Ejercicios BLJ
Acceso a información del estado del S. O. desde programas de usuario.

OBJETIVO.
Conocer que a través de los APIs (funciones, métodos, clases) de los lenguajes de programación (C++, Java), se puede obtener información y / o llevar a cabo funciones como las realizadas por los comandos del S.O.

INDICACIONES SOBRE EL DESARROLLO
En todos los puntos que sigue tendrá que explicar con detalle cómo logró llevar a cabo lo pedido, indicando trayectorias, comandos con su despliegue y / o acciones realizadas. En el caso de despliegue de comandos explique el significado de lo desplegado. Cuando haya preguntas específicas, no basta con aplicar y desplegar el resultado del comando, deberá usted contestarlas. Tendrá que abrir una terminal-ventana, en Ubuntu, y también puede ayudarse del File Manager.

PREVIO AL INICIO
- Copie el contenido del fólder *matBLI/ediini* de Sistemas Operativos (SO) en Comunidad. Este fólder contiene tanto el texto de la práctica como material de soporte.
- Después arranque la máquina virtual de Ubuntu y entre a la cuenta de "sisops".

1) Dentro del fólder ***edini*** se encuentran una serie de archivos en C++ (extensión .cc) y Java (extensión .java) que deberá utilizar para realizar los ejercicios. A continuación, se muestran diferentes formas de compilación.
   Para compilar y ejecutar en C++.
   a) **g++ *prog.cc*** : el ejecutable es dejado en ***a.out***
   b) **g++ *prog.cc -o proeje.exe*** : el ejecutable es dejado en ***proeje.exe***
   Para compilar y ejecutar en java.
   c) Se requiere, para poder usar los programas ejecutables de su directorio, debe tener el archivo ***.mydot***, en su *home directory*, conteniendo el comando **setenv PATH ".:$PATH"**; OBSERVACIÓN. Recuerde ejecutar ***source ~/.mydot***.
   d) **javac *prog.java***: el ejecutable es dejado en ***prog.class***
   e) **java *prog***: ejecución de ***prog.class***

2) * Los archivos ***argumentos.cc*** y ***argumentos.java*** permiten introducir cadenas de caracteres, como argumentos, al momento de mandar ejecutar el programa ejecutable. Compile y ejecute ambos programas con las cadenas de caracteres ***uno 2 tres 4***, mostrando como llevó a cabo la compilación y la ejecución. En cada programa, describa la parte del programa que maneja dichas cadenas de caracteres. _____
   En cada caso, en CC y en Java, ¿cuál fue el comando completo que aplicó?
   C++p

```
ubuntu:~/SO/matBLJ/ediini> g++ argumentos.cc -o argumentos.exe
ubuntu:~/SO/matBLJ/ediini> argumentos.exe uno 2 tres 4
#arg=5
argv[0] = argumentos.exe
argv[1] = uno
argv[2] = 2
argv[3] = tres
argv[4] = 4
```

java

```
ubuntu:~/SO/matBLJ/ediini> javac argumentos.java
ubuntu:~/SO/matBLJ/ediini> java argumentos uno 2 tres 4
#argumentos= 4
argv[0]=uno
argv[1]=2
argv[2]=tres
argv[3]=4
```

Hay alguna diferencia, en lo que respecta a los argumentos, entre lo que se ve desde CC y lo que se ve desde Java. De haber diferencias ¿Por qué las hay?
Sí hay diferencias. Es por la forma en que se reciben los argumentos en el main. En C++ el primer parámetro es el nombre del ejecutable y en java solo se reciben los argumentos del ejecutable. En el caso de Java omitan lo que aparece desplegado entre ( ).

