**Trabajo Práctico Nº 4: Threads**

1. Enumere las razones por las que un cambio de contexto entre hilos puede ser más rápido que un cambio de contexto entre procesos (ventajas de los threads sobre los procesos).

- La ventaja de los hilos sobre los procesos es que los hilos comparten el mismo espacio de memoria dentro de un proceso, lo que significa que cambiar de un hilo a otro dentro del mismo proceso es más rápido que cambiar de un proceso a otro, donde se necesita cambiar todo el espacio de direcciones.

- Los hilos comparten recursos como archivos abiertos dentro del mismo proceso, lo que evita la necesidad de duplicar estos recursos durante el cambio de contexto

1. Considere un servidor de archivos ejecutando como un proceso de un solo hilo en una máquina donde existen otros procesos ejecutándose. Para mejorar la performance, se desea agregar a dicho proceso soporte multihilo con acceso al núcleo para tratar cada petición de archivo por separado. Se decide utilizar threads de nivel de usuario. Sin embargo, la performance del proceso no mejora. ¿Cuáles pueden ser la(s) razones(s) para que esto ocurra? Justifique.

- Si el servidor de archivos tiene secciones críticas donde se bloquean recursos compartidos, como el acceso al disco o la memoria, los threads de nivel de usuario pueden no ser efectivos en la realización de operaciones de E/S concurrentes ya que los demás hilos a nivel usuario quedarían bloqueados.

- La asignación de hilos a procesadores físicos es manejada por la aplicación no por el SO, si el SO asigna el proceso a un solo procesador en un momento dado, solo un hilo de ese proceso podrá ejecutarse simultáneamente incluso si hay múltiples núcleos del procesador disponible.

-Se puede considerar el uso de threads a nivel kernel, ya que el sistema operativo puede administrar la asignación de hilos a núcleos de manera más eficiente que la aplicación.

1. Considérese un entorno en el que hay una asociación uno a uno entre hilos de nivel de

usuario e hilos de nivel de núcleo que permite a uno o más hilos de un proceso ejecutar llamadas al sistema bloqueantes mientras otros hilos continúan ejecutando. Explique por qué este modelo puede hacer que se ejecuten más rápido los programas multihilo que sus correspondientes versiones monohilo en una máquina monoprocesador.

En un programa multihilo, mientras un hilo está esperando una operación de entrada/salida bloqueante (por ejemplo, lectura de un archivo o una solicitud de red), otros hilos pueden continuar ejecutándose. Esto significa que el procesador no queda inactivo durante períodos prolongados esperando que se completen las operaciones de E/S, lo que resulta en una utilización más eficiente del tiempo de CPU y, en última instancia, en un mejor rendimiento. Esto permite que varios hilos de un proceso se ejecuten simultáneamente en varios núcleos de procesador, lo que puede aumentar la concurrencia y el rendimiento del programa multihilo.

1. **[LAB]** Implemente el siguiente código, compile y ejecute. Determine qué realiza cada parte del código. Ejecute reiteradas veces.

Nota: *para compilar incluir la librería pthread.h.*

a) ¿Siempre terminan todos los threads?

b) Comente las líneas con las instrucciones pthread\_join() y vuelva a ejecutar reiteradas veces. Describa y justifique el comportamiento observado.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  void \*funcion()  {  printf("Thread: %d\n", pthread\_self());  pthread\_exit(0);  }  int main()  {  pthread\_t th1, th2;  pthread\_create(&th1, NULL, funcion, NULL);  pthread\_create(&th2, NULL, funcion, NULL);  printf("El thread principal continúa ejecutando: %d\n", pthread\_self());  pthread\_join(th1,NULL);  pthread\_join(th2,NULL);  return 0;  } |

1. **[LAB]** Implemente el siguiente código, compile y ejecute. Determine qué realiza cada parte del código. Ejecute reiteradas veces.

Nota: *para compilar incluir la librería pthread.h.*

a)¿Siempre terminan todos los threads?

b) Descomente las instrucciones comentadas, compile y ejecute nuevamente. ¿Qué hace dicha instrucción? ¿qué efecto tiene en este código?

|  |
| --- |
| #include <stdlib.h>  #include <pthread.h>  #include <windows.h>  #define MAX 10  void \*funcion1(\*i)  {  printf("Thread: %d, el valor de i: %d\n", pthread\_self(), i );  pthread\_exit(0);  }  int main()  {    for(int i = 0; i < MAX; i++){  pthread\_create(&thid[i], &attr, funcion1, i);  }  /\*  printf("Start sleep\n");  Sleep(10000);  printf("End sleep\n");  \*/  return 0;  } |

1. **[LAB]** Implemente el siguiente código, compile y ejecute. Determine qué realiza cada parte del código. Ejecute reiteradas veces.

Nota: *para compilar incluir la librería pthread.h.*

1. ¿Siempre terminan todos los threads?
2. Comente las líneas con las instrucciones pthread\_join() y vuelva a ejecutar reiteradas veces. Describa y justifique el comportamiento observado.

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <pthread.h>  int var=10;  void \*funcion1()  {  sleep(30);  a = a \* 2;  println("Thread: %d - valor de a: %d", pthread\_self(), a);  pthread\_exit(0);  }  void \*funcion2()  {  sleep(5);  a = a + 5;  println("Thread: %d - valor de a: %d", pthread\_self(), a);  pthread\_exit(0);  }  int main()  {  pthread\_t th1, th2;  pthread\_create(&th1, NULL, funcion1, NULL);  pthread\_create(&th2, NULL, funcion2, NULL);  printf("El thread principal continúa ejecutando: %d\n", pthread\_self());  pthread\_join(th1,NULL);  pthread\_join(th2,NULL);  return 0;  } |

1. **[LAB]** “Hello World!” V2.0... ahora con Threads!

Escribir el código C que escriba en pantalla n veces el mensaje: “Hola Threads!. Soy el thread [idThread] ejecutando por [enésima] vez”, donde [idThread] es el identificador de thread asignado por el lenguaje C y [n-ésima] es la cantidad de veces que se imprimió el texto por pantalla. Para obtener este identificador se puede utilizar el siguiente código:  
**pthread\_self()**  
El código C debe crear 10 threads y ejecutar cada uno de ellos 1.000 veces (n = 1.000). Describir brevemente el resultado de la ejecución.