Universidad Nacional de San Agustín de Arequipa

Escuela Profesional de Ciencia de la Computación

Sistemas Operativos

Laboratorio 5: Hilos

Presentado por:
Rushell Vanessa Zavalaga Orozco

Docente :
Dra. Yessenia D. Yari R.

1. Indicar que hacen las siguientes funciones y que parámetros necesitan:

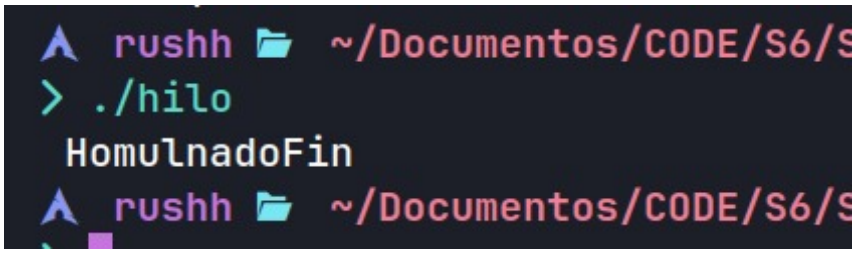
- **pthread_create** Crea un nuevo hilo.
  - **Parámetros:**
    - Puntero a pthread_t (identificación de hilo creado).
    - Atributos del hilo (o NULL para usar los valores predeterminados).
    - La función que el hilo ejecutará.
    - Argumento que se pasará a la función del hilo (puede ser NULL si no necesita argumentos).
- **pthread_join**: Bloquea el hilo que llama hasta que el hilo especificado finalice su ejecución.
  - **Parámetros**:
    - `pthread_t thread`: El identificador del hilo que quieres esperar.
    - `void **retval`: Un puntero que almacenará el valor de retorno de la función ejecutada por el hilo, si es necesario. Si no necesitas este valor, puedes pasar `NULL`.
- **pthread_detach**: Desvincula un hilo, permitiéndole liberar sus recursos al finalizar automáticamente. Marca el hilo como "desvinculado"(detached). Esto significa que los recursos del hilo se liberarán automáticamente cuando el hilo termine.
  - **Parámetros**:
    - `pthread_t thread`: El identificador del hilo que deseas desvincular.
- **pthread_exit**: Termina el hilo que llama a la función y puede devolver un valor específico que otros hilos (usando `pthread_join`) pueden capturar.
  - **Parámetros**:
    - `void *retval`: Un puntero a cualquier valor que quieras devolver para que pueda ser recogido por `pthread_join` (puede ser `NULL` si no necesitas devolver nada).
- **pthread_self**: Devuelve el identificador del hilo que llama a esta función. **Sin parámetros**
- **pthread_kill**: Envía una señal específica a un hilo, como una solicitud para terminar o ejecutar una acción específica.
  - **Parámetros**:
    - `pthread_t thread`: El identificador del hilo al que quieres enviar la señal.
    - `int sig`: La señal que deseas enviar. Puede ser cualquier señal válida en el sistema, como `SIGINT` o `SIGTERM`.

2. Analisis del código (hilo1) :

El siguiente código en C utiliza la biblioteca `pthread.h` para crear dos hilos que imprimen mensajes de manera intercalada, con una pausa de un segundo entre cada letra. Al finalizar la ejecución de ambos hilos, el programa imprime la palabra “Fin”.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>

void *hola(void *arg) {
    char *msg = "Hola";
    int i;
    for(i = 0; i < strlen(msg); i++) {
        printf("%c", msg[i]);
        fflush(stdout);
        usleep(1000000);
    }
    return NULL;
}

void *mundo(void *arg) {
    char *msg = " mundo";
    int i;
    for(i = 0; i < strlen(msg); i++) {
        printf("%c", msg[i]);
        fflush(stdout);
        usleep(1000000);
    }
    return NULL;
}

