Trabajo práctico Labo: Implementacin en C++
"Planificador Round Robin"

Normativa

Límite de entrega: Mircoles 13 de abril de 2016 a las 21:00 hs.

Normativa completa: Para más detalles véase "Información sobre la cursada" en el sitio Web de la materia. (`http://www.dc.uba.ar/materias/aed2/2015/2c/informacion`)

1. Normas de entrega adicionales

- **Este TP se realiza en forma individual.** No es necesario informar el grupo.
- **La entrega es en forma electrónica únicamente**. Para ello enviar un mail a la dirección `algo2.dc+tplabo@gmail.com` con asunto 'LU XXX/YY' donde XXX/YY es su número de libreta universitaria. Se debe adjuntar en este mail únicamente el archivo PlanificadorRR.h que representa su solución del TP. El sistema debería responder en un tiempo razonable si el TP fue aprobado o no. Si no recibe confirmación de la recepción de su TP luego de 30 min., por favor informe su situación a la lista de docentes. Se pueden hacer tantas entregas como se necesiten, hasta la fecha límite de entrega.

2. Enunciado

Un planificador (en inglés *scheduler*) es un componente funcional muy importante de los sistemas operativos multitarea y multiproceso. En un instante dado, en el ordenador pueden existir diversos procesos a ser ejecutados. Sin embargo, solamente uno de ellos puede ser ejecutado simultáneamente en un microprocesador. Así surge la necesidad de que una parte del sistema operativo gestione qué proceso debe ejecutarse en cada momento para hacer un uso eficiente del procesador. La función del planificador consiste en repartir el tiempo disponible de un microprocesador entre todos los procesos que están queriendo ser ejecutados.

En la práctica existen diversas políticas de planificación, algunas más complejas que otras, y que intentan optimizar distintas métricas de prioridad. *Round-Robin* es uno de ellos. Un planificador de tipo *Round-Robin* ejecuta los procesos (asignados a un mismo procesador) repartiendo el tiempo de procesador de manera equitativa. Para ello itera circularmente entre los procesos, ejecutando cada proceso por una unidad de tiempo fija (*quantum*), luego suspendiendo ese programa para dar paso al siguiente, y así sucesivamente. Una vez que termina de ejecutar el último proceso en la lista, vuelve a ejecutar el primero.

Eventualmente, un proceso nuevo puede ser lanzado por el sistema operativo, y debe ser integrado en la lista de ejecución. Es usual insertarlo inmediatamente antes del proceso que está siendo ejecutado actualmente. Análogamente, un proceso que cumple con todas sus tareas y termina, debe poder ser removido de la lista, ya que no requerirá más tiempo de procesador en el futuro.

El sistema operativo, además, se reserva el derecho de pausar y reanudar un proceso en la lista de ejecución, por el tiempo que considere necesario. Cuando un proceso es reanudado, debe seguir siendo ejecutado en el mismo lugar que tenía en la lista de ejecución, relativo a los otros procesos, al momento de ser pausado. Nada impide que pueda haber más de un proceso pausado simultaneamente. Si el proceso que es pausado está actualmente en ejecución, automáticamente pasa a ejecutarse el proceso siguiente.

Otra cosa que puede suceder, es que espontáneamente se genere una interrupción en el sistema. Una interrupción es una señal (procediente del software o del hardware) de que algún elemento del sistema requiere atención inmediata. Si esto sucede, el planificador es detenido por el sistema operativo, y una rutina de atención para esa interrupción procede a utilizar exclusivamente el procesador. Una vez atendida, el sistema operativo devuelve el control al planificador, y este retoma su funcionalidad habitual, comenzando a ejecutar nuevamente el proceso que fue interrumpido.

Se pide:

1. Implementar en C++ la clase paramétrica `PlanificadorRR<T>`, cuya interfaz se provee en el archivo `.h` adjunto, junto con la implementación de todos los métodos públicos que en ella aparecen.
   No pueden agregar nada público. Sí pueden, y deben, agregar cosas privadas, en particular los campos que les parezcan pertinentes. También pueden agregar funciones auxiliares, tanto de instancia como estáticas, clases auxiliares, etc., pero nada en la parte pública de la clase.

2. Implementar las funciones de test no implementadas en `tests.cpp`. El correcto funcionamiento de los test **no es garantía** de aprobación.
3. La implementación dada no debe perder memoria en ningún caso. Al momento de la corrección se hará el chequeo pertinente usando *valgrind*.
4. Se sugiere chequear las precondiciones con **assert** para facilitar la depuración de errores.
5. Está prohibido utilizar la biblioteca estándar de C++ (STL) con la excepción de las siguientes librerías: **iostream**, **string** y **cassert**.
6. Se admite hasta el estándar c++03. Está prohibido usar las funcionalidades que agrega el estándar c++11 o posteriores.
7. Se sugiere repasar la sección 10.2 del Cormen.

3. Recomendaciones

El objetivo de este trabajo práctico es familiarizarse con el lenguaje C++ y las características del mismo que se usarán en esta materia (templates, memoria dinámica, etc.), de manera de llegar mejor preparados a afrontar un desarrollo más grande y complicado como el TP3.

Respecto de los archivos provistos, si intentan compilar `tests.cpp` podrán hacerlo, pero no podrán linkearlo y generar un binario porque, por supuesto, va a faltar la implementación de todos los métodos de la clase testeada.

Una sugerencia para empezar es dejar la implementación de todos los métodos necesarios escrita, pero vacía, de manera de poder compilar. Pueden comentar todos los tests que requieran métodos aún no implementados de manera de poder usar la aplicación para el testing a medida que van implementando. Tengan en cuenta que algunos métodos pueden ser necesarios para muchas de las funciones de test, por lo tanto, es aconsejable empezar por esos test.

Además de los casos de test provistos por la cátedra les recomendamos fuertemente que realicen sus propios tests.

Dado que su implementación no debe perder memoria, es una buena práctica utilizar la herramienta *valgrind* durante el desarrollo, como mínimo en la parte de testing final.