Obligatorio 4
Programación 3
Curso 2012
Versión 1.0

1 Conocimientos previos

- Metodología de programación estructurada.
- Conocimientos básicos de C*.
- Backtracking, Greedy, Programación Dinámica, Divide & Conquer.

2 Problema

2.1 Descripción de la realidad

Un Sistema Operativo (SO) es aquel software que interactúa tanto con el hardware del computador como con las aplicaciones, es decir que es la interfaz entre los mismos. Una de sus principales responsabilidades es la de gestionar los distintos recursos de la computadora. Entre éstos, el mas importante es el uso del procesador. La forma en la que se reparte entre los procesos tiene un enorme impacto sobre el rendimiento del sistema, por lo que siempre se ha prestado especial atención a las políticas de planificación que se implementan y se han elaborado una gran cantidad de algoritmos para ello, considerando diferentes criterios dependiendo del propósito del SO.

Un Proceso **p** es un programa corriendo, y solo puede estar en uno de los siguientes estados:

– *nuevo*: Cuando el proceso es creado.
– e*jecutando*: El proceso tiene asignado un procesador y está ejecutando sus instrucciones.
– b*loqueado*: El proceso está esperando a tener todo lo necesario para estar listo para ejecutar.
– l*isto*: El proceso está listo para ejecutar, solo necesita del recurso procesador.
– f*inalizado*: El proceso finalizó su ejecución y libera el procesador.

Los procesos en un sistema operativo pueden depender para su ejecución de los resultados obtenidos por otros procesos en el sistema, es por ello que los procesos cuentan con precedencias entre sí, es decir, que un proceso para estar listo debe esperar que finalicen todos los procesos que le preceden. A dicho conjunto lo denominaremos *Pre*(**p**). Notar que si *Pre*(**p**) es vacío, significa que el proceso inmediatamente pasa a estado *listo,* esperando por ser asignado a un procesador.

El procesador es aquel que ejecuta los procesos. Solo un proceso puede tomar control del mismo en cada instante. En nuestra realidad consideramos que cada vez que un

proceso comienza a ejecutar lo hace hasta el final, sin ser interrumpido. Pensando en la transición de los estados mencionados anteriormente, una vez que el proceso pasa al estado ejecutando, el único estado al que puede pasar es al estado finalizado.

Un proceso puede ser seleccionado para ejecutar solo si se encuentra en estado listo. Si es seleccionado en el instante *t*, liberará el procesador en el instante *t* + T[**p**], siendo T[**p**] el tiempo de ejecución del proceso **p**.
Un computador puede contar con múltiples procesadores pero en nuestra realidad hay un solo procesador.

El planificador (*scheduler*) es el que decide cuál de los procesos en estado *listo* será el próximo en ser ejecutado cada vez que se libera el procesador. En nuestra realidad, éste se libera cada vez que termina el proceso que estaba en ejecución.

2.2 Requerimiento

Considere un conjunto P de procesos a ejecutar, de los cuales se conoce su tiempo de ejecución, y las precedencias entre ellos. Los procesos se identifican con un número llamado *pid* que varía entre 0 y n-1, siendo n = |P|.

En este contexto se considera que nunca existirán procesos que no estén en P.

Se desea obtener un ordenamiento entre ellos, que los ejecute a todos respetando sus precedencias.

Para esto se deberá implementar un planificador (*scheduler*) que dado el conjunto P, calcule un ordenamiento de forma que minimice el tiempo de espera total para la realidad descrita.

Dado un proceso, el tiempo de espera es la cantidad de tiempo que estuvo en estado listo, esperando por el procesador para ejecutar sus instrucciones.

El tiempo de espera total es la suma de los tiempos de espera de todos los procesos.

Las precedencias se representarán con una matriz bidimensional *Prec* donde *Prec(P**i**,P**j**)* indica con un ‘1’ si el proceso Pj es previo al proceso Pi, y con un cero en caso que no lo sea.

2.3 Formalización

Se desea formalizar el requerimiento pedido aplicando la técnica de Backtracking.

1) Realizar la formalización completa expresada en lenguaje natural, indicando forma de la solución, restricciones explícitas e implícitas, función objetivo y predicados de poda en caso que corresponda.
2) Expresar la formalización de la parte 1 en lenguaje formal.

2.4 Implementación

1) Se deberán implementar las funciones especificadas en el archivo scheduler.h.
2) Indicar por escrito cómo se refleja cada ítem de la formalización en el código de la implementación del problema.

2.5 Ejercicios

A) Preguntas:
   1) ¿Cómo implementa Backtracking la recorrida sobre el espacio de soluciones?
   2) ¿Qué diferencia a Backtracking de los algoritmos de "fuerza bruta"?
   3) ¿Qué es la función objetivo y cuando corresponde tenerla?
   4) Si no se controla un predicado de poda, ¿se pierden soluciones?
   5) ¿Cuál es la performance (en general) de los algoritmos de Backtracking?