3) * El comando ***printenv*** nos entrega la lista de las variables del medio ambiente de un *shell*. Ejecute ***printenv*** para ver el despliegue. Los archivos ***convarso.cc*** y ***convarso.java, convarsol.java*** obtienen también una lista de variables similar, sobre todo en el caso de C++. En el caso de la Máquina Virtual de Java, mostrará sus propias variables de medio ambiente; ver archivo adjunto *JavaSystemProp.docx*. Compile y ejecute ambos programas. En cada programa describa la parte principal con la que se obtienen dichas variables de ambiente, ya sea por una variable, por una función o por un g++método. _____
En el programa de C++ se agrega un parámetro al método main llamado envp, con ello se reciben todas las variables de ambiente.

```
ubuntu:~/SO/matBLJ/ediini> convarso.exe
envp[0]: XDG_CONFIG_DIRS=/etc/xdg/xdg-ubuntu:/etc/xdg
envp[1]: LANG=en_US.UTF-8
envp[2]: DISPLAY=:0
envp[3]: SHLVL=1
envp[4]: LOGNAME=sisops
envp[5]: XDG_VTNR=2
envp[6]: PWD=/home/sisops/SO/matBLJ/ediini
envp[7]: XAUTHORITY=/run/user/1000/gdm/Xauthority
envp[8]: GTK_IM_MODULE=ibus
envp[9]: COLORTERM=truecolor
envp[10]: XDG_SESSION_ID=2
envp[11]: GNOME_TERMINAL_SCREEN=/org/gnome/Terminal/screen/cf9001ff_f0
37_4cc8_a924_d0dc91cefdcf
envp[12]: IM_CONFIG_PHASE=2
envp[13]: XDG_SESSION_DESKTOP=ubuntu
envp[14]: GDMSESSION=ubuntu
envp[15]: TEXTDOMAINDIR=/usr/share/locale/
envp[16]: USERNAME=sisops
envp[17]: GNOME_DESKTOP_SESSION_ID=this-is-deprecated
envp[18]: WINDOWPATH=2
envp[19]: TEXTDOMAIN=im-config
```

En el programa de java se usa la función System.getProperties(), la cual obtiene todas las variables e la JVM.

```
ubuntu:~/SO/matBLJ/ediini> javac convarso.java
ubuntu:~/SO/matBLJ/ediini> conva
convarso.cc    convarso.exe   convarso.java  convarsol.java
ubuntu:~/SO/matBLJ/ediini> java con
convarso.cc*    convarso.exe*   convarsol.java*
convarso.class  convarso.java*
ubuntu:~/SO/matBLJ/ediini> java convarso
-- listing properties --
sun.desktop=gnome
awt.toolkit=sun.awt.X11.XToolkit
java.specification.version=11
sun.cpu.isalist=
sun.jnu.encoding=UTF-8
java.class.path=.
java.vm.vendor=Ubuntu
sun.arch.data.model=64
java.vendor.url=https://ubuntu.com/
user.timezone=
os.name=Linux
```

4) * Los archivos ***DatosUsu.cc*** y ***DatosUsu.java*** obtienen de manera parcial algunas de las variables del medio ambiente referente al usuario, como el equivalente a los comandos ***printenv USER***, ***printenv HOME*** y ***printenv PWD***; o también con ***echo $USER***, y similares. Compile y ejecute ambos programas. En cada programa diga con cuál función o método se obtuvo las variables de ambiente.
C++ se utiliza el comando getenv("comandoShell")
Java se utiliza System.getProperty("user.name/home/dir")

5) * Los archivos ***NomCom.cc*** y ***NomCom.java*** obtienen el nombre local de la computadora al igual que con el comando ***hostname***. Compile y ejecute ambos programas. En cada programa diga con cuál función o método se obtuvo el nombre de la computadora.

```
ubuntu:~/SO/matBLJ/ediini> g++ NomCom.cc -o NomCom.exe
ubuntu:~/SO/matBLJ/ediini> NomC
NomCom.cc   NomCom.exe  NomCom.java
ubuntu:~/SO/matBLJ/ediini> NomCom.exe

  Nombre de esta computadora: ubuntu

ubuntu:~/SO/matBLJ/ediini> javac Nom
NomCom.cc*   NomCom.exe*  NomCom.java*
ubuntu:~/SO/matBLJ/ediini> javac NomCom.java
ubuntu:~/SO/matBLJ/ediini> java NomCom

  Nombre de esta computadora: ubuntu
```

