Trabajo Práctico Nro. 3 (GRUPAL):

- ***Tema:*** Procesos, comunicación y sincronización
- ***Descripción:*** Codificar todos los programas mencionados en el presente trabajo, teniendo especialmente en cuenta las recomendaciones sobre programación mencionadas en la introducción de este trabajo
- ***Formato de entrega:*** Siguiendo el protocolo especificado anteriormente. Recomendamos realizar la entrega presencial
- ***Documentación:*** Todos los programas que se entreguen deben tener un encabezado y un fin de archivo. Dentro del encabezado deben figurar el nombre del programa, el trabajo práctico al que pertenece y el número de ejercicio dentro del trabajo práctico 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 del trabajo práctico 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 el trabajo práctico los ayudantes procederán a evaluar los ejercicios resueltos, en caso de encontrar errores se documentará en la carátula del TP que será devuelta al grupo con la evaluación final del TP y una fecha de reentrega en caso de ser necesaria (en caso de no cumplir con dicha fecha de reentrega el trabajo práctico será desaprobado). Cada ayudante podrá determinar si un determinado grupo debe o no rendir coloquio sobre el trabajo práctico presentado.

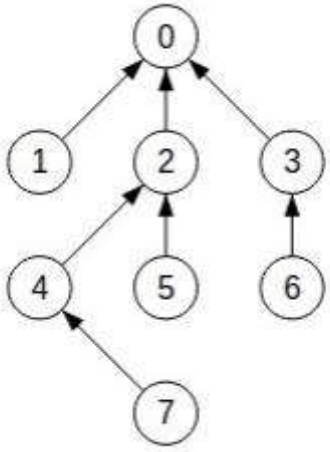
  Las notas sobre los trabajos también estarán disponibles en el sitio de la cátedra (www.sisop.com.ar) donde además del estado de la corrección se podrá ver un detalle de las pruebas realizadas y los defectos encontrados

  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.
- ***Fecha de entrega:*** **17/06/2020 o 18/06/2020**, dependiendo del día que se curse la materia.

Ejercicios:

Ejercicio 1:

Se solicita desarrollar un programa que dado un parámetro N de entrada recree el siguiente árbol de procesos. Cada proceso deberá imprimir su pid y el de toda su ascendencia. Ejemplo el proceso 7 deberá imprimir 7(PID) – 4(PID) – 2(PID) – 0(PID).



Adicionalmente, el proceso representado con PID 7 en el grafo, deberá crear N jerarquías de procesos hijos adicionales.
Por ejemplo:

Si N=2, deberá crear un proceso hijo que imprima por pantalla 8(PID) - 7(PID) – 4(PID) – 2(PID) – 0(PID) y a su vez este crear otro proceso hijo que muestre: 9(PID) - 8(PID) - 7(PID) – 4(PID) – 2(PID) – 0(PID).

Criterios de corrección:

Control	Criticidad
Compila sin errores con el makefile entregado	Obligatorio
Funciona correctamente según enunciado	Obligatorio
Existe algún tipo de ayuda para ejecutar el proceso	Obligatorio
Utiliza la función fork para crear procesos	Obligatorio
Valida que el parámetro de entrada sea un número entero no negativo	Obligatorio

Ejercicio 2:

Un determinado grupo de investigadores lo han contratado para desarrollar una función matemática que ejecute en varios Threads. La misma trabaja con la sucesión de Fibonacci.
El flujo del programa es el siguiente:
Primero recibirá por parámetro un número N y creará N thread donde cada uno ejecutará el algoritmo matemático mencionado bajo la siguiente lógica.
Supongamos que N=3
El primer Thread deberá sumar los 3 primeros números de Fibonacci (en este caso 1+1+2)
El segundo Thread deberá sumar los 2 primeros números de Fibonacci (en este caso 1+1)
El tercer Thread deberá devolver el primer número de la sucesión de Fibonacci (en este caso 1)
(los resultados lo irán almacenando en un array global donde cada thread tendrá una posición asignada para no pisar data)

Una vez finalizado el proceso se generarán 2 nuevos threads, el primero sumará todos los elementos del array (en este caso 4+2+1) mientras que el segundo los multiplicará (4*2*1). La información será almacenada nuevamente en un vector global, en este caso de 2 elementos.
Por último, se restará el resultado del producto con el resultado de la suma, el valor final deberá ser mostrado por pantalla.