B) ¿Es posible aplicar la técnica de Greedy para resolver el problema planteado utilizando como estrategia seleccionar en cada paso el proceso con menor tiempo de ejecución del conjunto de procesos listos? En caso de que sea posible justificar por qué conduce al óptimo. En caso de no serlo, plantee un contraejemplo.

C) Formalizar aplicando la técnica de Backtracking el problema planteado para la misma realidad, pero con una cantidad finita 'M' de procesadores. En esta realidad modificada se podrán ejecutar simultáneamente 'M' procesos e interesa saber qué procesos se están ejecutando a la vez en cada instante. Cada proceso podrá ejecutarse en cualquiera de los procesadores.

3 Preguntas

3.1 Greedy

1) ¿Cuántas alternativas se analizan en cada paso?
2) ¿Los algoritmos Greedy son aplicables a cualquier problema?
3) ¿Son aplicables a todas las instancias de un mismo problema?
4) ¿En qué propiedad basan su correctitud los algoritmos de Prim y Kruskal?
5) ¿Cómo aplica el algoritmo de Prim la propiedad MST?
6) ¿Cómo aplica el algoritmo de Kruskal la propiedad MST?
7) Mencione diferencias entre ambos algoritmos

3.2 Programación Dinámica

1) ¿Cómo se generan las soluciones de problemas utilizando la técnica de Programación Dinámica?
2) Mencione diferencias entre Programación Dinámica y D&C.
3) ¿Para qué es útil el principio de optimalidad?

3.3 Divide & Conquer

1) ¿En qué consiste la técnica de D&C?

4 Entregas semanales

Las entregas semanales se realizarán en los respectivos monitoreos.

- **Semana del 16 al 19 de Octubre**
  - Entregar formalización según ítems 2.3.1 y 2.3.2, y preguntas de 2.5.A.
- **Semana del 23 al 26 de Octubre**
  - Correcciones del 2.3.2. Todas las preguntas de la sección 3.
- **Semana del 30 de Octubre al 2 de Noviembre**
  - Ejercicios 2.5.B y 2.5.C. 2.4.2

5 Lenguaje a utilizar

El lenguaje a utilizar en este trabajo será `C` con las siguientes extensiones:

- Operadores `new` y `delete`.
- Pasaje de parámetros por referencia (uso de `&`).
- Declaración de tipos como en `C++` para registros y enumerados.
- Sobrecarga de funciones.
- Uso de `cin` y `cout`.
- Uso del tipo `bool` predefinido en `C++`.

6 Qué se espera

Para cada módulo de cabecera (`.h`) con los prototipos de las operaciones solicitadas, debe entregarse un módulo (`.cpp`) con la implementación de dichas operaciones. Debe respetarse estrictamente los prototipos especificados, esto es: nombre de la operación, tipo, orden y forma de pasaje de los parámetros y tipo de retorno. Los **módulos de cabecera** pueden bajarse de la página web del curso (`www.fing.edu.uy/inco/cursos/prog3`). Estos módulos no forman parte de la entrega, y por lo tanto, **no deben ser modificados**.

Los módulos deben funcionar en el ambiente MinGW instalado en facultad. Se espera que todos los módulos compilen sin errores (utilizando las flags "`-Wall`" y "`-Werror`"), se ejecuten sin colgarse y den los resultados correctos.

7 Forma de la entrega final

Se deberá entregar únicamente los archivos (respetando las mayúsculas en los nombres):

- scheduler.cpp

No se podrá entregar otros archivos que no sean éstos.

La primera línea de cada uno de los archivos debe contener un comentario (`/* ... */`) con la cédula de los autores, sin puntos ni dígito de verificación. Por ejemplo, si las cédulas son `1.234.567-8, 4.254.566-2 y 3.339.717-0`, la primera línea de cada archivo deberá ser exactamente:

```
/* 1234567 4254566 3339717 */
```

8 Advertencia sobre el manejo de la memoria

Cuando un programa contiene errores en el manejo de la memoria, su comportamiento puede ser inestable. Esto implica que algunas veces funciona correctamente y otras no. En ciertos casos esto puede inducir a creer (erróneamente) que ciertos programas, que en realidad son incorrectos, funcionan correctamente. Este aspecto es influenciado, entre otras cosas, por el *sistema operativo* en el que se ejecuten los programas. Recomendamos tener sumo cuidado con este punto y testear los módulos en sistemas operativos Windows NT, Windows 2000 o Windows XP. CON LA VERSION DE MINGW INSTALADA EN LAS SALAS DE INFORMÁTICA DE LA FACULTAD.

9 Sobre la individualidad del trabajo

Para este obligatorio rige el reglamento de No Individualidad publicado en la página web del curso.

10 Fecha de entrega

El trabajo debe entregarse el día **lunes 12 de noviembre del 2012** antes de las **24:00** horas.
La entrega se realizará mediante un formulario que se habilitará oportunamente en la página web del curso.