**Ejercicios BI04**

**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 *matBI* 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”.

**ACTIVIDADES INICIALES**

Después dar *login*, notará que su cuenta está preparada para que cómo *shell* lo atienda *tcsh*.

En su directorio base (*home*), copie el archivo ***upsc1tar*** del URL ***ftp://ftp.itam.mx/pub/investigadores/rrios/SisOpe/Prac/ejerBI04/***. Esto lo puede hacer, ya sea desde su browser, o desde su computadora usando el comando *wget*. Cuando use ***wget*** incluya en el URL, al final, el archivo ***upsc1tar***. Una vez en el directorio base aplique el comando ***tar –xvf upsc1tar*** que descomprimirá el archivo ***upsc1tar*** en su directorio base creando el directorio (folder) ***upsc1*** que contiene a su vez varios directorios y archivos; ***upsc1*** contiene archivos en C++ y Java.

**DESARROLLO**

Un (\*\*) indica que, en la pregunta, hay algo que desarrollar, ejecutar, describir y / o mostrar.

Esta práctica se calificará durante la sesión.

Ver los APIs Headers de C y C++ en el archivo adjunto.

Con el comando *man* puede encontrar información sobre las funciones de CC++.

1. Dentro del directorio ***upsc1*** 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++.

1. **g++  *prog.cc*** : el ejecutable es dejado en ***a.out***
2. **g++  *prog.cc -o proeje.exe*** : el ejecutable es dejado en ***proeje.exe***

Para compilar y ejecutar en java.

1. Se requiere, para poder usar los programas ejecutables de su directorio, debe tener el archivo ***.midir***, en su *home directory*, conteniendo el comando **setenv PATH “.:$PATH”**; OBSERVACIÓN. Recuerde ejecutar ***source ~/.midir***.
2. **javac *prog.java***: el ejecutable es dejado en ***prog.class***
3. **java *prog***: ejecución de ***prog.class***
4. \*\* 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?

CC:

g++ argumentos.cc -o argumentos.exe

./argumentos.exe uno 2 tres 4

Java:

javac argumentos.java

java argumentos uno 2 tres 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? En CC hay un argumento más que en Java. El argumento adicional es el comando de ejecución del archivo (es este caso, ./argumentos.exe). En el caso de Java omitan lo que aparece desplegado entre ( ).

1. \*\* 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, de una función o un método.

En el caso del archivo en C++, la variable envp es un arreglo que contiene todas las variables de ambiente y se pasa como parámetro a la función main.

En el caso de Java, el objeto System, que representa todo el sistema, se utiliza para extraer las “propiedades” (que serán las variables de ambiente). En el archivo ***convarso***, las imprime con sus valores, mientras que en ***convarsol*** sólo imprime los nombres de las variables de ambiente.

1. \*\* 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.

Tanto en el archivo cc como en el de java te muestran el nombre de usuario, el home directory y el directorio de trabajo.

En el caso del código C, esto se imprime debido a que el código que se corre es un printenv que obtiene el nombre del usuario y el home directory. Para obtener el directorio de trabajo se utiliza el comando de pwd. Para esto se utiliza la función getenv.

En el código de java para obtener los tres datos utiliza System.get.Property() recibiendo como parámetro el nombre de usuario, el home directory t ek directorio de trabajo.

1. \*\* 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.

C++: gethostname([lista de caracteres donde se guarda el hostname], [tamaño de la lista de caracteres])

Java: InetAddress.getLocalHost().getLocalHostname() (usa la biblioteca “java.net”)

1. \*\* 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*).

Como lo menciona anteriormente, el archivo ppiduid.cc despliega el ID del proceso que esta corriendo, así como el ID de su proceso padre. Para esto, se utiliza en el código C la función de getpid que obtiene el ID del proceso y getppid que obtiene el ID del proceso padre.   
**Funciones**

PID: getpid()

PPID: getppid()

UID: getuid()

GID: getgid()

1. \*\* 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***.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. \*\* 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. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. 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”. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. \*\* 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 métodos que arroja milisegundos directamente: *System.currentTimeMillis()*.

Muestre el código y la corrida de ambos programas.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. \*\* 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.