Sistemas Operativos

Departamento de Computación – FCEyN – UBA

Primer cuatrimestre de 2024

Primer parcial – 1er. cuatrimestre de 2024

Nombre y apellido: _____

Nº orden: _____ L.U.: _____ Cant. hojas: ____

1	2	3	4	Nota

- Numere las hojas entregadas. Complete en la primera hoja la cantidad total de hojas entregadas.
- Entregue esta hoja junto al examen, la misma **no** se incluye en la cantidad total de hojas entregadas.
- Cada ejercicio debe realizarse en hojas separadas y numeradas. Debe identificarse cada hoja con nombre, apellido, LU y número de orden.
- **Cada código o pseudocódigo debe estar bien explicado y justificado en castellano. ¡Obligatorio!**
- Toda suposición o decisión que tome deberá justificarla adecuadamente. Si la misma no es correcta o no se condice con el enunciado no será tomada como válida y será corregida acorde.
- La devolución de los exámenes corregidos es personal. Los pedidos de revisión se realizarán por escrito, antes de retirar el examen corregido del aula.
- **Cada ejercicio se calificará con Bien, Regular o Mal, dependiendo del apego de la resolución al punto central siendo evaluado.**
- **Los parciales se aprueban con al menos dos ejercicios B y uno R.** En otro caso, están desaprobados. No llevan nota numérica.

Ejercicio 1.

Un palíndromo es una palabra o expresión que dice lo mismo si se lee de izquierda a derecha, o de derecha a izquierda. Construya un programa que **usando pipes** siga la siguiente secuencia: Un proceso le envía o otro una cadena de caracteres que obtiene por input del usuario. Luego, éste la invierte y le envía esta nueva versión a un tercer proceso. Este último proceso también recibe la cadena original del primer proceso, las compara, decide si es o no un palíndromo e imprime el resultado por pantalla. Se considera provista, y puede usarse, la función `string invertir(string)`.

Ejercicio 2.

Considere un algoritmo de scheduling *preemptive* y con prioridades dinámicas, dónde números mayores implican mayor prioridad. $P_t(i)$ representa la prioridad del proceso i en el tiempo t:

$$P_{t+1}(i) = \begin{cases} P_t(i) + \alpha * P_t(i) & \text{si el proceso i está en espera} \\ P_t(i) + \beta * P_t(i) & \text{si el proceso i está ejecutando} \end{cases}$$

Los parámetros α y β se pueden configurar para muchos tipos diferentes de algoritmos de scheduling. Notar que la prioridad de un proceso es siempre mayor a cero. Justificando su respuesta, explique:

a) Describir el comportamiento que resulta de tener $0 \leq \alpha < \beta \leq 1$.

b) Describir el comportamiento que resulta de tener $0 \leq \beta < \alpha \leq 1$.

Al responder, considere las diferencias entre los procesos acotados por CPU y por Entrada/Salida.

Ejercicio 3.

Se desea implementar un sistema de generación de casos de test. Para tal fin, se cuenta con n procesos **Generador**, que saben generar los casos en sí, y un proceso **Coordinador**, que coordina la generación distribuida. Todos estos procesos comparten filesystem. **Implemente el código necesario para el proceso Coordinador y Generador de forma que cumpla lo pedido a continuación.** No se olvide de dejar bien en claro cuáles son las variables compartidas. Agregue comentarios donde considere necesario para esclarecer la idea de la implementación.

En primer lugar, el proceso Coordinador compila el código fuente de la aplicación a testear mediante la función `string compilarAplicación()`. Esta función devuelve como resultado la ubicación del archivo compilado.

Luego, cada proceso Generador instala la aplicación compilada mediante la función `void instalarAplicacion(int idGenerador, string ubicacion)`, que toma como primer parámetro el id (puede asumir que cada proceso Generador conoce su id) y como segundo parámetro la ubicación del archivo compilado.

Una vez que todos los Generadores terminaron de instalar la aplicación, el proceso Coordinador determina la cantidad óptima de casos de test a ser generados en conjunto por todos los procesos Generador (siempre mayor a n), mediante la función `int determinarCantCasosDeTest(string aplicacion, int n)`. Cada uno de los casos de test deberá ser generado por un único proceso Generador. *Mientras* que la cantidad de casos de test se computa, cada proceso Generador deberá empezar a generar casos de test, utilizando para eso la función `string generarTest()` que devuelve como resultado el caso de test generado. Para esta parte, no está permitido suponer de manera previa cuál Generador generará cada uno de los casos de test. Cuando termine de generar un caso, deberá constatar si quedan casos pendientes para generar y de ser así, proseguir generando.

Para finalizar, el proceso Coordinador escribe todos los casos de test a disco utilizando la función `void escribirTests(list<string> tests)` que toma como parámetro los casos de test generados.

Ejercicio 4.

Se tiene un sistema con 8 páginas y sólo 4 marcos de página. La memoria comienza vacía. Llegan los siguientes pedidos de memoria (número de página) en el siguiente orden: 3, 2, 3, 5, 2, 7, 8, 2, 6, 4.

Indique qué página se desaloja tras cada pedido utilizando los algoritmos FIFO, LRU y Second Chance y calcule el *hit-rate* en cada caso.