Trabajo Práctico Nro. 1 (GRUPAL):

- ***Tema:*** Programación de scripts en tecnología bash
- ***Descripción:*** Se programarán todos los scripts 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 scripts que se entreguen deben tener un encabezado y un fin de archivo. Dentro del encabezado deben figurar el nombre del script, 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:*** **29/04/2019 o 30/04/2019**, dependiendo del día que se curse la materia.

Introducción:

La finalidad del presente práctico es que los alumnos adquieran un cierto entrenamiento sobre la programación de shell scripts, practicando el uso de utilitarios comunes provistos por los sistemas operativos (GNU/Linux). Todos los scripts, están orientados a la administración de una máquina, o red y pueden ser interrelacionados de tal manera de darles una funcionalidad real.
Cabe destacar que todos los scripts deben poder ser ejecutados en forma batch o interactiva, y que en ningún caso serán probados con el usuario root, por lo cual deben tener en cuenta al realizarlos, no intentar utilizar comandos, directorios, u otros recursos que solo están disponibles para dicho usuario.
Todos los scripts deberán funcionar en las instalaciones del laboratorio 266, ya que es ahí donde serán controlados por el grupo docente.
Para un correcto y uniforme funcionamiento, todos ellos deberán respetar algunos lineamientos generales indicados en el trabajo práctico anterior.

Ejercicios:

Tip: En caso de tener problemas con un script bajado desde un dispositivo formateado con DOS, y que contiene ^M al final de cada línea puede usar el siguiente comando para eliminarlos:

```
tr -d '\r' <archivo_con_M >archivo_sin_M
```

En caso de ejecutar scripts que usen el archivo de passwords, en el laboratorio, debe cambiar “cat /etc/passwd” por “getent passwd”

Ejercicio 1:

Utilizaremos el archivo de contribuyentes de la AFIP, el cual puede descargar del siguiente vinculo:
http://www.afip.gob.ar/genericos/cInscripcion/archivos/apellidoNombreDenominacion.zip

Y tiene la explicación sobre formato y definición de los campos en la página de descarga:
http://www.afip.gob.ar/genericos/cInscripcion/archivoCompleto.asp

Ejecute el siguiente script y responda las preguntas que se detallan a continuación del mismo, tenga en cuenta que antes de poder ejecutarlo deberá otorgarle permisos de ejecución.

FORMATO DE ENTREGA: Se deberá entregar el script en un archivo con extensión .sh y las respuestas contenidas en el mismo como líneas comentadas.

```
#! /bin/bash

if [ $# != 2 ]; then
      echo "..."
      exit
fi

if ! [ -f "$2" ]; then
      echo "..."
      exit
fi

X=$1

awk 'BEGIN {
        FIELDWIDTHS = "11 30"
}
$1 == "'$X'" {
        print "Nombre                          CUIT"
        print "------------------------------ -----------"
        print $2" "$1
        exit
}' $2
```

Responda:

a) ¿Qué significa la línea "#!/bin/bash"?
b) ¿Con qué comando y de qué manera otorgó los permisos de ejecución al script?
c) Explique el objetivo general del script.
d) Complete los echo "..." con mensajes acordes a cada situación según lo que se valida y renombre la variable X con un nombre acorde a su utilidad.
e) Explique de manera general la utilidad de AWK.
f) Es posible ejecutar este script para procesar varios archivos a la vez en una única llamada? Ejemplifique.

IMPORTANTE: No copiar el archivo de padrón en la entrega del ejercicio.

Ejercicio 2:

Mejore el script del punto anterior, para que se pueda filtrar tanto por CUIT, DNI o por nombre o parte del mismo, dependiendo del modificador -c, -d o -n respectivamente.
También mejore la salida para que muestre un detalle de toda la información disponible.
Y al final un resumen de cantidad de tipos de contribuyente filtrados.

IMPORTANTE: No copiar el archivo de padrón en la entrega del ejercicio.

