Código: 75.001

Asignatura: Estructura de la Información

Curso: 2010/2011 – primer semestre

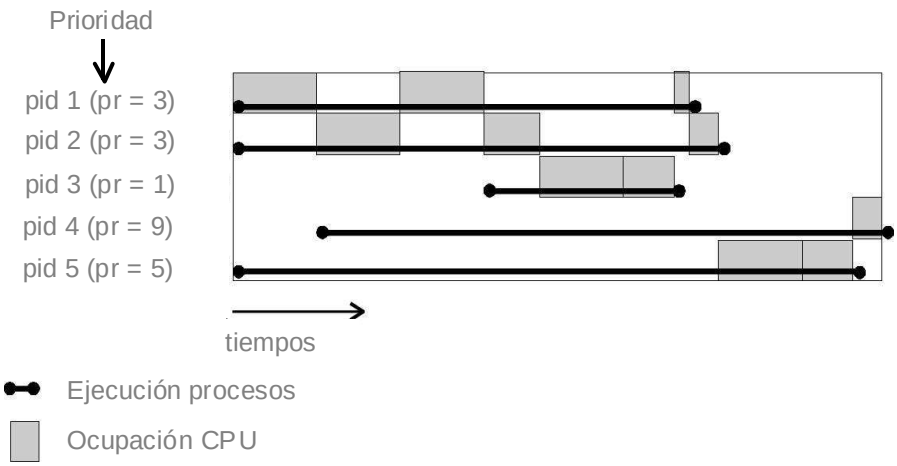
PEC 1

Ejercicio 1 [3 puntos]

Los sistemas operativos multiproceso disponen de un mecanismo que permite que el conjunto de procesos que están ejecutándose en cada momento (y compitiendo por la CPU entre ellos) sean servidos según unos ciertos criterios. Supongamos que estamos trabajando con un sistema operativo que trabaja con 10 niveles de prioridad por proceso, de 0 a 9, siendo 0 la más alta y 9 la más baja.

El sistema operativo va asignar la CPU de la siguiente forma: selecciona un proceso en ejecución (a continuación veremos cómo) y lo ejecuta durante un intervalo de tiempo limitado. Si el proceso acaba, se pasa a seleccionar otro proceso. Si no acaba, se le desasigna la CPU, se vuelve a poner en la secuencia de procesos que esperan y se selecciona otro para ser servido.

Cuando el sistema operativo debe seleccionar un proceso para servirlo selecciona aquel que tiene la prioridad más alta (más cercana a 0). Si hay más de uno, podrá seleccionar cualquiera de ellos (adviértase que esto es una simplificación de la realidad, donde normalmente las políticas para seleccionar el proceso suelen ser más complejas).



De esta forma, vemos que un proceso con una cierta prioridad no podrá ser servido a menos que hayan finalizado aquellos procesos que tienen más prioridad que él. Por otro lado, es posible eliminar un proceso de la cola de espera.

Se quieren tener las siguiente funcionalidades:

- Crear la secuencia vacía.
- Ejecutar un proceso, es decir, añadirlo a la secuencia dada, además de la información del proceso (por ejemplo, la dirección de memoria a partir de la cual reside el código a

ejecutar), su prioridad y el tiempo de CPU que necesita. Se debe retornar un identificador único, llamado PID – Process Identifier –, que identifica el proceso en la secuencia.

- Enviar un proceso a la CPU para que se sirva.
- Devolver un proceso a espera cuando ha acabado el tiempo máximo de ocupación de la CPU.
- Eliminar un proceso cuando acaba su ejecución.
- Eliminar un proceso identificado por PID de la secuencia.
- Permitir hacer un listado por pantalla de los procesos que están en espera ordenados de más a menos prioridad (a igual prioridad no importa el orden).

El TAD no decide durante cuánto tiempo se sirve un proceso ni si un proceso acaba su ejecución o se devuelve a la secuencia de procesos. Esto se decide de manera externa a nuestro TAD y por tanto no debe ser tratado.

Considerando que el número de procesos en espera no será mayor que N, se solicita:

Apartado 1.1) Define la signatura (operaciones y sintaxis) del TAD y sitúalo en la jerarquía de clases de la biblioteca de TADs de la asignatura (de donde debería extender, relaciones con otros, ...). Especializa (sustituye) los parámetros de las clases parametrizadas de la jerarquía, siempre que te sea posible, por los tipos que consideres más oportunos. Razona tu respuesta.