int main(int argc, char *argv[]) {
    pthread_t h1;
    pthread_t h2;
    pthread_create(&h1, NULL, hola, NULL);
    pthread_create(&h2, NULL, mundo, NULL);
    pthread_join(h1, NULL);
    pthread_join(h2, NULL);
    printf("Fin \n");
}
```

Listing 1: Código en C con hilos para imprimir mensajes intercalados



Primero veremos la utilidad de las bibliotecas incluidas en el código:

- `stdio.h`: Se utiliza para entrada y salida, específicamente para la función `printf`.
- `stdlib.h`: Incluye funciones generales de propósito, aunque en este código no se utiliza ninguna función específica de esta biblioteca.
- `string.h`: Funciones de manipulación de cadenas, como `strlen`, que se usa para obtener la longitud de las cadenas a imprimir.
- `unistd.h`: Funciones de control de proceso, como `usleep`, que introduce un retardo en microsegundos.
- `pthread.h`: Biblioteca para trabajar con hilos en C, para la creación y manejo de hilos.

Luego vemos dos funciones de los hilos.

1. `void *hola(void *arg)`: Imprime la palabra "Hola" letra por letra, con una pausa de un segundo (1000000 microsegundos) entre cada letra.
2. `void *mundo(void *arg)`: Imprime la palabra " mundo" (con un espacio inicial) letra por letra, también con una pausa de un segundo entre cada letra.

Ambas funciones tienen la misma estructura:

- Se define un puntero a `char` que apunta la cadena que se desea imprimir.
- Un bucle `for` recorre la cadena, lo imprime, y utiliza `fflush(stdout)` para forzar el vaciado del búfer de salida, garantizando que cada letra se muestre inmediatamente.
- La función `usleep(1000000)` introduce una pausa de un segundo entre cada letra para que el mensaje se imprima lentamente y de manera intercalada con el mensaje del otro hilo.

Por último la funcion main:

1. Declara dos variables de tipo `pthread_t`, `h1` y `h2`, para identificar los hilos que se van a crear.

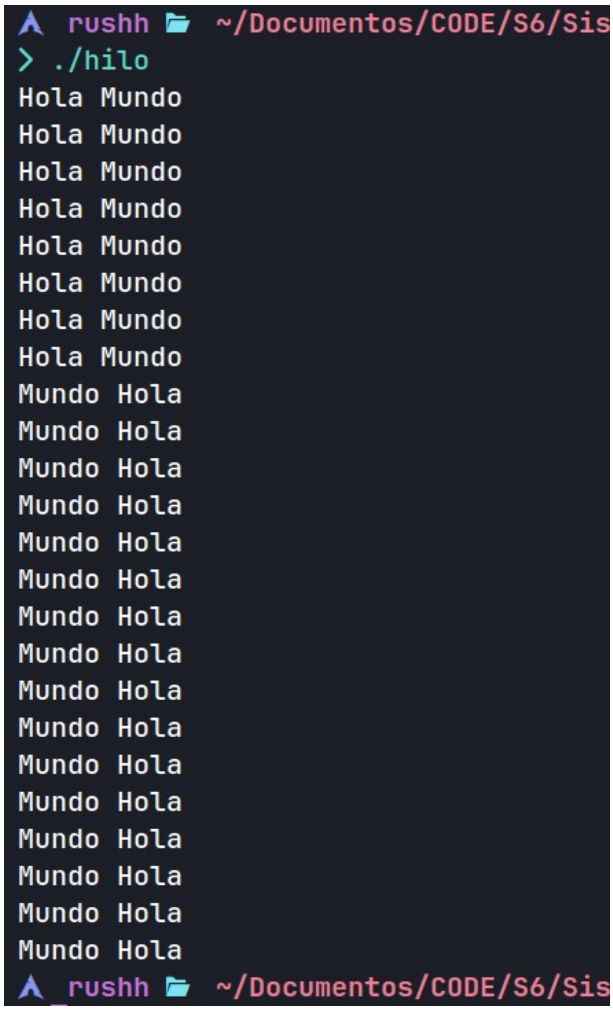
2. Llama a `pthread_create` dos veces:
    - La primera llamada crea el hilo `h1` y lo asocia con la función `hola`.
    - La segunda llamada crea el hilo `h2` y lo asocia con la función `mundo`.
3. Llama a `pthread_join` para esperar a que los hilos `h1` y `h2` terminen su ejecución antes de continuar.
4. Finalmente, imprime "Fin" al finalizar la ejecución de ambos hilos.

3. Analisis del código (hilo2) :

Este código en C crea dos hilos usando la biblioteca `pthread.h` y los configura para imprimir mensajes diferentes una cierta cantidad de veces.

