- 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.
- Los parciales tienen tres notas: I (Insuficiente): 0 a 59 pts, A- (Aprobado condicional): 60 a 64 pts y A (Aprobado): 65 a 100 pts. No se puede aprobar con A- ambos parciales. Los recuperatorios tienen dos notas: I: 0 a 64 pts y A: 65 a 100 pts.

Ejercicio 1.- Sincronización y concurrencia. **(30 puntos)**

Una montaña rusa en un parque de diversiones cuenta con **N** carritos con capacidad para **C** personas. La atracción cuenta con 3 secciones: la plataforma de inicio, la zona de viaje y la plataforma de retiro. Continuamente muchas personas entran emocionadas a vivir la adrenalina de la montaña rusa. Cada carrito va llegando de a uno a la plataforma de inicio donde se espera que la gente se suba a ellos. Solamente cuando está lleno, el carrito está listo para empezar el viaje y otro carrito debe llegar a la plataforma de inicio. De esta forma deben suceder los **N** carritos. Cuando el carrito termina todo su recorrido, llega a la zona de retiro, donde tiene que esperar que todos los pasajeros se bajen. Notar que los carritos claramente llegan a la plataforma de retiro en orden en el que salieron. Una vez concluido este hecho, el carrito avanza nuevamente a la plataforma de inicio con el fin de ser usado nuevamente para otro viaje.

Se requiere modelar y simular el funcionamiento de esta montaña rusa usando procesos y las herramientas de sincronización vistas en clase. Explique el modelo propuesto (descripción de los procesos). Justifique las herramientas de sincronización utilizadas. Escriba el pseudocódigo para cada tipo de proceso propuesto.

Ejercicio 2.- Scheduling. **(25 puntos)**

El sistema de control de una fase de una central nuclear ejecuta varios procesos. Algunos de estos procesos controlan el funcionamiento de aspectos esenciales de la operación del reactor (como la presión de agua y la temperatura) y activan una alarma en caso de que algún parámetro tome un valor riesgoso. A su vez, estos datos son tomados por otros procesos que los representan en gráficos e información visual en una pantalla para que las personas que operan en esta fase tengan conocimiento de lo que está sucediendo a cada momento. Estas personas determinan la forma en la que se visualizan los datos según lo que se precise monitorear y pueden cambiarla en cualquier momento, por ejemplo, eligiendo un tipo de gráfico en particular o aplicando filtros. También hay procesos que exportan los datos recopilados por día, semana y mes en un formato que facilita su análisis posterior.

Proponer un algoritmo de *scheduling* apropiado para este escenario, justificando claramente las decisiones tomadas y las estructuras de datos y políticas utilizadas. Considere los requerimientos según las necesidades de cada tipo de procesos.

Ejercicio 3.- Comunicación Inter-procesos **(30 puntos)**

Un grupo de investigadores descubrió un patrón inusual mientras analizaban los datos de las mediciones de energía de estrellas lejanas. Para ciertas estrellas, la cantidad total de números perfectos dentro de sus mediciones resultó ser también un número perfecto. 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, ya que requiere primero realizar una factorización en primos del número para luego encontrar sus divisores propios, y finalmente sumarlos para comparar el resultado contra el número original.

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 para todas las estrellas lleva horas de trabajo. La solución actual es dividir el conjunto de mediciones de una estrella entre integrantes del grupo, para que cada uno cuente en su computadora cuántos cumplen la propiedad, paralelizando el trabajo y disminuyendo el tiempo total del conteo. Se desea automatizar este conteo con un programa que realice esta paralelización en una supercomputadora con 50000 núcleos.

Además, es necesario recopilar los resultados que obtuvo cada integrante del grupo para cada estrella 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 dentro del conjunto de mediciones de una estrella, 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 4.- Gestión de memoria: **(15 puntos)**
Considere la siguiente secuencia de referencias a páginas: 1, 2, 2, 1, 3, 4, 5, 6, 1, 2, 2, 1, 3.

a) **(8 puntos)** 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.

i. LRU.
ii. Segunda oportunidad.

b) **(7 puntos)** Es posible cambiar el *hit-rate* del algoritmo que obtuvo el valor mayor, cambiando el orden de solicitud de las páginas. Reordene las solicitudes de página del punto anterior de modo que el *hit-rate* del algoritmo ganador en el punto anterior ahora sea el más bajo.