Actividad Práctica de Laboratorio Nro. 3:

- ***Tema:*** Procesos, comunicación y sincronización
- ***Descripción:*** Codificar todos los programas mencionados en el presente trabajo en C o C++ bajo plataforma GNU/Linux
- ***Formato de entrega:*** Según el protocolo especificado anteriormente
- ***Documentación:*** Todos los programas que se entreguen deben tener un encabezado y un fin de archivo. Dentro del encabezado deben figurar el nombre del programas, el número de APL al que pertenece y el número de ejercicio al que corresponde, el nombre de cada uno de los integrantes detallando nombre y apellido y el número de DNI de cada uno (tenga en cuenta que para pasar la nota final de la APL será usada dicha información, y no se le asignará la nota a ningún alumno que no figure en todos los archivos con todos sus datos), también deberá indicar el número de entrega a la que corresponde (entrega, primera reentrega, segunda reentrega, etc.).
- ***Evaluación:*** Luego de entregado los docentes y ayudantes procederán a evaluar los ejercicios resueltos. Estas evaluaciones estarán disponibles en el sitio de la cátedra (www.sisop.com.ar) donde además del estado de la corrección se podrá ver un detalle de las pruebas realizadas y los defectos encontrados.

  Cada ayudante podrá determinar si un determinado grupo debe o no rendir coloquio sobre el contenido presentado.

  Importante: Cada ejercicio cuenta con una lista de validaciones mínimas que se realizará, esto no implica que se puedan hacer otras validaciones al momento de evaluar el trabajo presentado.

- ***Planificación:***

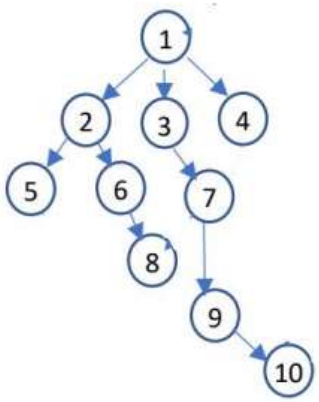
Tipo	Fecha Inicial	Fecha final	Cursada presencial y semi-presencial	Cursada Virtual	Exámenes Libres
Validación			Obligatoria	Recomendada	No Aplica
Entrega Final	30/11/2020	01/12/2020	No Aplica	Obligatoria	No Aplica
Recuperatorio	14/12/2020	15/12/2020	Obligatoria	Obligatoria	No Aplica

- ***Ejercicios:***

Ejercicio 1:

Se desea desarrollar un programa que genere el siguiente árbol de proceso, el cual está comprendido de 5 descendencias:

- Descendencia 1: Proceso 1
- Descendencia 2: Procesos 2, 3 y 4
- Descendencia 3: Procesos 5, 6 y 7
- Descendencia 4: Procesos 8 y 9
- Descendencia 5: Proceso 10



Cada proceso debe imprimir su nivel de descendencia, su Pid y el Pid de todos los procesos que lo precedieron. Adicionalmente, se puede recibir un parámetro N que indique cual es el último nivel de descendencia a mostrar. Por ej: Si N=2 crearía únicamente los procesos 1,2,3 y 4. Si el parámetro N no es enviado, genera el árbol completo.
Poner una espera, por tiempo o hasta apretar una tecla, antes de finalizar para permitir validar la correcta ejecución.

Criterios de corrección:

Control	Criticidad
Debe cumplir con el enunciado	Obligatorio
Se entrega el makefile correspondiente	Obligatorio
Se válida que el parámetro de entrada sea un número natural menos o igual a 5	Obligatorio
El ejecutable cuenta con una ayuda (-h, --help)	Obligatorio
Se debe poder evaluar el árbol de procesos	Obligatorio
Validación correcta de los parámetros	Obligatorio

Ejercicio 2:

Se deberá generar un programa que reciba por parámetros 3 valores: un path a un directorio de “entrada”, un path a un directorio de “salida” y el nivel de paralelismo.
Una vez iniciado el programa se deberá repartir todos los archivos del directorio de entrada en forma equitativa entre los N threads que se han generado. Cada thread deberá contar la cantidad de letras totales en el contenido de cada archivo, generando un nuevo archivo en el directorio de salida con el mismo nombre original y cuyo contenido sea:

