# Sistemas Operativos.

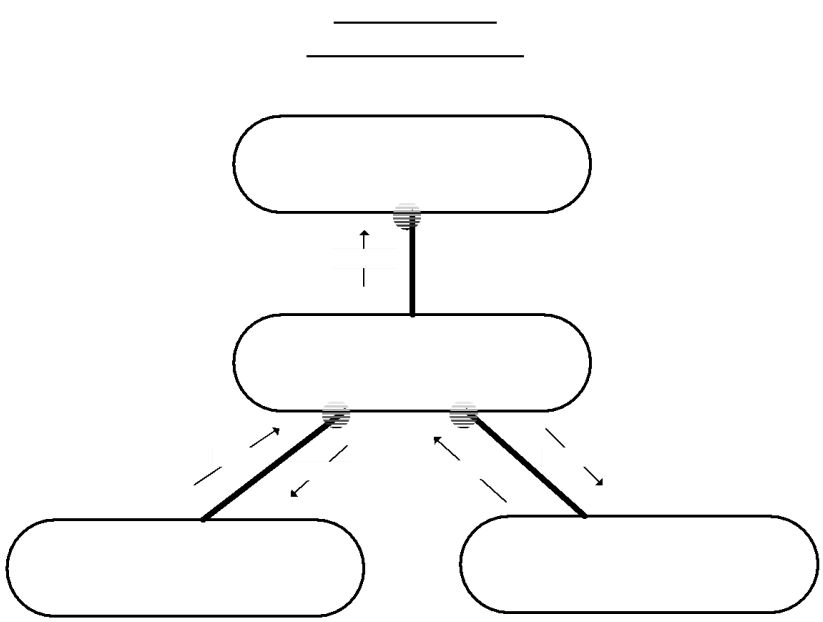
Examen Final Primavera ***02 junio del 2020.***

Nombre: Rebeca Baños García Clave Única: 157655 Calificación: _____

Entre [ ] aparecen los puntos en los cinco problemas. **Conteste usando TINTA. Duración: 2:00 hs.**

Al final deberá subir la solución de los cinco problemas, en un archivo *zip*, a Comunidad, en la entrada de Sistemas Operativos, dentro de Trabajos y Exámenes, en la liga *ExFinTermina*, siguiendo el formato de su clave única seguida de sus iniciales y el texto "ExFin", p.e. ***183459HACRExFin.zip***.

1***)*** **[2.4]** Usted deberá completar el desarrollo de un programa concurrente de cuatro programas ordinarios en Java. Un programa padre, un thread hijo y dos threads nietos. El diagrama aparece más abajo.
Se les da tres programas con código incompleto, que son *Mainprin.java*, *Thrprin.java* y *Uno.java*, que usted puede compilar y ejecutar a partir de *Mainprin.java*, con funcionamiento limitado. Dichos archivos java vienen acompañando este examen.
El programa padre *Mainprin.java* tiene que lanzar la ejecución del thread hijo *Thrprin.java* que a su vez lanzará la ejecución de los threads nietos *Uno.java* y *Dos.java*.



El programa *Mainprin.java* deberá imprimir dos valores decimales que son arrojados como "resultados" por los threads nietos *Uno.java* y *Dos.java*.
*Uno.java* se encargará de sumar los elementos de un arreglo de valores que viene de *Thrprin.java*. La suma de los valores deberá ser "regresado" a *Thrprin.java*. No modifique los valores a sumar.
*Dos.java* se encargará de multiplicar los elementos un arreglo de valores que viene de *Thrprin.java*. El producto de los valores deberá ser "regresado" a *Thrprin.java*. No modifique los valores a multiplicar.

>>>>SIGUIENTE

*Thrprin.java* deberá lanzar primero al thread *Uno.java*, esperando por su terminación, que incluye la devolución de la sumatoria. Después lanzara el otro thread, *Dos.java*, esperando por su terminación que incluye la devolución del producto.
Al final *Thrprin.java* deberá devolver, a *Mainprin.java*, los dos valores "regresados" por los nietos.

*Mainprin.java*, lanzará a *Thrprin.java*, y esperará a que termine, para entonces imprimir los valores "devueltos" por el mismo *Thrprin.java*. Estos dos programas tienen un ejemplo de cómo compartir un objeto para "regresar" los valores e imprimirlos.