Criterios de corrección:

Control	Criticidad
El script ofrece ayuda con –h, -? o –help explicando cómo se lo debe invocar	Obligatorio
Valida cantidad de parámetros mínimos.	Obligatorio
Funciona correctamente según enunciado.	Obligatorio
Resolver las 3 formas de filtro en una única llamada de AWK.	Deseable
Funciona con rutas relativas, absolutas o con espacios.	Obligatorio

Ejercicio 3:

Una agencia de lotería quiere controlar los cartones de bingo, Los mismos se encuentran cargados en un archivo de texto, el cual recibirá por parámetro. En cada línea del archivo contiene el número de cartón y 15 números separados por Tabs, cada 5 números representan una línea del cartón. El script al recibir la señal SIGUSR1 debe generar y mostrar por pantalla un número aleatorio del 0 al 99 (no debe repetir el número sorteado, ya que hay sólo una bolilla por número) y realizar el control de los cartones para mostrar cuándo un cartón obtuvo línea o bingo, mostrando también por pantalla los resultados. Al finalizar mostrará por pantalla un resumen de cantidad de bolillas sorteadas y los números de cartón ganadores de línea y bingo.
Debe omitir el CTRL + C y continuar ejecutando hasta encontrar el ganador.

Nota: Tal cual las reglas del bingo solo el primer cartón que complete línea será el ganador por línea y cuando se "cante" bingo el juego finaliza.

Cartones.txt

NumeroCarton	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	n11	n12	n13	n14	n15
1234	5	9	15	47	98	66	72	61	0	12	3	25	34	57	1
9999	10	78	45	75	0	26	47	68	81	3	12	55	69	1	33
1272	29	88	69	22	55	31	5	51	90	27	20	33	0	14	8

Criterios de corrección:

Control	Criticidad
El script ofrece ayuda con –h, -? o –help.	Obligatorio
Valida parámetros correctamente.	Obligatorio
Funciona correctamente según enunciado.	Obligatorio
Se adjuntan archivos de prueba por parte del grupo.	Obligatorio
Funciona con rutas relativas, absolutas o con espacios.	Obligatorio
Se implementan funciones	Opcional

Ejercicio 4:

Se desea realizar un script para controlar la similitud de un archivo base contra los archivos contenidos en un directorio. El script comparara el archivo pasado por parámetro contra cada uno de los archivos encontrados en una rama de directorios también pasado por parámetro, informando los archivos que tienen cierto porcentaje de igualdad. Si no se informa el directorio, tomará el directorio actual.
El script recibirá por parámetro el archivo base, la ruta de directorio (opcional) y un porcentaje mínimo de igualdad.

Criterios de corrección:

Control	Criticidad
El script ofrece ayuda con –h, -? o –help.	Obligatorio
Valida parámetros.	Obligatorio
Funciona correctamente según enunciado.	Obligatorio
Se adjuntan archivos de prueba por parte del grupo.	Obligatorio
Funciona con rutas relativas, absolutas o con espacios.	Obligatorio
Se implementan funciones	Deseable

Ejercicio 5:

Realizar un script que genere un backup comprimido de una ruta especificada.
El script recibirá los siguientes parámetros:

- Ruta de origen.
- -t para backupear todos los archivos o
- -x [extensión] para backupear los archivos con dicha extensión.

El nombre del archivo backup debe contener la fecha y hora de generación (al segundo) y en caso de ser por extensión, debe figurar la misma en el nombre.
Debe mantener los últimos 5 backups de cada tipo, eliminando el resto automáticamente y generar un archivo de log, con el mismo nombre del backup informando usuario de generación, fecha y hora y el listado de archivos resguardados para permitir una revisión rápida de que backup contiene cierto archivo sin tener que analizar los archivos comprimidos.

