Arquitectura de Computadores

GRADO EN INGENIERÍA INFORMÁTICA

Laboratorio 3: programación concurrente y consistencia de memoria

Curso 2020/2021

Jorge Rodríguez Fraile, 100405951, Grupo 81, 100405951@alumnos.uc3m.es

Índice

Jorge Rodríguez Fraile, 100405951

Estudio del código fuente

Búfer secuencial

Estudie la implementación del búfer secuencial (archivo de cabecera seqbuffer.h) y considere las siguientes cuestiones:

1. ¿Qué funciones de seq_buffer pueden lanzar excepciones?

   Seq_buffer<T>::put: En la 75 de full_buffer.

   Seq_buffer<T>::get: En la 88 de empty_buffer.

   Las funciones de next_position, size, empty y full

2. ¿Puede el constructor de seq_buffer lanzar alguna excepción? ¿Cuál?

   Si, puede saltar excepción si no se pasa el parámetro esperado y también la función size_{n}.

3. ¿Para qué sirve el dato miembro next_read_ de seq_buffer?

   Para leer la siguiente posición del buffer.

4. ¿Para qué sirve el dato miembro seq_write _ de seq_buffer?

   Para escribir la siguiente posición en el buffer.

5. ¿Cuál es el número máximo de elementos que puede almacenarse en un seq_buffer creado con size_ == 100?

   El número máximo de elementos del buffer puede almacenar será 100 posiciones.

6. ¿Qué ocurre si se hace un put() sobre un seq_buffer que está lleno?

   Salta la excepción full_buffer().

7. ¿Qué ocurre si se hace un put() sobre un seq_buffer que está vacío?

   Mete el elemento en la posición que se haya quedado el buffer.

8. ¿Qué ocurre si se hace un get() sobre un seq_buffer que está lleno?

   Saca el elemento del principio del buffer.

9. ¿Qué ocurre si se hace un get() sobre un seq_buffer que está vacío?

   Salta la excepción de empty_buffer()

Búfer con cerrojos

Estudie la implementación del búfer con cerrojos (archivo de cabecera lockedbuffer.h) y considere las siguientes cuestiones:

1. ¿Qué funciones de locked_buffer pueden lanzar excepciones?

   No lanza excepciones ya que se usan mutex y variables condición cuando se llena y cuando esta vacío.

   Las funciones size(), empty, full, next_position, is_empty, is_full.

2. ¿Puede el constructor de locked_buffer lanzar alguna excepción? ¿Cuál?

   Si, puede saltar excepción si no se pasa el parámetro esperado y también la función size_{n}.

   No sabemos

3. ¿Puede la función miembro put() lanzar una excepción? ¿Cuál?

   No lanza excepción se bloqueada cuando se esta lleno y no deja guardar el dato, se queda esperando a que se libere. Usa variable condiciones y mutex.

4. ¿Puede la función miembro get() lanzar una excepción? ¿Cuál?

   No lanza excepción se bloqueada cuando se está vacío y no deja guardar el dato, se queda esperando a que se libere. Usa variable condiciones y mutex.

5. ¿Qué diferencia hay entre full() e is_full()?

   Full comprueba si está llena con cerrojo y is_full comprueba si está llena, pero sin cerrojo.

6. ¿Qué diferencia hay entre empty() e is_empty()?

   empty comprueba si está vacía con cerrojo y is_empty comprueba si está vacía, pero sin cerrojo

7. ¿Cuál es el número máximo de elementos que puede almacenarse en un locked_buffer creado con size_ == 100?

   Caben 100 elementos.

8. ¿Qué ocurre si se hace un put() sobre un locked_buffer que está lleno?

   Se pone en espera el cerrojo de escribir y se libera el de lectura.

9. ¿Qué ocurre si se hace un put() sobre un locked_buffer que está vacío?

   Se introduce el elemento en el buffer.

10. ¿Qué ocurre si se hace un get() sobre un locked_buffer que está lleno?

    Coge el elemento del buffer y libera el cerrojo de full().

11. ¿Qué ocurre si se hace un get() sobre un locked_buffer que está vacío?

    Se pone en espera el cerrojo de leer y se libera el cerrojo de escritura esperando a que se metan nuevos elementos.

12. Investigue el efecto de la palabra reservada mutable. Si se eliminase la calificación de mutable sobre el dato miembro mut_ Qué funciones miembro habría que modificar? ¿Cómo?

    Mutable puede modificar una variable const.

    Las funciones full y empty para que no usen variables const, par apode modificar las variable en esta función.

    Quitando el const de esas funciones.