El siguiente despliegue es un ejemplo de salida del programa concurrente:

```
D:\folder>javac Mainprin.java

D:\folder>java Mainprin

 Uno: ADIOS.

 Dos: ADIOS.

 Thrprin: ADIOS.

Los dos valores recibidos son:
   18.537145812106843          # Estos valores varían, dependen de un random
   1718.2223439390125

 Mainprin: ADIOS.

D:\folder>
```

Pueden modificar los archivos que ya vienen programados en este problema.

>>>>SIGUIENTE

<<<<ANTERIOR

# Sistemas Operativos.

Examen Final Primavera ***02 junio del 2020.***

Nombre: Rebeca Baños García Clave Única: 157655 Calificación: _____

Entre [ ] aparecen los puntos en los cinco problemas. **Conteste usando TINTA. Duración: 2:00 hs.**

2) **[1.9]** Métodos de instancia en los objetos Thread de Java y estados de los threads.

**a)** Al instanciar un objeto de una clase derivada de Thread, aparecen, entre otros, dentro del objeto los métodos de instancia *run( )* y *start( )*. Explique las diferencias fundamentales entre ambos métodos y ejemplifique con instrucciones Java como se debe mandar la ejecución de ambos métodos (puede suponer que existe definida una clase derivada de Thread).

**b)** Además de *run( )* y *start( )*, mencione cuatro métodos de instancia, que aparezcan en un objeto derivado de una clase Thread.

**c)** Haga el diagrama de estados de los threads y explique las transiciones que llevan a los estados.

**d)** Mencione los beneficios principales, desde el punto de vista de una aplicación o sistema, al tener varios threads dentro de un mismo proceso (pesado).

3) **[1.9]** Conceptos de llamadas al sistema o funciones de procesos.

**a)** ¿Cuál o cuáles son las diferencias entre las funciones del sistema **fork()**, **execX(…)** y **exit(…)**, de manera funcional operativo?

**b)** Explique la operatividad de la función del sistema *system("comando")*. De las funciones del sistema vistas en clase, nombre las que se encuentran usadas dentro de esta función del sistema *system("comando")*.

**c)** Cuando un proceso ejecuta la función **exit(n)** ¿qué evento se lleva a cabo y que otro proceso es avisado de este evento?

**d)** Que pasaría, en el sistema, si un proceso no llegara a ejecutar **exit(n)** para asegurar su terminación.

4) **[1.9]** Problemáticas de la concurrencia.

**a)** ¿Cuáles dos problemas se pueden presentar en toda aplicación concurrente? Estos problemas al presentarse provocan el mal funcionamiento de la aplicación.

**b)** ¿Cómo son solucionadas estos dos problemas concurrentes? Puede basar su explicación apoyándose tangencialmente de Java, explicando las soluciones, no sólo refiriéndose a instrucciones.

**c)** Dentro de un sistema operativo, visto como aplicación concurrente ¿cuáles son sus secciones críticas?

**d)** Puede darse el caso, de un programa Java (proceso pesado) con varios threads, pueda terminar, habiendo todavía más de un thread (hilo) activo. ¿Explique cómo sucedería esta situación en Java? Puede justificar su respuesta con algún ejemplo o instrucciones.

>>>>SIGUIENTE

<<<<ANTERIOR

**5) [1.9]** Comandos de archivos.
En su directorio de trabajo (home directory) se encuentran los subdirectorios ***uno***, ***dos*** y ***tres***, así como los archivos ordinarios cuatro, cinco y seis. Por ahora, tanto los directorios como los archivos ordinarios, son hermanos jerárquicamente hablando, o son hijos del mismo directorio padre.
Explique usted, para cada uno de los comandos que se listan a continuación, lo que va ocurriendo con los archivos y los directorios, cuando se va ejecutando cada comando. **En cada caso (comando aplicado) dibuje una gráfica jerárquica donde muestre lo ocurrido**. Los ***subdirectorios*** enciérrelos con un rectángulo y los archivos ordinarios con un círculo.

**mv cuatro siete**
**mv dos ocho**
**mv siete ocho**
**mv uno tres**
**mv ocho siete**

>>>>SIGUIENTE

<<<<ANTERIOR

2) **[1.9]** Métodos de instancia en los objetos Thread de Java y estados de los threads.