Criterios de corrección:

Control	Criticidad
El script ofrece ayuda con –h, -? o –help.	Obligatorio
Valida parámetros.	Obligatorio
Funciona correctamente según enunciado.	Obligatorio
Se adjuntan archivos de prueba por parte del grupo.	Obligatorio
Funciona con rutas relativas, absolutas o con espacios.	Obligatorio
Se implementan funciones	Deseable

Ejercicio 6:

Realizar un script que renombre masivamente los archivos de un directorio.
El mismo deberá tomar un patrón de búsqueda en el nombre de los archivos y un valor de reemplazo.
El script recibirá por parámetro el directorio a analizar, el patrón a buscar y el texto de reemplazo.

DirectorioA:
Pepexx.log
Pedroxx.log
Pacoxx.log

Llamada al script:
reemplazar.sh DirectorioA xx 20190328

Resultado:
Pepe20190328.log
Pedro20190328.log
Paco20190328.log

Criterios de corrección:

Control	Criticidad
El script ofrece ayuda con –h, -? o –help.	Obligatorio
Valida parámetros.	Obligatorio
Funciona correctamente según enunciado.	Obligatorio
Se adjuntan archivos de prueba por parte del grupo.	Obligatorio
Funciona con rutas relativas, absolutas o con espacios.	Obligatorio
Se implementan funciones	Deseable

Trabajo Práctico Nro. 2 (GRUPAL):

- ***Tema:*** Programación de scripts básicos en PowerShell
- ***Descripción:*** Se programarán todos los scripts mencionados en el presente trabajo, teniendo especialmente en cuenta las recomendaciones sobre programación mencionadas en la introducción de este trabajo. Los contenidos de este trabajo práctico pueden ser incluidos como tema de parciales.
- ***Formato de entrega:*** Según el protocolo especificado anteriormente. Recomendamos realizar la entrega presencial
- ***Documentación:*** Todos los scripts que se entreguen deben tener un encabezado y un fin de archivo. Dentro del encabezado deben figurar el nombre del script, 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:*** **27/05/2019 o 28/05/2019**, dependiendo del día que se curse la materia.

Introducción:

La finalidad del presente práctico es que los alumnos adquieran un cierto entrenamiento sobre la programación de shell scripts en lenguaje PowerSell. Todos los scripts están orientados a la administración de una máquina, o red y pueden ser interrelacionados de tal manera de darles una funcionalidad real.
Cabe destacar que todos los scripts deben poder ser ejecutados en forma batch o interactiva, y que en ningún caso serán probados con el usuario administrator, por lo cual deben tener en cuenta al realizarlos, no intentar utilizar comandos, directorios, u otros recursos que solo están disponibles para dicho usuario, o usuarios del grupo. Todos los scripts deberán funcionar en las instalaciones del laboratorio 266, ya que es ahí donde serán controlados por el grupo docente.
Para un correcto y uniforme funcionamiento, se deberán respetar algunos lineamientos generales:

1. Modularidad:

Si bien es algo subjetivo del programador, se trata de privilegiar la utilización de funciones (internas / externas), para lograr una integración posterior menos trabajosa.

2. Claridad:

Se recomienda fuertemente el uso de comentarios que permitan la máxima legibilidad de los scripts.

3. Verificación:

Todos los scripts deben realizar un control de las opciones que se le indiquen por línea de comando, es decir, verificar la sintaxis de la misma. En caso de error u omisión de opciones, al estilo de la mayoría de los comandos deben indicar mensajes como:

```
"Error en llamada!
 Uso: comando [...]"
```

Donde se indicará entre corchetes [ ], los parámetros opcionales; y sin ellos los obligatorios, dando una breve explicación de cada uno de ellos.

Todos los scripts deben incluir una opción standard '-?' que indique el número de versión y las formas de llamada.