13. ¿Por qué no es necesario marcar como mutable a los datos miembro not_full_ y not_empty_?

    Porque están bajo la condición del mutex mut_.

    Por que not_full se libera y bloquea fuera de una función const, mientras que las otras se modifican dentro de este tipo de funciones.

Búfer libre de cerrojos

Estudie la implementación del búfer libre de cerrojos (archivo de cabecera atomicbuffer.h) y considere las siguientes cuestiones:

1. ¿Qué funciones de atomic_buffer pueden lanzar excepciones?

   Función size, next_position, empty y full.

   No lanza excepciones usa tipos atómicos.

2. ¿Puede el constructor de atomic_buffer lanzar alguna excepción? ¿Cuál?

   El size puede lanzar excepciones y la excepción será que el buffer está lleno.

   No lanza excepciones

3. ¿Puede la función miembro put() lanzar una excepción? ¿Cuál?

   No, usa los tipos atómicos para evitar esta clase de problemas.

4. ¿Puede la función miembro get() lanzar una excepción? ¿Cuál?

   No, usa los tipos atómicos para evitar esta clase de problemas.

5. ¿Cuál es el número máximo de elementos que puede almacenarse en un atomic_buffer creado con size_ == 100?

   El número máximo de elementos del buffer puede almacenar será 100 posiciones.

6. ¿Qué ocurre si se hace un put() sobre un atomic_buffer que está lleno?

   Espera a que cambie la variable atómica y mete el valor cuando se libera.

7. ¿Qué ocurre si se hace un put() sobre un atomic_buffer que está vacío?

   Se mete el elemento.

8. ¿Qué ocurre si se hace un get() sobre un atomic_buffer que está lleno?

   Se saca el elemento.

9. ¿Qué ocurre si se hace un get() sobre un atomic_buffer que está vacío?

   Espera a que cambie el tipo atómico, cuando se mete uno, y después lo puede meter.

10. Investigue para qué puede utilizarse el atributo del lenguaje alignas. Qué efecto podría tener el eliminar al calificación con el mismo de los datos miembro next_read_ y next_write_.

    Hace que los elementos se alineen en posiciones de memoria múltiplos de 64.

Los datos no estarían alineados al tamaño de palabra y esto provocaría que para accederá ciertas palabra haya que entrar en dos palabras.

11. ¿Por qué se usa un valor de alineamiento de 64 al usar alignas?

Hace que los elementos se alineen en posiciones de memoria múltiplos de 64, de esta manera caben 16 elementos por línea.

12. ¿Hay alguna operación potencialmente bloqueante en atomic_buffer?

El put y el get, son los que manejan las variables atómicas.

Evaluación del rendimiento

Evalúe los 3 programas con los siguientes casos: random y count.

Evaluación del test random

Evalúe el programa generando 1000 valores y 1000000 valores. En ambos casos estudie el tiempo total de ejecución para un tamaño de búfer de 2, 10, 100 y 1000.

Para 2:

Seq_buffer: 0.002061042 0.105361685

Locked_buffer: 0.028616941 18.73863590

Atomic_buffer: 0.01259724 0.16981632

Para 10:

Seq_buffer: 0.002137741 0.109996921

Locked_buffer: 0.008571305 2.116069087

Atomic_buffer: 0.004016317 0.177129948

Para 100:

Seq_buffer: 0.003220315 0.109651342

Locked_buffer: 0.004047812 0.373784463

Atomic_buffer: 0.003512579 0.167503493

Para 1000:

Seq_buffer: 0.00274023 0.108059074

Locked_buffer: 0.003619203 0.357328746

Atomic_buffer: 0.003471816 0.175619768

Evaluación del test count

Evalúe el programa contando palabras de los cheros quijote.txt y king-lear.txt (disponibles en el directorio data).

En ambos casos estudie el tiempo total de ejecución para un tamaño de búfer de 2, 10, 100, y 1000.

Para 2:

Seq_buffer: 0.16541328 0.19820910

Locked_buffer: 7.269957313 0.588429396

Atomic_buffer: 0.161169535 0.01937044

Para 10:

Seq_buffer: 0.16892974 0.034414091

Locked_buffer: 0.835107357 0.062444748

Atomic_buffer: 0.15334936 0.023112424

Para 100:

Seq_buffer: 0.169644018 0.020594987

Locked_buffer: 0.24405580 0.028136746

Atomic_buffer: 0.153731839 0.18040623

Para 1000:

Seq_buffer: 0.165828286 0.020710286

Locked_buffer: 0.235344674 0.024994646

Atomic_buffer: 0.149476338 0.033732000