TRABAJO PRÁCTICO Nº 1

SISTEMAS OPERATIVOS MULTIPROCESADOR

Temas de la unidad:

1. Tipos
   a. Simétricos
   b. Asimétricos
   c. Multinúcleo
   d. Mixtos
2. Tipos de conexión a memoria RAM
   a. UMA
   b. NUMA
3. Sincronización
4. Planificación del uso de la CPU
   a. Afinidad
   b. Migración
   c. Balance
5. Administración de las memorias caché del procesador
   a. Tipos de caché
   b. Problemáticas de sincronización

Objetivos específicos:

- Que el alumno adquiera los conceptos básicos de las problemáticas y las respectivas soluciones de los S.O.A. basados en multiprocesadores.
- Desarrollar en el alumno las habilidades necesarias para administración de los S.O.A. basados en multiprocesadores.

- *Es recomendable desarrollar la práctica sobre el sistema operativo Linux, preferentemente en distribuciones Ubuntu, Mint o Debian. Los ejercicios de esta práctica tienen sentido cuando se realizan sobre hardware real. Dentro de una máquina virtual posiblemente se obtengan resultados que no coincidan con la realidad, ya que se utilizan procesadores, cache y memoria virtual.*

1) Escribir un script en bash que imprima por salida estándar los números 1 al 10000 utilizando una estructura repetitiva *for*. Al finalizar todas las impresiones, mostrar un mensaje avisando la finalización de dicho script. Guardar el script como *01_monopr.sh*.
   - **a.** Modificar el script para que las escrituras por salida estándar se realicen en forma concurrente utilizando el operador *&*. Guardar el nuevo script como *01_multipr.sh*. Describir brevemente lo que ocurre con la secuencia de números escrita por pantalla.
   - **b.** Modificar el script utilizando la palabra reservada *wait*, para que las escrituras por pantalla queden reordenadas. Guardar el nuevo script como *01_wait.sh*.
   - **c.** ¿Se podría utilizar *sleep* para lograr la misma salida que en el punto anterior? Guardar el nuevo script como *01_sleep.sh*.
   - **d.** ¿Qué diferencia hay entre *wait* y *sleep*?

- *Se puede encontrar ayuda del operador **&** dentro de **man bash**, en el cual se encuentra: "If a command is terminated by the control operator &, the shell executes the command in the background in a subshell. The shell does not wait for the command to finish, and the return status is 0."*

2) Utilizando la herramienta *time*, analice la ejecución de los scripts del ejercicio anterior *monopr.sh* y *multipr.sh*.
   - **a.** Redirigir las salidas a los ficheros *02_time-monopr.log* y *02_time-multipr.log* respectivamente.
   - **b.** Dentro de cada fichero agregar comentarios describiendo cómo se utilizó *time* y explicar brevemente que sucede con el tiempo de ejecución.

- *nos referimos a la herramienta **time** que se accede al ejecutar **/usr/bin/time**, y se instala con **sudo apt-get install time**, ya que existe una confusión con el utilitario de bash llamado también **time**. Al escribir **help time** se accede a la ayuda del utilitario de bash. Al ejecutar **man time** se accede a la ayuda de **/usr/bin/time**.*

3) Utilizando la herramienta *perf*, analice la ejecución de los scripts del ejercicio anterior *monopr.sh* y *multipr.sh*.
   - **a.** Redirigir las salidas a los ficheros *03_perf-monopr.log* y *03_perf-multipr.log* respectivamente.
   - **b.** Dentro de cada fichero agregar comentarios describiendo cómo se utilizó *perf*, qué parámetros fueron utilizados y explicar brevemente lo que sucede con los cambios de contexto.

- ***perf** requiere de "instrumentación" en el kernel de linux, para ello, se deben instalar herramientas embebidas en el kernel, que se encuentran en el paquete **linux-tools**.*
  - *sudo apt-get install linux-tools-common linux-tools-generic linux-tools-$(uname -r)*

Más información en www.brendangregg.com/perf.html

4) Escribir un script *04_hardware-dmi.sh* que, utilizando el comando *dmidecode*, genere como salida un archivo llamado *04_hardware.txt* que contenga:
- **a.** Nombre del fabricante del sistema.
- **b.** Versión del procesador que se está utilizando.
- **c.** Fecha de inicio de la BIOS.
- **d.** Memoria máxima que el sistema permite.

