Procesos e hilos: el *downloader*

Departamento de Automática

Universidad de Alcalá

/gso>

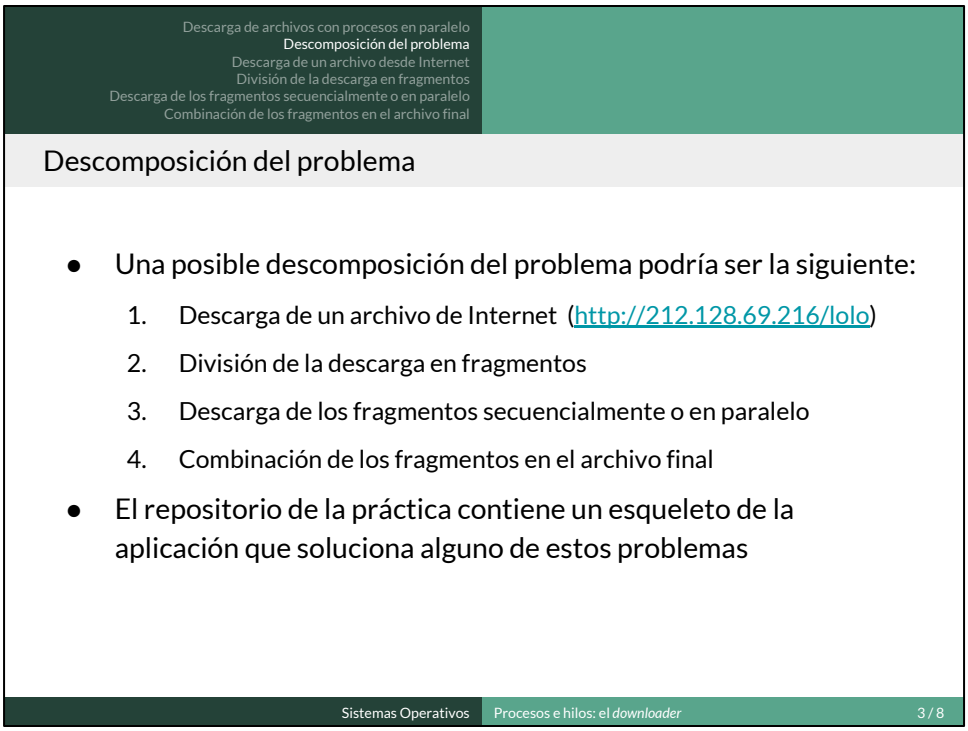
Descarga de archivos con procesos en paralelo

- Objetivo principal: descarga de un archivo de gran tamaño de forma eficiente desde Internet
- Otros objetivos:
  - Utilizar la biblioteca POSIX para crear procesos
  - Aplicar primitivas de sincronización básicas entre procesos

Especificación de la práctica

Implementar un programa que descargue el archivo localizado en http://212.128.69.216/lolo (tamaño: 1GiB aprox.) dividiendo la descarga en fragmentos (*chunks*) que serán descargados de forma secuencial o en paralelo y después combinados en un único archivo

La selección del modo de descarga (secuencial o paralelo) se realizará mediante argumentos introducidos al programa mediante línea de órdenes.



¡ATENCIÓN! Es necesario modificar el valor de la macro `TARGET_URL` definida en la línea 6 del archivo `downloader/downloader.c` por esta otro:

```
#define TARGET_URL "http://212.128.69.216/lolo"
```

En un primer paso, hay que ejecutar los comandos de los recuadros en la línea línea de órdenes. Los modificadores que se emplean en las órdenes tienen el siguiente significado:

- **`-O`**: escribe el contenido de la descarga en un archivo con el mismo nombre que el del recurso remoto
- **`-o [nombre]`**: escribe el contenido de la descarga en un archivo con el nombre especificado
- **`-s`**: modo silencioso. No muestra por pantalla el progreso de la descarga ni los mensajes de error
- **`-H [cabecera]`**: incluye el texto "cabecera" como extra en la cabecera HTTP de la petición

Una vez comprobado que la descarga funciona correctamente, hay que completar un programa de ejemplo que llame a `exec()` para que los alumnos comprueben el funcionamiento de la llamada al sistema. Los fundamentos de la llamada al sistema ya se han visto en clase de teoría. Del conjunto de llamadas al sistema `exec()`, la más sencilla de utilizar es `execlp()`. La función recibe como parámetros la el nombre del programa a ejecutar y una lista con los argumentos terminada en `NULL`:

```
int execlp(const char *file, const char *arg, ..., NULL);
```

¡ATENCIÓN!: el primer argumento siempre que tiene que ser **el propio nombre del programa**, por ejemplo:

```
execlp(“ls”, “ls”, “-l”, NULL);
```

El primer programa que habría que realizar es uno que ejecute directamente la orden `execlp()` para llamar a curl. Los programas no tienen por qué almacenarse en el repositorio, pero los alumnos pueden hacerlo si así lo desean. Para que el compilador encuentre declarada la función `execlp()` es necesario incluir el siguiente archivo de cabecera:

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

Los alumnos podrán comprobar cómo no se ejecuta ninguna línea a continuación de la llamada a `execlp()`, ya que, tras la llamada al sistema, el proceso deja de ejecutar el programa original y pasa a ejecutar el programa indicado en los parámetros de la función.

