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).

2. 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.

3. 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.

4. **[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;
}
```

5. **[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;
}
```

6. **[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>

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;
}
```

7. **[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.