Concurrencia y Paralelismo

GEI 2024

Práctica #2 (Compresión Concurrente)

En esta práctica vamos a implementar un compresor/descompresor concurrente de ficheros.El código proporcionado implementa un compresor/descompresor no multithread. El compresor genera un archivo formado por varios fragmentos numerados (denominados **chunks**). El descompresor lee estos ficheros, y genera de nuevo el fichero original.

La compresión/descompresión se realiza utilizando la librería zlib. Para poder compilar código que la use es necesario tener instalados los archivos de cabecera de la librería. Instalelos con la herramienta de paquetes correspondiente. Por ejemplo, en ubuntu:

```
$ sudo apt-get install zlib1g-dev
```

El código proporcionado consta de los siguientes módulos:

- **compress**, que hace de interfaz con la librería zlib para comprimir buffers en memoria.
- **chunk_archive**, que permite guardar y recuperar archivos formados por bloques arbitrarios de datos especificados mediante la estructura **chunk**.
- **queue**, que implementa una cola que se utiliza para guardar los chunks según se leen de la entrada, y para recuperar la salida del proceso aplicado.
- **options**, que procesa las opciones de linea de comandos.
- **comp**, que une los distintos módulos para implementar el compresor/descompresor.

Se pide:

Ejercicio 1 (**Cambiar la cola para que sea thread-safe**) La cola proporcionada no es thread-safe (dos threads usando la misma cola a la vez podrían provocar un fallo). Cambie la implementación de la cola para que los accesos concurrentes no provoquen problemas. Considere los casos de que la cola se llene o se vacíe aplicando el patrón productores/consumidores. Pruebe este apartado haciendo un pequeño programa multithread de ejemplo que inserte y elimine elementos de forma concurrente.

Ejercicio 2 (**Cambiar la implementación de comp para que la compresión se haga simultaneamente en varios threads**) El código actual procesa los chunks de la cola de entrada de forma secuencial. Modifíquelo para que **opt.num_threads** threads cojan chunks de la cola de entrada y los procesen para la cola de salida de forma concurrente.

La implementación de **chunk_archive** no necesita que los chunks se añadan en orden al archivo comprimido, por lo que no es necesario preocuparse del orden en que los threads añaden chunks al archivo.

Ejercicio 3 (**Cambiar la lectura/escritura de chunks para que sea concurrente**) En la implementación actual se leen todos los chunks, después se procesan, y por último se escriben en el fichero destino. Como la cola tiene tamaño finito, esto pone un límite al tamaño máximo de los ficheros que se pueden comprimir (**q-¿size*chunk_size**). Cambie la implementación para que la lectura y escritura de chunks se haga en dos threads independientes.

Entrega

La fecha límite de entrega es el 3 de marzo. El código inicial está disponible a través de github classroom en https://classroom.github.com/a/RvDu8CFb.