Sistemas Operativos

Departamento de Computación – FCEyN – UBA

Primer cuatrimestre de 2023

Ejercicios de repaso – 25/04 - Primer cuatrimestre de 2023

Ejercicio 1.IPC:
Un padre inventó un juego de adivinanzas para jugar con sus 10 hijos.

Explicación del juego:

I El padre piensa un número secreto random $0 \leq S < 50$.

II En cada ronda, cada hijo i debe pensar un número random $0 \leq K_i < 100$, escribirlo en un papel, y dárselo al padre para que lo lea.

III El padre va leyendo todos los números en orden.

- Si el número $K_i > S$, entonces el hijo i pierde.
- Si el número $K_i \leq S$, entonces el hijo i sigue jugando en la siguiente ronda.

IV Luego de la ronda, cada hijo debe esperar un segundo, y luego empezar la nueva ronda (volver al paso II).

V El juego continúa indefinidamente mientras quede más de un hijo jugando. En caso de que quede un solo hijo jugando, este es el ganador.

Ejemplo (con 3 hijos):

Supongamos que el padre piensa número secreto 50.

Ronda	Hijo 1	Hijo 2	Hijo 3
Ronda 1	✓30	× 52	✓20
Ronda 2	✓49	-	✓10
Ronda 3	✓48	-	× 80

Ganador: Hijo 1 en 3 rondas.

Se pide modelar este juego en un programa utilizando pipes y señales, y respetando las siguientes condiciones:

- Cuando un hijo pierda, se debe terminar el proceso mediante una señal de sigkill.
- Cuando un hijo gane, se le debe avisar al mismo mediante una señal de sigterm. Luego, el hijo debe imprimir un mensaje "Gané", junto con su i.
- Los hijos deberán esperar un segundo entre cada ronda.

Ejercicio 2. Scheduling:

Se tiene un sistema de vigilancia que utiliza cámaras y alarmas. Estos dispositivos se comunican con un servidor que atiende distintos tipos de procesos con una política de scheduling que deberá desarrollarse para este escenario particular. Los módulos de procesamiento de video son procesos que leen el stream de video que llega desde una cámara y luego corren un algoritmo de detección de objetos. A su vez, estos procesos persisten las secuencias de video en discos del servidor. Para este tipo de procesos se quiere evitar situaciones de scheduling poco "justas". En caso de detectar patrones considerados riesgosos, el sistema debe alertar a los operadores para que actúen de inmediato. Para este fin, se cuenta con un módulo de alarma que se lanza en un proceso y que gestiona su activación. Es crítico poder garantizar que cualquier alarma se active dentro de un deadline estricto que no puede perderse. Por otro lado, el servidor cuenta con suficiente espacio para almacenar temporalmente muchas horas de frames en un formato "crudo" de video de todas las cámaras. Sin embargo, periódicamente se lanzan procesos que levantan grandes volúmenes de video grabados durante el día y le aplican un algoritmo de compresión que permite luego reemplazar las secuencias originales por otras de mucho menor tamaño. Estos procesos son lanzados durante la noche, cuando las áreas se encuentran mucho menos transitadas, por lo que las cámaras se configuran para trasmitir sólo en caso de detección de movimiento, así que la carga de procesos de procesamiento activos de video es muy baja y en forma de ráfagas de corta duracion.

Ejercicio 3. Sincronización:
Nos han pedido que esta vez hagamos un simulador pero de vuelo.
Más precisamente para un controlador áereo. Este controlador es el encargado de hacer aterrizar aviones en pistas.

El simulador funciona de la siguiente manera:

Se tienen N aviones, que serán representados por procesos y una pista de aterrizaje. Nunca una pista puede estar siendo utilizada por dos aviones al mismo tiempo, porque ya sabemos qué pasaría.

Los aviones comenzarán en vuelo. Puede asumir que cada proceso de avión toma como parámetro el índice `i` correspondiente a su id de avión y que el mismo irá de 0 a $N - 1$.

Para aterrizar, los aviones llamarán a la función `pedir_pista()`. Los aviones podrán aterrizar llamando a la operación `aterrizar()`. Tenga en cuenta que sólo podrán hacerlo si la pista está vacía. De no estarlo deberán esperar en orden del llamado `pedir_pista()`.

Al terminar de aterrizar deberán llamar a `aterrizé_bien()` y luego podrá aterrizar el siguiente avión en el orden descripto.

Una vez hayan aterrizado todos los aviones, se procede al famoso "asadazo de camaradería" entre todos los pilotos llamando (curiosamente desde el avión) a la función `asadear()` para luego poder despegar nuevamente con la función `despegar()` y volver a comenzar. No deberá bloquearse más de lo necesario.

Sólo puede utilizar mutexes, semáforos y tipos de datos básicos atómicos. Cualquier otra estructura que desee utilizar deberá definirla y escribir su código primero. Puede utilizar el tipo cola con todas las operaciones típicas (encolar, desencolar, buscar_elemento, iesimo, etc)

- Realizar el pseudocódigo del simulador.
- Extender el pseudocódigo realizado anteriormente para poder tener M pistas

Ejercicio 4. Gestión de memoria:
Considere la siguiente secuencia de referencias a páginas: $5, 5, 6, 1, 6, 2, 3, 4, 6, 5$

a) Realice el seguimiento de cada uno de los algoritmos de reemplazo listados abajo, considerando que el sistema cuenta con 4 *frames* (todos ellos inicialmente libres). Además, indique el *hit-rate* para cada algoritmo. Cual tiene mayor o menor hit-rate?

   i. FIFO.
   ii. LRU.
   iii. Second Chance.

b) Para cada algoritmo de reemplazo de páginas del ítem anterior con menor *hit-rate*, exhiba un escenario (secuencia de páginas) donde su *performance* (*hit-rate*) sea superior a los otros dos.