PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN
IIC2233 - PROGRAMACIÓN AVANZADA

Actividad 09

1^o **semestre 2018**

17 de mayo

Threading

Introducción

¡Oh, no! La oficina del DCC se cansó de ver *memes* todo el día y se ha tomado unas "vacaciones indefinidas". Si no son reemplazados pronto, todas las actividades de los alumnos de IIC2233 serán autocorregidas con la nota por defecto: un 2,233.

Instrucciones

Si quieres salvar tus notas y pasar el ramo, gran programador, deberás programar una oficina que pueda sustituir al DCC hasta que vuelvan de sus vacaciones.

Simulación

La simulación consistirá en un día de oficina donde el administrador se encarga de asignar las tareas a los programadores quienes, cuando las vayan terminando, deberán guardarlas en el sistema. En esta simulación, el día de oficina transcurre entre las 9:00 y las 17:00 (*i.e.* 480 minutos), donde en tu simulación un segundo real equivaldrá a 60 minutos en el programa. Para facilitar esto, se te entrega la función `reloj(minutos)` que se encargará de la conversión. Esta función llama simplemente a `time.sleep` por la cantidad de minutos que le entregues.

Tareas

Las tareas no son más que objetos que cuentan con un tiempo que demoran en ser realizadas (`duracion`). Esta clase Tarea **se te entrega**; no es necesario modificarla. Esta tiene el método `avanzar()`, el cual añade un minuto a los minutos trabajados en la tarea, y además indica si es que esta está terminada, devolviendo un booleano.

Creación de tareas

Constantemente se van creando tareas, y asignando a los programadores, esto puede ocurrir en el *main* de tu programa o dentro del administrador. Luego de ser creadas, el administrador distribuirá entre los programadores. Luego de crear cada tarea, la siguiente tarea es creada luego de t tiempo, donde t es un entero que distribuye uniforme entre 10 y 15 minutos (de la jornada).

Programadores

Para la simulación tendrás que programar un grupo de desarrolladores, quienes tendrán que realizar tareas que tengan asignadas. Cada programador puede realizar una tarea a la vez, y las tareas que le van llegando las guarda como pendiente, y las va realizando por orden en que se le asignan (*i.e.* FIFO o *first-in, first-out*). Además, los programadores pueden ser lentos o activos. Ser lento significa que tiene una probabilidad del 40 % de no avanzar en la tarea (es decir, luego de cada minuto, la probabilidad de haber avanzado un minuto de la tarea es de 60 %), mientras que un activo avanza el 90 % las veces (cuando no avanzan es porque se distraen viendo *memes*). Un programador es lento con probabilidad 30 % y activo el restante 70 %. Al terminar el horario de trabajo, el programador deja lo que esta haciendo y se retira (pista amigable: *daemon* cumple esta funcionalidad).

Administrador

Además de estos programadores, existe una persona encargada de administrar las tareas a realizar. Este administrador de tareas, a medida que se van creando las tareas, las va repartiendo a los programadores. La tarea será asignada a quien le quede menos cantidad de tareas por terminar. Al inicio de la simulación, se crea y se asigna una tarea a cada programador, y luego comienza la creación de tareas cada $t \in (10, 15)$ minutos.

Entregar Trabajo

Al terminar una tarea, los programadores deben *entregar* su trabajo en la plataforma de la universidad. Como esta plataforma no es muy buena, solo puede acceder a ella un programador a la vez. El proceso de subir el trabajo en sí toma un tiempo fijo. Si el programador es "lento", significa que perderá 20 minutos luego subirá su trabajo, los programadores "activos"se demoran 5 minutos. Para manejar que solo se entregue una sola tarea a la vez, se deberá utilizar `threading.Lock` (**no se puede** utilizar la librería `Queue` para esto).

Cierre Oficina

La oficina se cierra y se apagan los computadores a las 17:00.

Registros de eventos

Debes avisar en todo momento lo que está ocurriendo. El `print` debe notificar:

- cuando se entregan tareas a los programadores: "Nueva tarea id=4 de 35 minutos para Nebil.",
- cuando un programador termina una tarea de la forma: "Jaime: he terminado una tarea.",
- cada vez que entregan la tarea al sistema: "Belén: He entregado una Tarea.",
- al finalizar la simulación.

Notas

- Utiliza la función `reloj(minutos)` para simular por minutos.
- Para controlar el uso de recursos simultáneamente, **debe** usar `Lock` de `threading`.
- Para que *threads* que están corriendo terminen junto a tu programa, estos deberán ser *daemon*.

Requerimientos

- (2,0 pts) `Programador`
  - (0,5 pts) Implementa clase como *thread* correctamente.
  - (0,5 pts) Usa correctamente *threads* de tipo `daemon`.
  - (0,5 pts) `Programador` es capaz de trabajar en las tareas, según sus cualidades (lento o activo).
  - (0,5 pts) `Programador` es capaz de entregar tareas al sistema, según sus cualidades.
- (1,0 pt) Se maneja la concurrencia al entregar tareas con el uso de `threading.Lock`.
- (0,5 pts) Se crean las tareas correctamente con su distribución.
- (1,0 pt) Implementa clase `Administrador` y repartición de tareas correctamente.
- (0,5 pts) Correcta simulación.
- (1,0 pt) Correctos registros/*prints* de eventos.

Entrega

- **Lugar:** En su repositorio de GitHub en la **carpeta** `Actividades/AC09/`
- **Hora:** 16:30