Sistemas Operativos

Departamento de Computación – FCEyN – UBA

Segundo cuatrimestre de 2022

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

Ejercicio 1.IPC:
Se quiere verificar una serie de numeros si son perfectos. Se considera que un número es perfecto si la suma de sus divisores propios es igual a dicho número. Por otro lado, los divisores propios de un número son todos sus divisores excluyendo al número mismo. Por ejemplo, los divisores propios de 12 son $1, 2, 3, 4$ y 6, y su suma es 16, por lo que 12 no es un número perfecto, mientras que los divisores propios de 6 son $1, 2$ y 3, y su suma da 6, por lo que 6 sí es un número perfecto. Verificar si un número es perfecto para números grandes es una operación sumamente costosa.

Actualmente se cuenta con una función de C que dado un número verifica si éste es perfecto en tiempo óptimo. Sin embargo, verificar todas las mediciones lleva horas de trabajo. Se desea automatizar este conteo con un programa que realice esta paralelización

Además, es necesario recopilar los resultados que obtuvo cada proceso con el fin de poder realizar un reporte. Para esta tarea, se cuenta con un script, `stargpt`, que integra al software `ChatGPT`. Primero ejecuta su script con el parámetro `-generar-parrafo`, luego se tipean los números perfectos obtenidos, y finalmente el programa crea un párrafo con el reporte, el cual se debe agregar en un archivo de texto. Se ha solicitado también integrar la automatización de este procedimiento.

Se pide:

a) **(15 puntos)** Realizar el código que se encargará de paralelizar el conteo de números perfectos, utilizando para ello subprocesos que se comunicarán mediante **pipes**.

Se deberá contar con un proceso coordinador y una cantidad parametrizable de subprocesos verificadores. El coordinador deberá leer la cantidad **n** de verificadores a través de los argumentos del programa. Luego, deberá utilizar las siguientes dos funciones para iterar los números que deberán ser procesados: `boolean hayNumero()` que indica si todavía queda algún número por leer, e `int leerNumero()` que lee el siguiente número, si lo hay.

Cada vez que el coordinador lea un número, deberá enviarlo inmediatamente a algún verificador para su procesamiento, pero de forma tal de asegurar que la cantidad de números que reciba cada uno sea equitativa respecto al resto.

Los verificadores tomarán cada número recibido y llamarán a una función `boolean esPerfecto(int numero)` que dado un número indica si este es perfecto. Finalmente, el coordinador deberá recibir de cada verificador la cantidad de números que cumplieron la condición y guardarlos en un arreglo `int totales[n]`.

Se sugiere utilizar el siguiente *template* (atención a todas las partes marcadas con `COMPLETAR`):

```
int main(int argc, char const* argv[]) {
        int n = atoi(argv[1]);
        // COMPLETAR
        while (hayNumero()) {
                int siguienteNumero = leerNumero();
                // COMPLETAR
        }
        int totales[n];
        // COMPLETAR
}
```

b) **(15 puntos)** Se desea modificar el código anterior para que luego de recibir los números de cada verificador, el coordinador utilice los valores del arreglo `int totales[n]` para pasárselos a un programa externo llamado `stargpt`, con el argumento `-generar-parrafo`, el cual lee por *entrada estándar* un listado de números, uno por línea, y luego imprime un párrafo por *salida estándar*. Si bien esta aplicación está programada para imprimir por *salida estándar*, el coordinador deberá ejecutarla asegurándose de que en vez de hacer eso, el párrafo se agregue al final del archivo `informe.txt`.

Considerar las siguientes funciones:

- `int open(char* archivo, O_WRONLY | O_APPEND)`: abre el archivo en modo de *escritura append* (escribe al final del archivo), retornando el descriptor correspondiente, o -1 en caso de error.
- `int dprintf(int fd, char* format, ...)`: igual a `printf(...)` pero imprime en el descriptor `fd` que se le pasa como primer parámetro.

Para resolver este ítem no se podrá utilizar la función `system()`.

El código entregado deberá estar escrito en C, y se podrán utilizar las funciones de la biblioteca estándar además de las provistas. El código deberá ser sintácticamente válido y respetar las buenas prácticas mencionadas durante las clases. Por simplicidad, siempre que esto no impacte en la solución, se permitirá omitir el chequeo de errores de aquellas funciones que retornen negativo en caso de error (tales como `open()`). Todas las decisiones implementativas deberán estar debidamente justificadas.

Ejercicio 2. Scheduling:

Una seriografía es una técnica para el estudio de los órganos en movimiento. Se realiza utilizando un aparato llamado seriógrafo, que ejecuta varias radiografías por segundo y muestra en una pantalla una serialización digital de estas imágenes, dando como resultado una especie de video. Existen seriógrafos que permiten editar algunas características de las imágenes a medida que se van generando, mientras se está llevando a cabo el estudio médico. Entre otras cosas, permiten ajustar el brillo y el contraste de las imágenes, y hacer zoom-in y zoom-out. Así, se permite una edición "en vivo" del video. Se tienen entonces los siguientes procesos:

- uno que genera las imágenes digitales a partir de los valores resultantes al irradiar al paciente
- uno que responde a los botones de ajuste de brillo y contraste
- uno que responde a los botones de ajuste de zoom

¿Qué política de scheduling permite esta toma y edición de imágenes "en vivo" de manera eficiente? Justificar.

Ejercicio 3. Sincronización:
Un cruce de río es transitado por lobos y cabras. Se usa un bote para cruzarlo con la particularidad que solo tiene capacidad para 4 animales. Para garantizar la seguridad de las especies, nunca puede pasar que se suban 3 lobos y 1 cabra o 3 cabras y 1 lobo. Cualquier otra combinación es segura.
Una vez que todos abordan() al bote, solo uno de los animales puede tomar los remos y empezar a remar().
Modelar el problema utilizando únicamente procesos lobos/cabras para cumplir con el comportamiento descrito. Asumir que solo es relevante el tráfico en una de las direcciones.

Ejercicio 4. Gestión de memoria:

a) Considere la siguiente secuencia de referencias a páginas: $5, 5, 6, 1, 6, 2, 3, 4, 6, 5$

   i. Realice el seguimiento para 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*?

      1) FIFO.
      2) LRU.
      3) Second Chance.

   ii. 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. Esta secuencia debe ser una permutación de las dadas inicialmente.

b) Suponer una secuencia de referencias a páginas que contiene repeticiones de largas secuencias de referencias a páginas seguidas ocasionalmente por una referencia a una página aleatoria. Por ejemplo, la secuencia: 0, 1, ... , 511, 431, 0, 1, ... , 511, 332, 0, 1, ... consiste en repeticiones de la secuencia 0, 1, ... , 511 seguidas por referencias aleatorias a las páginas 431 y 332.

   i. ¿Por qué los algoritmos de reemplazo LRU, FIFO y Second Chance no serán efectivos para manejar esta dinámica para una cantidad de frames menor al largo de la secuencia?

   ii. Si este programa tuviera disponibles 500 frames, describir un enfoque de reemplazo de páginas que funcione mejor que los algoritmos LRU, FIFO o Second Chance.