

Los principales resultados esperados de esta práctica son:

- Aprender a sincronizar procesos utilizando la sincronización entre procesos padre e hijos.
- Aprender a implementar grafos de precedencia.
- Experimentar y profundizar en aspectos relacionados con la creación de procesos que pueden ocasionar resultados aparentemente erróneos.

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

- Resolver los problemas de sincronización planteados en las dos primeras prácticas combinando las dos herramientas vistas: *señales* y sincronización mediante *fork* y *wait*.