En C++ se necesita la librería unistd.h y con ella podemos llamar el método gethostname. El método recibe un apuntador a un arreglo de caracteres, donde se guardará el nombre del host, y el tamaño del arreglo. Regresa un entero donde se guarda el exit status code, y podemos notar que si es menor que 0 es porque hubo un error durante la ejecución.

En java se usa la librería java.net.*, la cual nos da acceso a la clase InetAddress y al método getLocalHost que nos regresa un objeto de tipo InetAddress y a partir de este podemos obtener el nombre del host con el método getHostName. Lanza excepción si no encuentra el host.

6) * El archivo ***ppiduid.cc*** obtiene en una primera instancia tanto el *pid* (process id) del proceso como su respectivo *ppid* (parente process ID) tal y como lo desplegaría el comando ***ps –l***. En segunda instancia obtiene tanto el número de cuenta (*uid*) como el grupo (*gid*) sobre el que se está ejecutando el proceso, tal y como lo muestra el comando ***id***. Compile y ejecute. Indique la función con la que se obtuvo cada valor (*pid, ppid, uid, gid*).

```
ubuntu:~/Desktop/ediini> g++ ppiduid.cc
ubuntu:~/Desktop/ediini> a.out

Process ID (PID)= 12694
Parent Process ID (PPID)= 2300

Real User ID (uid)= 1000
Real Group ID (gid)= 1000
```

Con las siguientes instrucciones:

```
ubuntu:~/Desktop/ediini> cat ppiduid.cc
// ppiduid.cc

#include <stdio.h>
#include <unistd.h>

int main()
{

  printf("\nProcess ID (PID)= %d\n", getpid() );
  printf("Parent Process ID (PPID)= %d\n", getppid() );

  // Parcialmente parecido al comando "id"
  printf("\nReal User ID (uid)= %d\n", getuid() );
  printf("Real Group ID (gid)= %d\n\n", getgid() );

  return 0;
}
```

En los siguientes ejercicios (del 7 al 10) se tiene que completarlos con programación, y mostrárselos al profesor.

7) ** Estudie, ejecute y compile los archivos ***DatosSis.cc*** y ***DatosSis.java***. Copie el archivo ***DatosSis.cc*** en ***DatosSis2.cc*** el cual deberá modificar para que ahora la variable de medio ambiente se pase como argumento al arrancar el programa, pudiendo entonces dar cualquiera de las variables. En lugar de tres variables de medio ambiente, sólo maneje una. Debe quedar protegido por si no se le pasa ningún argumento. Con el comando ***printenv*** se pueden listar las variables del medio ambiente. Muestre el código y la corrida de ***DatosSis2.cc***.

```
ubuntu:~/matBLJ/ediini> g++ DatosSis2.cc -o DatosSis2Eje.exe
ubuntu:~/matBLJ/ediini> DatosSis2Eje.exe LOGNAM
No existe la variableubuntu:~/matBLJ/edi LOGNAME
Variable pedida LOGNAME: sisops
ubuntu:~/matBLJ/ediini> g++ DatosSis2.cc -o DatosSis2Eje.exe
ubuntu:~/matBLJ/ediini> DatosSis2Eje.exe LOGNAM
No existe la variable
ubuntu:~/matBLJ/ediini> DatosSis2Eje.exe LOGNAME
Variable pedida LOGNAME: sisops
ubuntu:~/matBLJ/ediini> DatosSis2Eje.exe PATH
Variable pedida PATH: .:.:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sb
in:/bin:/usr/games:/usr/local/games:/snap/bin
ubuntu:~/matBLJ/ediini>
```