Ejercicios:

IMPORTANTE: la versión mínima a utilizar para confeccionar y evaluar este TP debe ser Powershell 6, ya que es la versión de Powershell Core donde se realizará la corrección

Ejercicio 1:

En base al siguiente código, responder:

```
Param($pathsalida)
$existe = Test-Path $pathsalida
if ($existe -eq $true) {
    $lista = Get-ChildItem -File
    foreach ($item in $lista) {
        Write-Host "$($item.Name) $($item.Length)"
}
} else {
    Write-Error "El path no existe"
}
```

Responda:

a) ¿Cuál es el objetivo del script?

b) ¿Qué validaciones agregaría a la definición de parámetros?

c) ¿Con qué cmdlet se podría reemplazar el script para mostrar una salida similar?

Ejercicio 2:

Desarrollar un script que cuenta la cantidad de archivo duplicados que hay en un directorio. Se considera un archivo duplicado a aquellos que tienen el mismo nombre y peso, sin importar su contenido. El script debe recibir un solo parámetro llamado "path" que debe indicar el directorio a analizar, el cual se debe recorrer recursivamente.

La salida del script debe ser un listado solo con los nombres de los archivos duplicados (sin paths y una sola vez por grupo de archivos duplicados).

Criterios de corrección:

Control	Criticidad
Debe cumplir con el enunciado	Obligatorio
El script cuenta con una ayuda visible con Get-Help	Obligatorio
Validación correcta de los parámetros	Obligatorio
El directorio se recorre recursivamente	Obligatorio
Se deben usar hash-tables (arrays asociativos)	Obligatorio

Ejercicio 3:

Se tiene un archivo CSV con los datos de las multas reportadas en el partido de La Matanza. Dicho archivo contiene la patente, el monto y la fecha en la que se realizó la multa. Desarrollar un script que permita resumir estos datos por año y por patente. El script debe recibir dos parámetros:

- -Entrada: Path del archivo CSV de entrada.
- -Salida: Path del archivo CSV de salida (resumen).

Formato del archivo CSV de entrada

```
"Patente","ValorMulta","Fecha"
"AAA000","1000","01/01/2015"
"AAA001","10300","21/01/2015"
"AAA000","5000","01/07/2017"
"AA222BB","1000","05/02/2019"
"AA222BB","1000","18/04/2019"
"GRS580","1000","01/12/2019"
"NYF445","6000","25/08/2015"
"AAA000","1000","04/02/2015"
```

Formato del archivo CSV de salida:

```
"Patente","Año","TotalMultas"
"AAA000","2015","2000"
"AAA000","2017","5000"
```

```
“AAA001”,”2015”,”10300”
“AA222BB”,”2019”,”2000”
“GRS580”,”2019”,”1000”
“NYF445”,”2015”,”6000”
```

Criterios de corrección:

Control	Criticidad
Debe cumplir con el enunciado	Obligatorio
El script cuenta con una ayuda visible con Get-Help	Obligatorio
Validación correcta de los parámetros	Obligatorio
El archivo CSV de salida debe estar ordenado por patente de manera ascendente	Obligatorio
Se deben usar los comandos Import-CSV y Export-CSV	Obligatorio
El separador de los archivos CSV debe ser “,”	Obligatorio
Se deben respetar los nombres de los parámetros y de las columnas del CSV, tanto de entrada como de salida	Obligatorio

Ejercicio 4:

Un directorio cuenta con cientos de archivos de logs de errores. Desarrollar una script que cree un archivo .ZIP que contenga todos los archivos en cuyo contenido se encuentre la cadena pasada por parámetro.

Parámetros:

- -PathLogs: Path del directorio que contiene los archivos de log.
- -PathSalida: Path del archivo ZIP generado por el script. El parámetro debe contener el nombre del archivo.
- -Cadena: Cadena a buscar dentro de los archivos.