Criterios de corrección:

Control	Criticidad
Utilizar la biblioteca thread de C++	Obligatorio
Compila sin errores con el makefile entregado	Obligatorio
Funciona correctamente según enunciado	Obligatorio
Existe algún tipo de ayuda para ejecutar el proceso	Obligatorio
Valida que el parámetro de entrada sea un número entero mayor a 0	Obligatorio
Utiliza una clase hiloFibonacci cuyo constructor reciba el número N a calcular en la sucesión	Opcional

Ejercicio 3:

El club de barrio “La Juanita” está teniendo problemas para llevar a cabo la correcta contabilidad del pago
de las cuotas de sus asociados, así como el control de la correcta asistencia en los días correspondientes
debido al incremento de socios que se ha dado en el último año. Para automatizar de la forma más económica posible lo contactan a usted para brindar solución a dicha problemática.
El club mantiene los siguientes archivos (Todos los archivos separan registros por punto y coma (;)):

- Socios: contiene: Nombre;Apellido;DNI;NombreDelDeporte;DíaAsistencia. Por ahora no está permitido anotarse a más de un deporte y además solo se puede asistir un día a la semana.
    - El campo nombreDelDeporte puede tomar los valores: “Futbol”,”Voley”,”Basquet” y “Natación” (respetando las minúsculas y mayúsculas)
    - El campo Día puede tomar los valores “Lunes, Martes, Miercoles, Jueves, Viernes, Sabado y Domingo”
- Pagos: Archivo separado por punto y coma que posee el siguiente formato: DNI;Fecha. Si el pago se realiza del 1 al 10 se realiza un descuento del 10%.
    - Formato de fecha: YYYY-MM-DD
- Asistencia: Archivo separado por Punto y Coma que posee el siguiente formato: DNI;DiaAsistencia
    - El campo DiaAsistencia toma los mismos valores que el del archivo Socios

En cuanto a los deportes que se practican, se tiene la siguiente información (no se encuentra en ningún archivo, puede estar como valores en el código):

Deported	Días posibles	Valor de la cuota
Fútbol	Lunes - Miércoles	$ 1.000
Vóley	Martes - Jueves	$ 1.200
Básquet	Viernes	$ 1.300
Natación	Sábado	$ 1.800

Lo que necesita principalmente el club es:

1. Calcular automáticamente el monto total cobrado en el mes (contemplando los descuentos).
2. Detectar asociados que no hayan pagado la cuota mensual.
3. Detectar asistencias en días que no corresponden.

Dado que en el club utilizan Linux como sistema operativo, después de la etapa de análisis con el personal
del club, se definen los siguientes lineamientos:

1. Habrá dos procesos productores, uno leerá la información del archivo Pagos y el otro del archivo Asistencia, ambos enviarán la información a través de memoria compartida a un proceso consumidor, que será el encargado de realizar el proceso correspondiente para responder a las necesidades del club, mostrando los resultados por pantalla.
2. Tiene que existir alternancia estricta en el envío de información entre los procesos productores.

Criterios de corrección:

Control	Criticidad
Compila sin errores con el makefile entregado	Obligatorio
Funciona correctamente según enunciado	Obligatorio
Existe algún tipo de ayuda para la ejecución de los procesos	Obligatorio
Utiliza semáforos Posix y Memoria Compartida	Obligatorio
Cierra correctamente los recursos utilizados	Obligatorio
Finalizan correctamente todos los procesos	Obligatorio
No hay perdida de información	Obligatorio

Ejercicio 4:

En una PyME de desarrollo de software, el área de testing necesita de una herramienta que pueda mínimamente detectar “fugas” de memoria, así como excesiva utilización de CPU, para esto le solicita a usted, pasante de la empresa, que implemente dicha herramienta siguiendo con las especificaciones de diseño que fueron realizadas por el líder del equipo de testing;

1. Codificar un proceso Principal cuya tarea será crear dos procesos hijos (Control y Registro), los tres procesos deberán quedar ejecutando en segundo plano. Una vez creados ambos procesos hijos, el proceso Principal deberá quedar a la espera de la señal SIGUSR1, cuando se reciba dicha señal los tres procesos deberán finalizar.
2. El proceso Control deberá detectar (cada un segundo) si algún proceso en ejecución supera un valor límite de consumo de memoria o CPU, si dicha situación ocurre deberá enviar al proceso Registro un conjunto de datos a través de un FIFO.