8) ** Vuelva a modificar ***DatosSis2.cc*** para que marque error en caso de que el argumento pasado no corresponda a ninguna variable de ambiente. La función *getenv(varamb)*, en caso de que la variable de

ambiente no exista, devuelve el apuntador a nulo o (char *) 0. Puede ver el archivo ***convarso.cc***, para ver como preguntar por nulo.

```
ubuntu:~/matBLJ/ediini> DatosSis2Eje.exe PATH
Variable pedida PATH:  .:.:/usr/local/sbin:/usr/local/
in:/bin:/usr/games:/usr/local/games:/snap/bin
ubuntu:~/matBLJ/ediini> DatosSis2Eje.exe LOGNA
No existe la variable
ubuntu:~/matBLJ/ediini> DatosSis2Eje.exe LOGNAm
No existe la variable
ubuntu:~/matBLJ/ediini>
```

9) ** Estudie, compile y ejecute los programas ***Fecha.cc*** y ***Fecha.java***.
Fecha.java hace uso de la clase Date para obtener la fecha y tiempo del día.
Fecha.cc hace uso de las funciones "time" y "ctime". "time" devuelve una medida en segundos, que representa la fecha y tiempo del día, de tipo "time_t". "ctime" convierte un valor tipo "time_t" en una cadena de caracteres. El tipo "time_t" es equivalente al tipo "long", por lo tanto en lugar del especificador "%d" deberá usar "%ld". _____

```c
// Fecha.cc

#include <stdio.h>
#include <time.h>

int main()
{
    time_t a;
    char *st1;

    time( &a );
    printf("a= %ld seg.\n",a);

    // Similar al comando "date"
    st1 = ctime( &a );
    printf("st1= %s\n",st1);

    return 0;
}
```

Se debe cambiar la instrucción en el print manualmente o con la instrucción nano en terminal.

```
ubuntu:~/Desktop/ediini> g++ Fecha.cc
ubuntu:~/Desktop/ediini> a.out
a= 1650485818 seg.
st1= Wed Apr 20 13:16:58 2022
```

Así ya imprime correctamente.

```
ubuntu:~/Desktop/ediini> javac Fecha.java
ubuntu:~/Desktop/ediini> java Fecha
Fecha: Wed Apr 20 13:19:19 PDT 2022
```

Con Java no hay inconveniente.

10) ** Estudie, compile y ejecute los programas ***FechaG.cc*** y ***FechaG.java***.
FechaG.java hace uso de la clase *GregorianCalendar* para obtener diferentes características de fecha y tiempo del día. Aquí se tienen expresión de fechas en milisegundo.
FechaG.cc hace uso de la función *localtime* para obtener diferentes características de fecha y tiempo del día. Aquí sólo se pueden tener mediciones en segundos, no hay para milisegundos.
Modifique ambos archivos, para que entre la 1era. y 2da. medición de tiempo que hacen los programas, se pongan a dormir cuatro segundos. Al final deberá calcular e imprimir la diferencia entre la primera y segunda medición. En el caso de CC++ esta diferencia se hará en segundos mientras que en JAVA se expresará en milisegundos.

En el caso de ***FechaG.cc*** hará uso de la función "sleep(segundos)", que se asocia con el encabezado “unistd.h”.

En el caso de ***FechaG.java*** hará uso de la función "Thread.sleep(milisegundos)", que se usa de la siguiente forma:

```
try
{  Thread.sleep(milisegundos); }
catch (InterruptedException e)
{ }
```

Tenga cuidado al transformar horas, minutos y segundos a milisegundos, en el caso de Java. Como información hay un método que arroja milisegundos directamente: *System.currentTimeMillis()*.