Hora de inicio
Número de Thread
Cantidad de vocales
Cantidad de consonantes
Cantidad de “otros caracteres”.
Hora de finalización

Adicionalmente se deberá mostrar por pantalla:

- Cada thread, su número y qué archivos le tocó analizar
- El nombre del archivo con menor cantidad de caracteres totales.
- El nombre archivo con mayor cantidad de caracteres totales.
- El nombre del primer archivo finalizado.
- El nombre del último archivo finalizado

Criterios de corrección:

Control	Criticidad
Debe cumplir con el enunciado	Obligatorio
Se entrega el makefile correspondiente	Obligatorio
El ejecutable cuenta con una ayuda (-h, --help)	Obligatorio
Validación correcta de los parámetros	Obligatorio
Proveer archivos de ejemplo para realizar pruebas	Obligatorio
Utiliza la biblioteca de C++ para crear threads	Obligatorio
Acepta correctamente paths absolutos y relativos	Obligatorio
No se permiten soluciones que utilicen los syscalls system() o popen()	Obligatorio

Ejercicio 3:

Implementar el clásico juego de la memoria “Memotest”, pero alfabético.
Para ello deberá crear dos procesos no emparentados que utilicen memoria compartida y se sincronicen con semáforos.
Deberá existir un proceso “Cliente”, cuya tarea será mostrar por pantalla el estado actual del tablero y leer desde teclado el par de casillas que el usuario quiere destapar.
También existirá un proceso “Servidor”, que será el encargado de actualizar el estado del tablero en base al par de casillas ingresado, así como controlar la finalización partida.

Características del diseño:

- ✓ El tablero tendrá 16 casillas (4 filas x 4 columnas)
- ✓ Se debe garantizar que no se pueda ejecutar más de un cliente a la vez conectado al mismo servidor
- ✓ Se deberá garantizar que solo pueda haber un servidor por computadora
- ✓ Cada vez que se genere una nueva partida, el servidor deberá rellenar de manera aleatoria el tablero con 8 pares de letras mayúsculas (A-Z). Cada letra seleccionada solo deberá aparecer dos veces en posiciones también aleatorias
- ✓ El servidor se ejecutará y quedará a la espera de que un cliente se ejecute
- ✓ Tanto el cliente como el servidor deberán ignorar la señal SIGINT (Ctrl-C)
- ✓ El servidor deberá finalizar al recibir una señal SIGUSR1, siempre y cuando no haya ninguna partida en progreso
- ✓ El cliente deberá mostrar por pantalla el tiempo para saber cuánto se tarda en resolver el juego

Ejemplo la interfaz del cliente:
Esta interfaz es solo a nivel de ejemplo y quedará a criterio del grupo el diseño de la misma, por ejemplo, sería válido si quisieran hacer el ingreso de la fila y la columna en dos entradas separadas.

```
  0 1 2 3
0 - - - -
1 - - - -
2 - - - -
3 - - - -

Ingrese Fila-Columna: 0-0
```

```
  0 1 2 3
0 A - - -
1 - - - -
2 - - - -
3 - - - -

Ingrese Fila-Columna:
```

Criterios de corrección:

Control	Criticidad
Funciona según enunciado	Obligatorio
Compila sin errores con el makefile entregado	Obligatorio
Cumple con todas las características de diseño indicadas	Obligatorio
Cierra correctamente los recursos utilizados al finalizar	Obligatorio
No hay pérdida de información	Obligatorio
El ejecutable cuenta con una ayuda (-h, --help)	Obligatorio

Ejercicio 4:

Implementar un sistema de almacenamiento de datos que permita crear y eliminar colecciones (Tablas), así como insertar, consultar y eliminar sus filas (Registros).
Para ello deberá crear dos procesos no emparentados que se comuniquen a través de FIFOs.

Características del diseño:
Existirá un proceso que llamaremos **Interfaz,** cuya tarea será brindar al usuario la posibilidad de ingresar acciones que serán validadas y posteriormente enviadas, a través de un FIFO, a un proceso que llamaremos **GCD**, éste último será el encargado de ejecutar la acción sobre la colección y enviar de nuevo al proceso **Interfaz** la respuesta correspondiente por otro FIFO, dicha respuesta se mostrará por pantalla.