5) Escribir un script *05_architecture.sh* que, utilizando el comando *lscpu*, imprima por salida estándar:
- **a.** Si el tipo de arquitectura utilizada por el sistema es de 32 o 64 bits.
- **b.** Si el sistema es "*mono CPU*" o "*multi CPU*", según cantidad de *CPU(s)* disponibles.

6) Inicie un reloj que se actualice a cada segundo en su escritorio gráfico utilizando el comando *xclock -d -update -1*. Dentro de un fichero *06_clock.txt* responda a las siguientes preguntas:
- **a.** Si ahora se oprime la combinación de teclas *Ctrl+Z*, y luego se ejecuta el comando *fg %1*, ¿qué sucede con el estado del proceso paso a paso?
- **b.** Si se vuelve a oprimir la combinación de teclas *Ctrl+Z,* y luego se ejecuta el comando *bg %1*, ¿qué sucede con el estado del proceso paso a paso?
- **c.** ¿Cómo podría hacer para que se inicie el proceso directamente sin perder control de la línea de comandos?

➢ *Es necesario instalar previamente el paquete **x11-apps**. En caso de no tenerlo instalado, instalar con **sudo apt-get install x11-apps**.*

7) Escribir un script *07_process-trap.sh* que:
- **a.** Cada segundo imprima un punto por salida estándar.
- **b.** Cuando comience la ejecución del script, previo a imprimir los puntos, se deberá imprimir por salida estándar el *PID*.
- **c.** Utilizando el comando *trap* capturar la pseudo señal *EXIT*, para que cada vez que termine la ejecución del script se imprima el mensaje "*Fin del script*".
- **d.** Capturar la señal *SIGINT***,** generada al oprimir la combinación de teclas *Ctrl+C*. Al detectar dicha interrupción se debe mostrar el mensaje "*Interrupción detectada*" y continuar la ejecución.
- **e.** Capturar la señal *SIGTSTP*, generada al oprimir la combinación de teclas *Ctrl+Z*. Al detectar dicha interrupción se debe mostrar el mensaje "*Suspender ejecución*" y finalizar la ejecución.
- **f.** Desde otra línea de comandos, utilizar el comando *kill* para enviar señales *SIGINT* y *SIGTSTP* al script, y probar que muestren los mensajes definidos en los puntos anteriores. Mencionar en un archivo *07_process-kill.txt* las pruebas realizadas.

8) ¿Qué comandos en bash pueden simular la implementación de un semáforo atómico? Dentro de un fichero *08_atomic-bash.sh* escriba un ejemplo.

9) Explique en un fichero *9_taskset.txt*:
- **a.** ¿Para qué sirve el comando *taskset* en Linux?
- **b.** Mencione bajo qué condiciones es conveniente utilizarlo.
- **c.** Escriba una breve conclusión luego de utilizar dicho comando sobre un programa que utilice fuertemente la caché y tenga una ejecución larga en el tiempo. Puede ser algo como comprimir o descomprimir un archivo grande (500mb) con tar -z, gzip, zip o la ejecución de un programa pesado como Firefox, libreoffice, gimp, etc.

10) Utilizando la herramienta *perf* medir los *cache misses*, *cache references (hits)* y tiempo transcurrido de la ejecución de *01_monopr.sh* y luego:
- **a.** Guardar los resultados obtenidos en un fichero *10_cache.log*.
- **b.** Repetir el proceso utilizando *taskset* y agregar los resultados en el fichero anterior.
- **c.** Explicar brevemente diferencias al utilizar *taskset*.