**a)** Al instanciar un objeto de una clase derivada de Thread, aparecen, entre otros, dentro del objeto los métodos de instancia *run( )* y *start( )*. Explique las diferencias fundamentales entre ambos métodos y ejemplifique con instrucciones Java como se debe mandar la ejecución de ambos métodos (puede suponer que existe definida una clase derivada de Thread).

**b)** Además de *run( )* y *start( )*, mencione cuatro métodos de instancia, que aparezcan en un objeto derivado de una clase Thread.

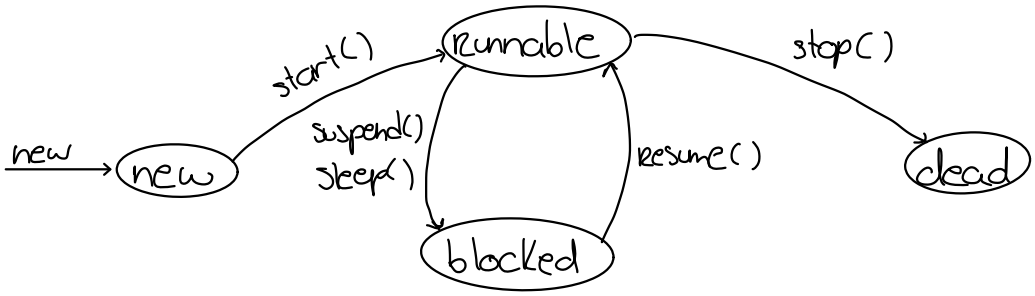
**c)** Haga el diagrama de estados de los threads y explique las transiciones que llevan a los estados.

**d)** Mencione los beneficios principales, desde el punto de vista de una aplicación o sistema, al tener varios threads dentro de un mismo proceso (pesado).

a) El método run( ) se encarga de iniciar un thread, es equivalente al método main, ya que dentro de este método se encuentra el cuerpo del thread. En cambio el método start( ) se encarga de ejecutar un nuevo thread a partir de un thread que lo invoca, es decir, el método start utiliza el método run( ) y además continua la ejecución en el thread invocador.

b) suspend( ); sleep( ); resume( ), stop( ).

c)




en new entran los nuevos threads que van a ser tratados, tienen que pasar por la instancia start( ) para poder pasar al estado de runnable, donde se mantienen como en la "cola de ready" esperando ser ejecutados. De ahí pueden pasar a blocked o dead, pasan a blocked por los métodos de sleep( ) o suspend( ). esto es para esperar una señal I/O o porque terminó su rebanada de tiempo; regresan a Runnable con el método resume( ) para poder volver a correrlo y por último pasa a dead si se detiene con el método stop( ).

d) Tener varios threads ayuda a que el proceso se haga menos costoso, ya que los recursos se comparten y la planificación es más ordenada ya que el orden de flujo lo controlan los threads.

3) **[1.9]** Conceptos de llamadas al sistema o funciones de procesos.

**a)** ¿Cuál o cuáles son las diferencias entre las funciones del sistema **fork()**, **execX(…)** y **exit(…)**, de manera funcional operativo?

**b)** Explique la operatividad de la función del sistema *system("comando")*. De las funciones del sistema vistas en clase, nombre las que se encuentran usadas dentro de esta función del sistema *system("comando")*.

**c)** Cuando un proceso ejecuta la función **exit(n)** ¿qué evento se lleva a cabo y que otro proceso es avisado de este evento?

**d)** Que pasaría, en el sistema, si un proceso no llegara a ejecutar **exit(n)** para asegurar su terminación.

a) forK() se encarga de crear un nuevo proceso hijo cargando el mismo programa del padre, creando un hijo clon, exec() se utiliza después del forK(), donde reemplaza al hijo creado por un nuevo programa y lo arranca, por último exit() se encarga de cerrar archivos abiertos, liberar todos los recursos, salvar estadísticas de uso de recursos, enviar un "wake-up" al padre y llamar al switch.

b) La función system (comando) nos permite realizar la ejecución de un comando del sistema, por lo tanto la salida de este no podrá ser guardada en una variable, sin embargo el retorno de esta función será el valor retornado por el comando.

c) Se termina el proceso hijo con el PID n y se le avisa al padre del hijo que ha terminado.

d) Algunos archivos podrían permanecer abiertos o no se liberarían todos los recursos utilizados

4) **[1.9]** Problemáticas de la concurrencia.

**a)** ¿Cuáles dos problemas se pueden presentar en toda aplicación concurrente? Estos problemas al presentarse provocan el mal funcionamiento de la aplicación.