Acciones a realizar:

- **CREATE COLLECTION** - Crea una nueva colección
  Sintaxis: CREATE COLLECTION nombre campo1 campo2 [.. campoN]
  Ejemplo: CREATE COLLECTION Producto código descripción
  El campo1 será la clave primaria (no permite duplicados).
  Es mandatorio el nombre y al menos dos campos.
  Si la colección ya existe, el GCD responderá con el mensaje; **ERROR: Colección existente**.

- **DROP COLLECTION** - Elimina una colección existente
  sintaxis: DROP COLLECTION nombre
  Ejemplo: DROP COLLECTION Producto
  Es obligatorio el nombre.
  Si la colección no existe, el GCD responderá con el mensaje; **ERROR: Colección inexistente**.

- **ADD IN** - Agrega una fila en una colección existente
  sintaxis: ADD IN nombre campo1=valor1 campo2=valor2 [.. campoN=valorN]
  Ejemplo: ADD IN Producto código=1 descripción=Harina
  Es mandatorio el nombre y al menos dos campos con sus valores.
  Si la fila existe (comparando clave primaria) responderá con el mensaje; **ERROR: Fila existente**.
  Si los nombres de los campos o la cantidad de los mismos no coinciden con los de la colección se responderá con el mensaje; **ERROR: Campos inválidos**.

- **FIND** - Consulta una fila en una colección existente
  Sintáxis: FIND nombre valor
  Ejemplo: FIND Producto 1
  Es mandatorio el nombre y el valor a encontrar.
  Si la fila existe (comparando el valor con el campo clave primaria), se responderá con los valores de los campos de dicha fila, por ejemplo; **1 Harina**.
  Si la fila no existe se responderá **ERROR: Fila inexistente**.

- **REMOVE** - Elimina una fila en una colección existente
  sintaxis: REMOVE nombre valor
  Ejemplo: REMOVE Producto 1
  Es mandatorio el nombre y el valor a eliminar.
  Si la fila no existe (comparando clave primaria) se responderá con el mensaje; **ERROR: Fila inexistente**.

Al realizar una acción de manera exitosa, se responderá con el mensaje **Acción OK**

El proceso **GCD** deberá ejecutar en segundo plano y gestionar las acciones sobre las colecciones de datos, que se guardarán en disco como archivos de texto plano.
Finalizará su ejecución al recibir una señal SIGUSR1.

El proceso **Interfaz** ejecutará en primer plano, quedando a la espera de las acciones mencionadas anteriormente cuya confirmación será con un ENTER. Deberá validar la sintaxis de las acciones, si ésta es incorrecta se deberá informar por pantalla **ERROR: Sintaxis incorrecta**, de lo contrario se procederá a enviar por el FIFO la acción pertinente. Para finalizar la ejecución se deberá ingresar como acción la palabra **QUIT**.

Criterios de corrección:

Control	Criticidad
Compila sin errores con el makefile entregado	Obligatorio
Funciona según enunciado	Obligatorio
Existe algún tipo de ayuda para manejar la ejecución de los procesos	Obligatorio
Proveer archivos de ejemplo para realizar pruebas	Obligatorio
Cierra correctamente los recursos utilizados	Obligatorio
Finalizan correctamente todos los procesos	Obligatorio
El ejecutable cuenta con una ayuda (-h, --help)	Obligatorio

Ejercicio 5:

Repite la consigna del ejercicio 4 pero la interfaz en lugar de comunicarse con el proceso GCD a través de FIFO lo hace mediante Sockets ya que se encuentra físicamente en computadoras diferentes.
Debe contar con parámetros o bien un archivo de configuración para poder indicar el puerto y la dirección IP a conectarse.

Criterios de corrección:

Control	Criticidad
Compila sin errores con el makefile entregado	Obligatorio
Funciona según enunciado	Obligatorio
Proveer archivos de ejemplo para realizar pruebas	Obligatorio
Existe algún tipo de ayuda para manejar la ejecución de los procesos	Obligatorio
Cierra correctamente los recursos utilizados	Obligatorio
Finalizan correctamente todos los procesos	Obligatorio
El ejecutable cuenta con una ayuda (-h, --help)	Obligatorio