11) Explique en un fichero *11_pthreads.txt* para qué sirven las siguientes sentencias *PThreads* y en qué casos se podría utilizar cada una:
- ***a.*** *pthread_create()*
- ***b.*** *pthread_join()*
- ***c.*** *pthread_mutex_lock()*
- ***d.*** *pthread_mutex_trylock()*
- ***e.*** *pthread_cond_wait()*
- ***f.*** *pthread_cond_signal()*
- **g.** *pthread_cond_broadcast()*

12) Copie el código presentado en el siguiente enlace y, en nuevo fichero *12_desync.c*, implemente los cambios necesarios para sincronizar los hilos y que el resultado sea cero.

https://github.com/utnfrlp/SOA/blob/master/TP1/desync.c

➢ *Para ejecutar el código C previamente es necesario compilarlo con la librería **gcc**. En caso de no tenerla instalada se puede instalar con **sudo apt-get install gcc libc6-dev**.*
- *Compilar con **gcc desync.c -o desync.out -lpthread***
  *(la extensión en el fichero compilado es opcional, y se debe utilizar el parámetro **-lpthread** para indicar al compilador que se utilizará la librería PThreads).*
- *Ejecutar con **./desync.out***

13) Describa brevemente en un fichero *13_mutex.sh* cómo se puede garantizar exclusión mutua, o *mutex*, entre procesos en bash.

14) Explique en un fichero *14_thread-safety.log*:
- **a.** ¿Qué significa *thread safety*?
- **b.** ¿Qué condiciones debe cumplir un proceso para considerarse *thread safe*?
- **c.** ¿En qué situaciones se debe garantizar que un proceso sea *thread safe*?
- **d.** Cite ejemplos.

15) Analice el código presentado en el siguiente enlace con el *Algoritmo de Peterson*, y explique en un fichero *15_petersonA.txt* qué tarea realiza.

https://github.com/utnfrlp/SOA/blob/master/TP1/peterson_a.c

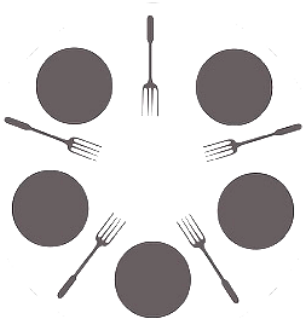
16) Analice el código presentado a continuación para ver fallas en el Algoritmo de Peterson en la computación multiprocesamiento actual y el uso de *__sync_synchronize()*. Luego en un fichero *16_petersonB.txt* explique qué sucede y por qué.

https://github.com/utnfrlp/SOA/blob/master/TP1/peterson_b.c

17) En un fichero *17_single-writer.c* implemente una estrategia "*Single writer / multi reader*", garantizando la exclusión mutua, sobre un contador en el que 10 *threads* intentan aumentar su valor de a una unidad hasta que dicho contador llegue a 100.

18) Problema de la cena de los filósofos.

- **a.** En un fichero *18_philosophersA.c* implementar el problema en lenguaje *C*, con 5 filósofos y 5 tenedores, y ejecutarlo hasta conseguir lograr un interbloqueo.
- **b.** En un nuevo fichero *18_philosophersB.c*, ¿qué cambios debería realizar en el código para garantizar que no haya un interbloqueo?



19) ***(EXTRA)*** Utilizando la librería *threading* https://docs.python.org/3/library/threading.html, reescribir el *ejercicio 11* a lenguaje *Python* en un fichero *19_python-threading.py* con los cambios necesarios para sincronizar los hilos y que el resultado sea cero.

- *Condiciones de entrega: se debe entregar un fichero .zip con todos los ejercicios dentro, el cual deberá tener el nombre del grupo y el número de trabajo práctico, por ejemplo:* ***GRUPO_01_TP1.zip***. Dicho fichero será subido a Moodle por un integrante del grupo y, para que la entrega se haga efectiva, deberá ser validado por los demás integrantes.
- *Condiciones de evaluación: Como pueden existir diversas formas de realizar cada ejercicio, siempre que se consiga el objetivo del mismo se considerará que la respuesta es válida.*
- *Entregas fuera de los plazos estipulados de entrega tendrán impacto en la nota, pudiendo hacer que el trabajo práctico sea desaprobado.*