Muestre el código y la corrida de ambos programas.
C++

```
ubuntu:~/SO/matBLJ/ediini> FechaG.exe
a= 1650913908 seg.
st1= Mon Apr 25 12:11:48 2022

HOUR= 12      MINUTE= 11      SECOND= 48
DAY OF MONTH= 25      MONTH= 4      YEAR= 2022
WEEK DAY= 1      YEAR DAY= 115
------------------------------
------------------------------
------------------------------
a= 1650913912 seg.
st1= Mon Apr 25 12:11:52 2022

HOUR= 12      MINUTE= 11      SECOND= 52
DAY OF MONTH= 25      MONTH= 4      YEAR= 2022
WEEK DAY= 1      YEAR DAY= 115
Diferencia en segundos: 4
```

```
int ini = vtm->tm_sec;
printf( "------------------------------\n");
printf( "------------------------------\n");
printf( "------------------------------\n");
sleep(4);

time( &a );
printf( "a= %d seg.\n", a);

// Similar al comando "date"
st1 = ctime( &a );
printf( "st1= %s\n", st1);

vtm = localtime( &a );
aux();
int fin = vtm->tm_sec-ini;
printf("Diferencia en segundos: %i\n", fin);
```

Java

```
double ini = System.currentTimeMillis();
 try
{
      Thread.sleep(4000);
}
catch (InterruptedException e)
{
      System.out.println("There was an exception: "+e.toString());
}

System.out.println("----------------------------");
System.out.println("----------------------------");
System.out.println("----------------------------");

calendar = new GregorianCalendar();
System.out.println("HOY= " + (new Date()).toString() );
aux();
double fin = System.currentTimeMillis() - ini;
System.out.println("La diferencia en milisegundos fue de " + fin);
```

```
ubuntu:~/SO/matBLJ/ediini> java FechaG
HOY= Mon Apr 25 12:12:25 PDT 2022
DATE: 25        MONTH: 4        YEAR: 2022
WEEK_OF_YEAR: 18        WEEK_OF_MONTH: 5
DAY_OF_MONTH: 25        DAY_OF_YEAR: 115        DAY_OF_WEEK: 2
HOUR: 0        AM_PM: 1
HOUR_OF_DAY: 12
MINUTE: 12        SECOND: 25        MILLISECOND: 618
--------------------------
--------------------------
--------------------------
HOY= Mon Apr 25 12:12:29 PDT 2022
DATE: 25        MONTH: 4        YEAR: 2022
WEEK_OF_YEAR: 18        WEEK_OF_MONTH: 5
DAY_OF_MONTH: 25        DAY_OF_YEAR: 115        DAY_OF_WEEK: 2
HOUR: 0        AM_PM: 1
HOUR_OF_DAY: 12
MINUTE: 12        SECOND: 29        MILLISECOND: 691
La diferencia en milisegundos fue de 4004.0
```

11) ¿** Elabore ahora un programa C++ que calcule la diferencia entre dos mediciones de tiempo empleando ahora la función *clock( )*, en microsegundos. Busque y entienda el funcionamiento de *clock( )*. El consumo de tiempo tendrá que llevarla a cabo con un ciclo, ya que no se puede usar *sleep( )*. La función *clock* hace uso del encabezado "time.h" y su resultado lo devuelve en tipo *clock_t*, similar a *time_t*. ¿Esta función que valor es el que devuelve? ¿En segundos o alguna otra unidad de tiempo? La diferencia de tiempo, en esta función, sólo incluye tiempo de procesamiento. _____

```c
/* clock example: frequency of primes */
#include <stdio.h>      /* printf */
#include <time.h>       /* clock_t, clock, CLOCKS_PER_SEC */
#include <math.h>       /* sqrt */

int main ()
{
  clock_t t;
  t = clock();
  t = clock() - t;
  printf ("It took me (%f seconds).\n",t,((float)t)/CLOCKS_PER_SEC);
  return 0;
}
```

```
ubuntu:~/Desktop/ediini> source ~/.mydot
ubuntu:~/Desktop/ediini> tiempo.exe
It took me (0.000001 seconds).
ubuntu:~/Desktop/ediini>
```