```
#include <stdio.h>
#include <pthread.h>

#define N 8
#define M 16

const char *mensaje[2] = {"Hola Mundo", "Mundo Hola"};
const int cantidad[2] = {N, M};

void imprime_mensaje(void *ptr) {
        int i;
        int id;

        id = *(int *) ptr;
        for(i = 0; i < cantidad[id]; i++)
                printf("%s\n", mensaje[id]);
        return;
}

int main(int argc, char *argv[]) {
        pthread_t hilo0, hilo1;
        int id0 = 0, id1 = 1;

        pthread_create(&hilo0, NULL, (void *) imprime_mensaje, (
   void *) &id0);
        pthread_create(&hilo1, NULL, (void *) imprime_mensaje, (
   void *) &id1);

        pthread_join(hilo0, NULL);
        pthread_join(hilo1, NULL);

        return 0;
}
```

Listing 2: Código en C con hilos para imprimir múltiples mensajes



Primero definimos constantes y variables.

- `#define N 8` y `#define M 16`: Estas constantes representan la cantidad de veces que se imprimirá cada mensaje.
- `const char *mensaje[2]`: Un arreglo de dos cadenas de caracteres que contiene los mensajes a imprimir: "Hola Mundo" y "Mundo Hola".
- `const int cantidad[2]`: Un arreglo de enteros que especifica la cantidad de repeticiones para cada mensaje. El primer mensaje se imprime 8 veces y el segundo 16 veces.

La función `imprime_mensaje` toma un puntero genérico `void *ptr` como argumento, que se convierte a un puntero de tipo `int *` para obtener el identificador del mensaje (0 o 1). Según el identificador:

1. La función accede al mensaje correspondiente en `mensaje[id]`.
2. Utiliza un bucle `for` para imprimir el mensaje tantas veces como indique `cantidad[id]`.
3. La función `printf` imprime el mensaje seguido de un salto de línea (`\n`).

La función `main` realiza las siguientes tareas:

1. Declara dos variables de tipo `pthread_t`, `hilo0` y `hilo1`, para identificar los hilos.
2. Define dos enteros, `id0` y `id1`, que representan los identificadores de mensaje 0 y 1, respectivamente.
3. Llama a `pthread_create` dos veces para crear los hilos:
   - El primer hilo `hilo0` se asocia con la función `imprime_mensaje` y recibe `id0` como parámetro.
   - El segundo hilo `hilo1` se asocia con la misma función y recibe `id1` como parámetro.
4. Llama a `pthread_join` para esperar a que ambos hilos terminen antes de continuar, lo que garantiza que el programa no finalice hasta que ambos hilos hayan terminado de imprimir.

4. Utilidad de la función `fflush(stdout)`

La función `fflush(stdout)` se utiliza para vaciar el búfer de salida de `stdout`. Esto significa que cualquier dato que esté en el búfer se envía inmediatamente a la consola. Normalmente, cuando imprimimos caracteres en la consola, estos se almacenan en un búfer y solo se muestran cuando el búfer está lleno o cuando el programa termina. Al usar `fflush(stdout)`, podemos forzar que cada carácter o línea se imprima de inmediato, sin esperar a que el búfer se llene. Esto es útil cuando queremos ver la salida en tiempo real.

5. Hilo 1: ¿Qué pasa si elimino `usleep(1000)`?

En el contexto de un programa multihilo, la función `usleep` introduce una pausa en la ejecución de un hilo, lo que permite que otros hilos puedan ejecutarse en paralelo. Si eliminas la llamada a `usleep(1000)`, el hilo podría imprimir los caracteres de su mensaje mucho más rápido, posiblemente de una sola vez, sin dar oportunidad a que otros hilos puedan intercalar sus mensajes en la consola.

6. Hilo 2: ¿Se pueden crear más hilos y ejecutar una función `imprime_mensaje`?

Sí, en C podemos crear tantos hilos como lo permita la memoria y los recursos del sistema. Puedes crear múltiples hilos y hacer que cada uno ejecute la función `imprime_mensaje` con diferentes argumentos.