Los datos a enviar son:

- PID
- Nombre
- Tipo de exceso (Memoria – CPU – Ambos)
- Hora del sistema (HH:MM: SS)

Los valores **límite** deben ser pasados como parámetros al proceso **Principal**.

1. El proceso **Registro** recibirá los datos y los registrará en un **archivo** (una línea por cada proceso), cabe destacar que solo se registrará el mismo proceso a los sumo dos veces, por ejemplo;

${PID}: Supera CPU (Primer registro)
${PID}: Supera memoria o Ambos (Segundo registro)
ó
${PID}: Supera memoria (Primer registro)
${PID}: Supera CPU o Ambos (Segundo registro)
(Nota el campo ${PID} deberá ser reemplazado por el PID del proceso que haya generado el evento)
Solo se registrará una vez aquellos procesos donde el primer registro sea por exceso de tipo Ambos o que solo excedan siempre CPU y no memoria o viceversa.

Criterios de corrección:

Control	Criticidad
Compila sin errores con el makefile entregado	Obligatorio
Funciona correctamente según enunciado	Obligatorio
Existe algún tipo de ayuda para la ejecución del proceso	Obligatorio
Valida correctamente los parámetros	Obligatorio
Cierra correctamente los recursos utilizados	Obligatorio
Finalizan correctamente todos los procesos	Obligatorio

Ejercicio 5:

La UNLaM desea centralizar la gestión de asistencias, para ello requiere un sistema que de acuerdo al rol del usuario que lo utilice permita:

El servidor contendra:

El listado de alumnos/docentes de la Universidad. Un archivo con el siguiente formato:

- Usuario.txt

NOMBRE|CONTRASEÑA|ROL|COD_COMISION
LUCA|x5sqsa|D|1
JUAN|x2s1sa|A|1
PEPE|x3sbsa|A|1
ADRI|xaxqsa|D|2
SOFI|abxdda|A|2

El listado de todas las asistencias que se generarán.
Mostrar por pantalla los datos necesarios para loguearse.
Mostrará por pantalla y guardará en un archivo de extensión .log todas las conexiones que se establezcan como la información que reciba y envíe a sus clientes.

Como cliente:

- Al conectarse con el servidor deberá mostrar una pantalla para ingresar usuario y password. Dependiendo el rol con el que se logue deberá:
- Docente (D)
    - Consultar asistencias de sus alumnos. Enviará como parámetro la fecha (formato de todas las fechas yyyy-mm-dd) a consultar, y el sistema imprimirá el listado de alumnos del curso asignado mostrando el presentismo correspondiente.
    - Cargar asistencias. Como primer paso escribirá la fecha de asistencia y luego listará por pantalla (de uno a la vez) los nombres de todos los alumnos asociados a esa comisión. El docente deberá indicar si están presentes o no completando con la letra P o A en cada caso.
    - Asistencia_[FECHA]_[COMISIÓN].txt

NOMBRE|PRESENTE
JUAN|A
PEPE|P

NOTA: No se requieren parámetros, con la fecha y hora actual se deberá determinar el listado de alumnos. Si el archivo ya se encuentra cargado arrojar un error

- Alumno (A)
    - Consultar si asistió en determinado día
    - Consultar el porcentaje de asistencias/inasistencias

Criterios de corrección:

Control	Criticidad
Compila sin errores con el makefile entregado .	Obligatorio
Funciona correctamente según enunciado .	Obligatorio
Existe algún tipo de ayuda.	Obligatorio
Utiliza socket. Modelo cliente servidor.	Obligatorio
Cierra correctamente los recursos utilizados.	Obligatorio
Finalizan correctamente todos los procesos.	Obligatorio
Soporta múltiples clientes en simultaneo.	Obligatorio