Criterios de corrección:

Control	Criticidad
Debe cumplir con el enunciado	Obligatorio
El script cuenta con una ayuda visible con Get-Help	Obligatorio
Validación correcta de los parámetros	Obligatorio
Los nombres de los parámetros deben ser los indicados en el enunciado	Obligatorio
Se debe usar la clase ZipFile para la compresión de los archivos	Obligatorio
Los archivos a evaluar no tienen que estar limitados por extensión	Obligatorio

Ejercicio 5:

Desarrollar un script que cada una determinada cantidad de tiempo realice una de las siguientes acciones:

- Informar la cantidad de procesos que se encuentran corriendo en ese momento.
- Indicar la cantidad de archivos que contiene un directorio.

El script puede recibir los siguientes parámetros:

- -Procesos: Parámetro de tipo switch que indica que se mostrará la cantidad de procesos corriendo al momento de ejecutar el script.
- -Archivos: Parámetro de tipo switch que indica que se mostrará la cantidad de archivos que contiene un directorio.
- -Directorio: Solo se puede usar si se pasó “-Archivos”. Indica el directorio a evaluar.
- Importante: “-Procesos” y “-Archivos” no se pueden usar al mismo tiempo. “-Directorio” se puede usar solamente si “-Archivos” se encuentra presente.

Criterios de corrección:

Control	Criticidad
Debe cumplir con el enunciado	Obligatorio
El script cuenta con una ayuda visible con Get-Help	Obligatorio
Validación correcta de los parámetros	Obligatorio
Se deben respetar los nombres y tipos de los parámetros	Obligatorio
Crear grupos de parámetros (ParameterSetName)	Obligatorio
La salida debe ser únicamente un número	Obligatorio

Ejercicio 6:

Desarrollar un script que permita realizar producto escalar y trasposición de matrices. La entrada de la matriz al script se realizará mediante un archivo de texto plano. La salida se guardará en otro archivo que se llamará "salida.nombreArchivoEntrada". El formato de los archivos de matrices debe ser el siguiente:

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

Parámetros que recibirá el script:

- -Entrada: Path del archivo de entrada. No se debe realizar validación por extensión de archivo. Se asume que todos los archivos de entrada tienen matrices válidas.
- -Producto: De tipo entero, recibe el escalar a ser utilizado en el producto escalar. No se puede usar junto con "-Trasponer".
- -Trasponer: De tipo switch, traspone la matriz. No se puede usar junto con "-Producto".

Criterios de corrección:

Control	Criticidad
Debe cumplir con el enunciado	Obligatorio
El script cuenta con una ayuda visible con Get-Help	Obligatorio
Validación correcta de los parámetros	Obligatorio
Se deben respetar los nombres y tipos de los parámetros	Obligatorio
Crear grupos de parámetros (ParameterSetName)	Obligatorio
Se debe respetar el nombre del archivo de salida	Obligatorio
La matriz puede contener cualquier "double" válido	Obligatorio

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:*** **01/07/2019 o 02/07/2019**, dependiendo del día que se curse la materia.

Ejercicio 1:

Crear un proceso que, dependiendo de lo informado por parámetro, realice lo siguiente:

`-d` cree dos procesos demonios, que deberán simplemente ejecutar cada 2 segundos un loop hasta 1.000.000 realizando una acumulación en una variable, y no finalizan (simplemente para que generen algo de procesamiento).
`-h` cree 10 procesos hijos, donde cada uno realiza el mismo procesamiento que en el caso anterior, el padre se quedará esperando a que se presione una tecla y en este momento se cierran todos los procesos.
`-z` cree 6 procesos nietos, de dos hijos distintos, y 2 de ellos deberán quedar como zombies. El padre se quedará esperando a que se presione una tecla para poder verificar la correcta generación de los procesos, y luego cierra los procesos.