**b)** ¿Cómo son solucionadas estos dos problemas concurrentes? Puede basar su explicación apoyándose tangencialmente de Java, explicando las soluciones, no sólo refiriéndose a instrucciones.

**c)** Dentro de un sistema operativo, visto como aplicación concurrente ¿cuáles son sus secciones críticas?

**d)** Puede darse el caso, de un programa Java (proceso pesado) con varios threads, pueda terminar, habiendo todavía más de un thread (hilo) activo. ¿Explique cómo sucedería esta situación en Java? Puede justificar su respuesta con algún ejemplo o instrucciones.

a) Se presenta el problema de resultados no deterministas o el deadlock o abrazo mortal.

b) Se puede controlar la RACE CONDITION implementando una exclusión mutua entre procesos, condicionando a que solo un proceso pueda entrar a su sección crítica.

c) Es un segmento de código, de donde cada uno de los procesos concurrentes acceden al objeto compartido.

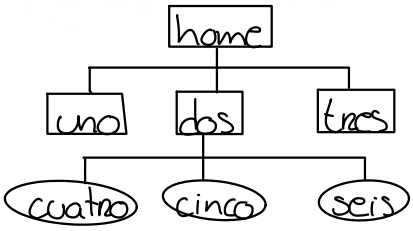
d) Esto puede suceder ya que no se detuvo de manera correcta el Thread. es decir, pudo ponerse en pausa con el método sleep( ) y en ese momento el programa pesado se detuvo, lo que provoca que el hilo al terminar su tiempo de sleep vuelva a estar activo.

**5) [1.9]** Comandos de archivos.

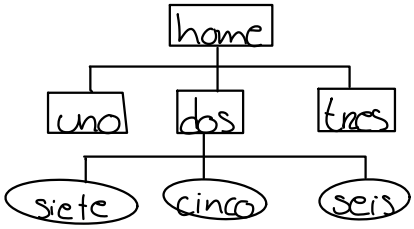
En su directorio de trabajo (home directory) se encuentran los subdirectorios ***uno***, ***dos*** y ***tres***, así como los archivos ordinarios cuatro, cinco y seis. Por ahora, tanto los directorios como los archivos ordinarios, son hermanos jerárquicamente hablando, o son hijos del mismo directorio padre.

Explique usted, para cada uno de los comandos que se listan a continuación, lo que va ocurriendo con los archivos y los directorios, cuando se va ejecutando cada comando. **En cada caso (comando aplicado) dibuje una gráfica jerárquica donde muestre lo ocurrido**. Los ***subdirectorios*** enciérrelos con un rectángulo y los archivos ordinarios con un círculo.

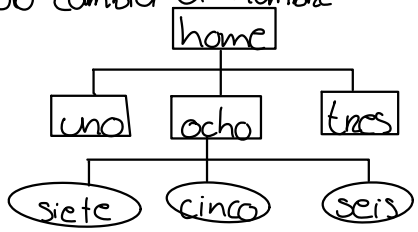
**mv cuatro siete**
**mv dos ocho**
**mv siete ocho**
**mv uno tres**
**mv ocho siete**



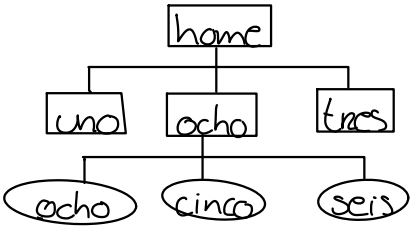

mv cuatro siete: cambia el nombre del archivo ordinario cuatro a siete.



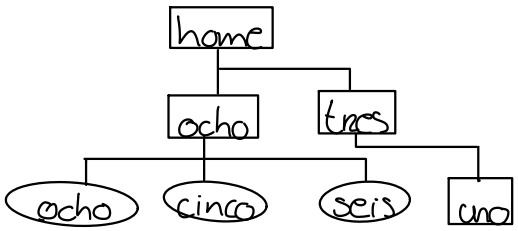

mv dos ocho: mueve el directorio dos al directorio ocho, como no existe el directorio ocho, solo cambia el nombre




mv siete ocho: cambia el nombre del archivo siete a ocho.




mv uno tres: mueve el directorio uno al tres.



mv ocho siete: cambia el directorio ocho al siete, como no existe, le cambia el nombre.