Sistemas Operativos

Departamento de Computación – FCEyN – UBA

Primer cuatrimestre de 2024

Recuperatorio del primer parcial – 2/07 - 1er. cuatrimestre de 2024

1	2	3	4	Nota
R	R	B	B-	A

- 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.

Implementaremos una versión básica y colaborativa del popular juego Wordle, donde se intenta adivinar la palabra del día en t intentos. Primero, el proceso padre propone una palabra de 5 caracteres. Luego, los procesos hijos, numerados del 0 al $n-1$, modificarán secuencialmente la cadena. El proceso hijo 0 recibirá del padre la palabra y deberá modificar el primer carácter de ésta para luego pasársela al hijo 1, quien modificará el segundo carácter y así sucesivamente para todos los procesos. Es decir, el hijo i recibe la cadena, la modifica y se la pasa al hijo $i+1$. El hijo $n-1$ deberá pasarle la palabra final al proceso padre, quien la usará para jugar utilizando la función `bool adivinarWordle(char *palabra)`. Si la función devuelve `true`, el padre imprime en pantalla "Ganamos" y termina; si no, la secuencia se reiniciará hasta un máximo de $t-1$ intentos.

Aclaraciones:

- Los procesos deberán comunicarse **exclusivamente** a través de pipes.
- Todos los procesos tienen acceso a una función `char *modificar_palabra(char *palabra, unsigned int pos)`, que devuelve una copia de la oración pasada por referencia con el carácter en la posición pos modificado.

Ejercicio 2.

Se tiene un sistema con un procesador y un *scheduler* **SJF** (*shortest job first*) con desalojo por entrada/salida. El sistema tiene un costo de cambio de contexto de 1 segundo.

a) Realizar el diagrama de Gantt para los siguientes procesos.

PID	Llegada	Tiempo de ejecución	Tiempo de bloqueo (inc.)
1	0	7	3-6
2	0	5	
3	1	3	
4	5	6	2-4

Donde el tiempo de bloqueo (entrada/salida) se contabiliza dentro del tiempo de ejecución, por lo que los mismos están incluidos. Es decir, si una tarea tiene tiempo de ejecución de 5 segundos y bloquea en 2-3, ejecutará en $T=0$, $T=1$, bloqueará en $T=2$, $T=3$ y ejecutará en $T=4$ (suponiendo que es la única tarea en el sistema). Su tiempo de uso de CPU será de 3 segundos.

b) Calcular el **waiting time** y **turnaround** promedio.

c) El sistema, ¿está libre de inanición? ¿Cómo lo solucionaría? Justificar.

Ejercicio 3.

Queremos implementar un juego distribuido en el que un proceso central genera un escenario para cada jugador. Estos tratar de resolverlo, cuando lo logran notifican al Coordinador, que finalmente determina un ganador. La descripción detallada es la siguiente:

El sistema estará compuesto por los procesos **Coordinador()** y **Jugador(int id)**. Tendremos un único Coordinador pero podrá haber muchos Jugadores queriendo participar. Coordinador es un proceso que nunca finaliza.

El proceso Coordinador se quedará esperando hasta tener la cantidad de jugadores que permiten iniciar el juego. Esta cantidad está definida en la variable `JUGADORES_REQUERIDOS`. Cada jugador nuevo que ingresa en el sistema deberá darle aviso al Coordinador indicando su id. Luego, se quedará esperando a que el Coordinador le dé autorización de participar de una partida.

Una vez que el Coordinador reúne la cantidad de jugadores necesarios, inicializará la partida (`void inicializar_partida()`) y avisará a los jugadores aceptados del comienzo de la misma. Luego, el Coordinador generará tantos escenarios como `JUGADORES_REQUERIDOS`. Para eso utilizará el método `escenario *nuevo_escenario()` que retorna un sólo escenario por vez (consideren existente una estructura llamada `escenario`). *A medida* que reciben sus respectivos escenarios, los jugadores que participan de la partida podrán comenzar a jugar.

Los jugadores juegan llamando a la función `boolean resolver_escenario(escenario *escenario)` que retorna un booleano que indica si logró resolver el escenario o no. En caso de no haber podido el jugador podrá volver a ejecutar `boolean resolver_escenario(escenario *escenario)` hasta conseguirlo. Cuando un jugador logre resolver su escenario, le avisará al Coordinador y esperará a saber si es el ganador.

Una vez que el Coordinador sepa que todos los jugadores finalizaron, elegirá a un ganador con la función `int elegir_un_ganador(int[] jugadores)` que devolverá el id del jugador ganador. El Coordinador avisará a los demás jugadores que ya hay un ganador y estos validarán si el id del ganador es el mismo que el propio. Si es así entonces ejecutarán `void gané()` y si no, `void perdí()`.

Finalmente, el Coordinador se volverá a preparar para comenzar otra partida.

Ejercicio 4.

Responda verdadero o falso y justifique.

a) Second Chance no sufre de la anomalía de Bedalys.

b) Si tenemos 4 marcos de página y más de 6 pedidos de página, para cualquier orden que nos llegue, Second Chance siempre es mejor que LRU.

c) Thrashing se elimina aumentando los marcos de pagina.

Nota: La anomalía de Belady se produce cuando se aumenta la cantidad de marcos de página en un algoritmo de reemplazo pero al mismo tiempo aumenta la cantidad de fallos de página para una secuencia dada.