A continuación, los alumnos pueden incluir una llamada al sistema `fork()` para crear un proceso hijo y que sea el proceso hijo el que ejecute el programa. De esta forma, el proceso padre puede continuar su ejecución después de crear el proceso hijo.

El proceso padre puede esperar a que el hijo finalice su ejecución con la llamada al sistema `wait()`. Para poder utilizarla, es necesario incluir los siguientes archivos de cabecera:

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

El uso de la llamada al sistema es el siguiente:

```
int status;
…
wait(&status);
```

En la variable `status` quedará almacenado el valor con el que el hijo ha terminado. Por convención, si el estado de terminación es cero, se considera que el proceso hijo ha terminado correctamente. Cualquier valor distinto de cero se considera un error.

División de la descarga en fragmentos

- Se parte de los siguientes datos:
  - El tamaño total del archivo
  - El número de fragmentos
- Por simplicidad, el tamaño del archivo es fijo y conocido por adelantado[1]
  - Su valor es de **1047491658** bytes
- El número de fragmentos se pasa como argumento desde la línea de órdenes
  - Por ejemplo, si la descarga se tiene que dividir en 10 fragmentos, hay que llamar al programa de la siguiente forma:

    ```
    $ ./downloader 10
    ```
  - Si la división no es exacta, el último fragmento contendrá el resto de bytes
- Es necesario obtener los argumentos de línea de órdenes:
  - **Solución**: comprobar cómo se obtienen los argumentos en C (ver el código esqueleto)
- Es necesario definir un algoritmo para obtener el comienzo y el final de cada fragmento
  - **Solución**: diseñar una expresión que calcule el comienzo y el final de cada fragmento a partir del tamaño de fragmento, el número de fragmentos y el fragmento actual (ver el código esqueleto)

(1) El tamaño del archivo se puede conocer empleando el mismo programa curl. curl permite obtener únicamente las cabeceras HTTP de la transferencia sin llegar a transferir el archivo completo. En la cabecera HTTP aparece el tamaño del contenido a transferir, esto es, del archivo. El campo de la cabecera correspondiente al tamaño es “Content-Length” y su valor se puede obtener directamente aplicando el siguiente filtro:

```
$ curl -sI http://212.128.69.216/lolo | grep Content-Length | tr -d '
' | cut -d: -f2
```

Descarga de los fragmentos secuencialmente o en paralelo

- El usuario puede escoger si desea realizar una descarga de los fragmentos secuencial o en paralelo
- Para especificar el modo de descarga, se emplea un segundo argumento de línea de órdenes (ver el código esqueleto):
  - **`S`**: opción de descarga secuencial

    `$ ./downloader 10 S`
  - **`P`**: opción descarga en paralelo

    `$ ./downloader 10 P`
- Para facilitar la llamada a `exec()`, se proporciona una función específica

Función de descarga de un fragmento

```
void download_fragment(char* url, long from, long to, char* outfile)
```

- El comportamiento del programa padre depende del modo de descarga

Descarga de los fragmentos secuencialmente o en paralelo (II)

- Pasos para una descarga secuencial:
  1. Hacer `fork()` de un proceso hijo
  2. Hacer que el proceso hijo ejecute curl para descargar un fragmento, proporcionándole el inicio, el final y el nombre del archivo parcial
  3. El proceso padre debe esperar a que el hijo finalice la descarga
  4. Repetir pasos 1 a 3 hasta que se hayan descargado todos los fragmentos
- Pasos para una descarga en paralelo:
  1. Hacer `fork()` de un proceso hijo
  2. Hacer que el proceso hijo ejecute curl para descargar un fragmento, proporcionándole el inicio, el final y el nombre del archivo parcial
  3. Repetir pasos 1 a 3 hasta que se hayan descargado todos los fragmentos
  4. El proceso padre debe esperar a que todos los hijos finalicen las descargas, por ejemplo empleando un bucle while o for

Hay una indicación en el archivo esqueleto sobre cómo generar el nombre del archivo parcial. Es necesario declarar el vector de caracteres donde se va a guardar la cadena con el nombre del archivo.

Combinación de los fragmentos en el archivo final

- Esta parte no es obligatoria
- La combinación de archivos se puede conseguir fácilmente empleando la redirección de la salida estándar del intérprete de órdenes
- Por ejemplo, para concatenar cinco archivos parciales con nombres: “download-1”, “download-2”, “download-3”, “download-4” y “download-5” podemos emplear la orden:

```
$ cat download-* > complete_file
```

- El intérprete de órdenes expandirá el carácter ‘*’ y lo sustituirá por la lista de archivos del directorio actual que comiencen por “download-”[1]
- Se puede comprobar si el archivo final es el mismo que el original comprobando su *hash* MD5:

```
$ md5sum < complete_file
```

- El *hash* del archivo es “e4578889e6daf247d59a1a90f9113ce9”

(1) A esto se le llama “*file globbing*”. La operación la realiza el intérprete de órdenes antes de ejecutar el programa destino. No funciona directamente con la llamada al sistema `exec()`.