Cada proceso deberá mostrar por pantalla la siguiente información:
`PID: pppp PPID: nnnn Parentesco-Tipo: [hijo/nieto]-[normal/demonio/zombie]`

Criterios de corrección:

Control	Criticidad
Compila sin errores con el script entregado	Obligatorio
Funciona correctamente según enunciado	Obligatorio
Existe algún tipo de ayuda para la ejecución del proceso	Obligatorio

Ejercicio 2:

Una empresa recibe en la administración central un archivo de stock de productos de cada una de sus sucursales de venta al finalizar el día. Se necesita conocer el stock consolidado de la empresa para saber si se tiene que reponer algún producto para mantener el stock de venta.

Como son varias sucursales se evaluó que la mejor forma de hacerlo es procesar con varios threads (en cantidad a informar por parámetro) que se repartan el trabajo de procesamiento de los archivos, en la forma más equitativa posible.

Se cuenta con un archivo con el listado maestro de artículos, donde se indica para cada uno el stock mínimo que debe haber en la empresa. Si el stock está por debajo de este número, el proceso deberá generar un registro de pedido de compra informando la cantidad necesaria a comprar para que el stock quede al doble del stock mínimo del artículo.

Ej, Aceite, stock mínimo 100, stock actual 73, entonces genera un pedido por 127.

El proceso deberá tomar por parámetro la cantidad de threads de procesamiento, el directorio donde se encuentran los archivos de las sucursales y el archivo del maestro de productos; y deberá generar como salida el archivo de stock consolidado y el archivo de pedido de compras, más un archivo de log que indique qué archivos procesó cada thread.

Se entrega un juego de archivos para poder probar la ejecución con un volumen de datos acorde para poder visualizar la ejecución de los threads, que podrán descargar desde el portal de Sisop en "Material Adicional / Guías y Ejercicios" o desde aquí:

http://sisop.com.ar/files/catedra/apuntes/ejercicios/TP3_Ej2.zip

Formato de los archivos, tienen campos de ancho fijo:

Maestro de artículos

ID	numérico	8 dígitos
Descripcion	string	40 caracteres
Stock_minimo	numérico	5 dígitos

Stock de sucursales

ID	numérico	8 dígitos
Stock_actual	numérico	5 dígitos

El archivo de sucursal lleva el nombre de la misma y la fecha de generación: SanJusto_20190401.txt

NOTA: No copiar los archivos de ejemplo en la entrega del ejercicio.

Criterios de corrección:

Control	Criticidad
Compila sin errores con el script entregado	Obligatorio
Funciona correctamente según enunciado	Obligatorio
Existe algún tipo de ayuda para la ejecución del proceso	Obligatorio
Valida correctamente los parámetros	Obligatorio
Acepta correctamente paths absolutos y relativos	Obligatorio
No se aceptan soluciones que utilicen system() o popen()	Obligatorio

Ejercicio 3:

El gobierno de la provincia de Buenos Aires coloco muchas cámaras para detectar los excesos de velocidad. Las cámaras están colocadas en zonas de circulación de máxima 60.
Estas cámaras envían la patente y la velocidad a la que circulaba el automóvil a un sistema centralizado, que está ejecutando como demonio, a través de una estructura FIFO.
El sistema debe mostrar por pantalla la patente, la velocidad y la cámara informó el paso de un auto.
A su vez se deben registrar el movimiento de todos los autos y guardarlos en un archivo llamado "Transito_AAAAMMDD.txt", el cual debe informar los datos.
Ejemplo:

```
HJP123 Cam1 45 km/h
PLI584 Cam3 65 km/h
AA530BA Cam5 59 km/h
LSK768 Cam1 54 km/h
```

Al finalizar cada día se desea guardar en un archivo “Crear_Multas_AAAAMMDD.txt” la patente de los autos en exceso seguidos de la hora, la velocidad y el importe de la multa. Las multas se calculan $1500 multiplicado por cada unidad de velocidad superior a la máxima. (Si un auto va a 80 y la máxima es 60 se debe cobrar $1500*20 = $30000).