Apartado 1.2) Haz la especificación del TAD. Usa el estilo de especificación visto en el apartado 1.2.3. Se valorará especialmente la concisión (ausencia de elementos redundantes o innecesarios), precisión (definición correcta del resultado de las operaciones), completitud (consideración de todos los casos posibles en que se puede ejecutar cada operación) y carece de ambigüedades (conocimiento exacto de cómo se comporta cada operación en todos los casos posibles) de su solución. Es importante responder este apartado usando una descripción condicional y no procedimental. La experiencia nos demuestra que no siempre resulta fácil distinguir entre ambas descripciones, es por ello que hacemos especial hincapié insistiendo que pongáis mucha atención en vuestras definiciones. A título de ejemplo indicaremos que la descripción condicional (la correcta a usar en el contrato) de *llenar un vaso vacío con agua* seria:

> @pre el vaso se encuentra vacío.
> @post el vaso se encuentra lleno de agua

En cambio una descripción procedimental (incorrecta para usar en el contrato) tendría una forma similar a:

> @pre el vaso deberia encontrarse vacío, si no se encontrara vacío debería vaciarse.
> @post Se acerca el vaso al grifo y se echa agua hasta que esté lleno

Debéis también tener en cuenta que un contrato debería disponer de invariante siempre que esta fuera necesaria para describir el TAD.

Ejercicio 2 [2 puntos]

Se desea implementar un algoritmo para evaluar un polinomio de grado *n*. Los coeficientes del polinomio se guardan en un vector de *n+1* elementos. Por ejemplo, un polinomio de grado 2 como $2x^2+3x-5$ se guardaría en un vector de tres posiciones V=[2, 3, -5]. A continuación se muestra una solución al problema:

```
(1)   long evalua(int[] vector, int x, int grado){
(2)       long s, pot;
(3)       int i, j;
(4)       s=0;
(5)       i=0;
(6)       while(i<=grado){
(7)           j=0;
(8)           pot=1;
(9)           while(j< i){
(10)              pot *= x;
(11)              j++;
(12)          }
(13)          s+=pot*vector[i];
(14)          i++;
(15)      }
(16)      return s;
(17)  }
```

Se pide:

Apartado 2.1) Indicar detalladamente la complejidad asintótica de la solución propuesta.

Apartado 2.2) Proponer un algoritmo que mejore la eficiencia asintótica temporal de la solución propuesta, indicando dicha eficiencia (no hace falta realizar el cálculo detallado).

Ejercicio 3 [1 punto]

¿Qué estructura de datos utilizarías para representar una operación aritmética con paréntesis, por ejemplo 2 * (4 +5 / (2 +1) * 4) +1. Razonadlo y haced un esquema de como quedaría la expresión anterior. ¿Cual sería el recorrido más adecuado para evaluar el resultado de la operación?

Ejercicio 4 [1,5 puntos]

Ordena las siguientes cotas de complejidad de menor a mayor, indicando también las que son iguales: $O(0.5 \cdot 2^n)$, $O(n+1)$, $O(2^{n/2})$, $O(\log_{10} n)$, $O((n+1)^2)$, $O(n \cdot \log\ n)$, $O(2\log_6 n)$, $O(n^n)$, $O(3n^2)$, $O(n!)$, $O(n(n+\log\ n))$, $O(1)$, $O(1+1/n)$, $O(n^{n-2})$. Justifica brevemente las relaciones del orden obtenido.

Ejercicio 5 [2 puntos]

Apartado 5.1) Se conoce la representación de las colas mediante vectores y con una estructura secuencial, y también se han estudiado una serie de mejoras a través de la utilización de una gestión circular de las mismas. Enumere brevemente las ventajas y/o inconvenientes que representaría una gestión circular en la implementación de las pilas.

Apartado 5.2) En las listas ordenadas implementadas mediante vectores podemos mejorar la eficiencia de la consulta mediante una búsqueda dicotómica. En la implementación encadenada no se puede mejorar la eficiencia de esta manera, es necesario realizar una búsqueda comenzando por el primer elemento. Comentar la mejora en la eficiencia que obtendríamos si para hacer la búsqueda dispusiéramos de un puntero al elemento que ocupa la posición media de la lista.

Apartado 5.3) Hemos visto el funcionamiento de la implementación mediante montículos de los árboles binarios. ¿Cree posible una implementación por montículo de árboles ternarios?. Si lo cree posible, ¿de qué dimensión se debe de crear el vector si queremos almacenar un árbol cuasicompleto de X elementos?. Proponga un ejemplo.

Apartado 5.4) Ordenar el siguiente vector mediante el algoritmo de *heap-sort* utilizando la versión que trabaja con un solo vector. Muestre todos los estados intermedios del vector y la situación de los elementos en cada paso:

4	1	7	2	6	3