7. Modificación en el código de Hilo 2: ¿Qué sucede si se duermen la mitad de los hilos creados?

Si pausamos la mitad de los hilos creados, mediante una llamada a `usleep` o `sleep` dentro de la función que ejecutan, esos hilos se suspenderán temporalmente. Esto tiene varios efectos posibles:

- **Paralelismo reducido**: Los hilos que están "dormidos"no consumirán tiempo de CPU, lo cual permite que los hilos activos puedan ejecutar sus tareas sin interferencia. Sin embargo, esto reduce el número de hilos que trabajan simultáneamente.
- **Retardo en la ejecución total**: Si la mitad de los hilos están dormidos y deben esperar un tiempo considerable para ponerse como activos, el programa completo podría tardar más en terminar, especialmente si esos hilos son necesarios para completar ciertas tareas.

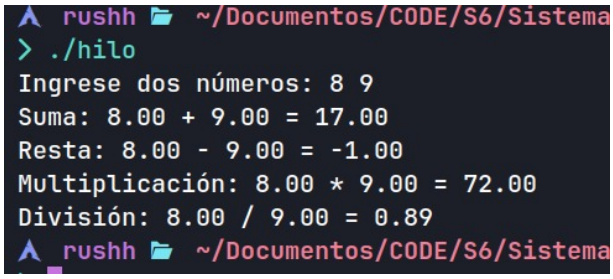
8. Calculadora Simple con Hilos en C

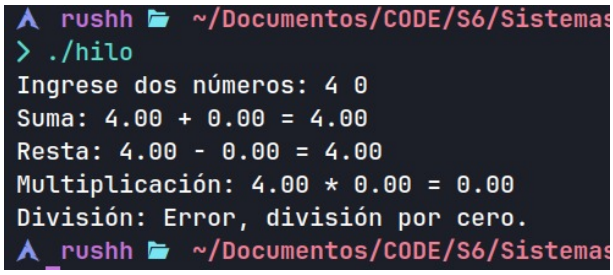
Calculadora simple con cuatro operaciones básicas (suma, resta, multiplicación y división) utilizando hilos. Cada operación se ejecuta en un hilo independiente, permitiendo realizar cálculos en paralelo.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

typedef struct {
    double num1;
    double num2;
} Numbers;

void *suma(void *args) {
    Numbers *op = (Numbers *) args;
    printf("Suma: %.2f + %.2f = %.2f\n", op->num1, op->num2, op->
   num1 + op->num2);
```

```
    return NULL;
}

void *resta(void *args) {
    Numbers *op = (Numbers *) args;
    printf("Resta: %.2f - %.2f = %.2f\n", op->num1, op->num2, op->
   num1 - op->num2);
    return NULL;
}

void *multiplicacion(void *args) {
    Numbers *op = (Numbers *) args;
    printf("Multiplicaci n: %.2f * %.2f = %.2f\n", op->num1, op->
   num2, op->num1 * op->num2);
    return NULL;
}

void *division(void *args) {
    Numbers *op = (Numbers *) args;
    if (op->num2 != 0) {
        printf("Divisi n: %.2f / %.2f = %.2f\n", op->num1, op->
   num2, op->num1 / op->num2);
    } else {
        printf("Divisi n: Error, divisi n por cero.\n");
    }
    return NULL;
}

int main() {
    pthread_t hilo_suma, hilo_resta, hilo_multiplicacion,
   hilo_division;
    Numbers nu;

    printf("Ingrese dos n meros: ");
    scanf("%lf %lf", &nu.num1, &nu.num2);

    pthread_create(&hilo_suma, NULL, suma, &nu);
    pthread_create(&hilo_resta, NULL, resta, &nu);
    pthread_create(&hilo_multiplicacion, NULL, multiplicacion, &nu)
   ;
    pthread_create(&hilo_division, NULL, division, &nu);

    pthread_join(hilo_suma, NULL);
    pthread_join(hilo_resta, NULL);
    pthread_join(hilo_multiplicacion, NULL);
    pthread_join(hilo_division, NULL);

    return 0;
}
```

Listing 3: Código en C con hilos para imprimir mensajes intercalados





1. **Inicio del Programa (`main`)**
   - Se declaran cuatro variables de tipo `pthread_t` para manejar los hilos, y una variable `Numbers` llamada `nu`.
   - Se solicita que se ingresen dos números, los cuales se almacenan en `nu.num1` y `nu.num2`.
2. **Creación de Hilos para Operaciones**
   - Se crean cuatro hilos para realizar las operaciones de suma, resta, multiplicación y división, pasando como argumento un puntero a la estructura `nu`.
3. **Esperar la Finalización de los Hilos**
   - Se usa `pthread_join` para cada hilo creado, lo que asegura que el programa principal espere la finalización de cada operación antes de continuar. Esto garantiza que los resultados de todas las operaciones se impriman antes de que el programa finalice.
4. **Finalización del Programa**
   - Una vez que todos los hilos han completado sus operaciones y el programa principal ha esperado a cada uno, el programa termina.