Ejemplo:

```
YJP544 12:30 12/2/2019 66 km/h $9000
PLI584 13:23 12/2/2019 65 km/h $7500
VGT777 13:30 12/2/2019 61 km/h $1500
LLK677 14:00 12/2/2019 100 km/h $60000
```

No se debe programar el proceso de las cámaras para generación de las patentes, este es un sistema externo que se comunica por el FIFO. Sólo se debe generar el sistema central, que debe tomar por parámetro el nombre del fifo.

Criterios de corrección:

Control	Criticidad
Compila sin errores con el script entregado	Obligatorio
Funciona correctamente según enunciado	Obligatorio
Existe algún tipo de ayuda para la ejecución del proceso	Obligatorio
Puede haber más de un proceso de cámara ejecutando a la vez	Obligatorio
Ejecuta como demonio	Obligatorio
Acepta correctamente paths absolutos y relativos	Obligatorio
No se aceptan soluciones que utilicen con system() o popen()	Obligatorio

Ejercicio 4:

El mismo gobierno quiere realizar un sistema centralizado con una base de datos para los infractores en el cual se accede colocando el partido como parámetro. La base de datos contendrá Partido, Patente, Nombre del titular, Cantidad de multas y Monto total a pagar. Tener en cuenta que el usuario no debe visualizar los datos de otro partido distinto al que pertenece.

Este sistema debe poder:

- Ingresar multas nuevas, indicando la patente. En caso de que la patente no este se debe crear un nuevo elemento en la base de datos. En caso de que este, debe aumentar el número de multas y se le debe sumar el nuevo monto al monto a pagar.
- Buscar lista de registros a suspender. Los registros a suspender son aquellos de las personas que deben un monto total mayor a $20.000 y/o que poseen más de 3 multas.
- Cancelar multas, es decir saldar las multas y borrar a las personas que pagaron.
- Buscar el monto a pagar de una patente determinada
- Ver el monto total que deben pagar todos los infractores.

Para esto se solicita que creen un servidor que atienda las solicitudes generadas desde un proceso cliente que se le entrega a cada municipalidad, que están conectadas por una red privada (VPN) a la central de La Plata. Al ingresar al sistema el usuario informa por parámetro el partido a que pertenece y la dirección del servidor.

No es necesario controlar duplicidad de registros ni consistencia de los datos de la base, como tampoco la identificación o login del usuario.

Criterios de corrección:

Control	Criticidad
Compila sin errores con el script entregado	Obligatorio
Funciona correctamente según enunciado	Obligatorio
Existe algún tipo de ayuda para la ejecución del proceso	Obligatorio
Acepta correctamente paths absolutos y relativos	Obligatorio
No se aceptan soluciones que utilicen con system() o popen()	Obligatorio
Funciona tanto localmente como a través de la red desde distintas máquinas	Obligatorio
Utiliza sockets	Obligatorio
Soporta múltiples clientes (se probará con más de 5)	Obligatorio
Maneja correctamente el cierre de los puertos al finalizar	Deseable

Tiene archivo de configuración (deben entregarlo)	Deseable
Se conecta con nombre de máquinas	Deseable

Ejercicio 5:

Debido a la mala señal de internet de un partido, se optó por realizar una versión del sistema anterior, pero a nivel Local utilizando memoria compartida.

Criterios de corrección:

Control	Criticidad
Compila sin errores con el script entregado	Obligatorio
Funciona correctamente según enunciado	Obligatorio
Existe algún tipo de ayuda para la ejecución del proceso	Obligatorio
Acepta correctamente paths absolutos y relativos	Obligatorio
No se aceptan soluciones que utilicen con system() o popen()	Obligatorio
Utiliza memoria compartida	Obligatorio
Soporta múltiples clientes (se probará con más de 5)	Obligatorio
Maneja correctamente la eliminación de los recursos al finalizar	Obligatorio
Tiene archivo de configuración (deben entregarlo)	Deseable