UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

ELO 321
Teoría de Sistemas Operativos
Tarea #01
Octubre 11, 2020

DEPARTAMENTO DE ELECTRONICA

Semestre 2020 - 2

Profesor: Ioannis Vourkas

Ayudante: Alfonso Cortés

Contenidos

Los contenidos principales a tratar son el manejo de *llamadas a sistema*, el manejo de *hilos* y *procesos*, y la *comunicación* entre procesos.

Objetivos

- Practicar el desarrollo de código C que realiza llamadas a sistemas y crea procesos e hilos.
- Familiarizarse con el desarrollo modular/incremental de código C en OS Linux.
- Comprender los beneficios de aplicaciones multi-hilo (multi-threaded)

Descripción del trabajo a realizar

Parte A: ***Procesos*** (**45 Pts**)

La conjetura enunciada por el matemático ***Lothar Collatz*** en 1937, dice que, si tenemos cualquier número entero positivo ***n*** y aplicamos el siguiente algoritmo, la sucesión de números generada siempre alcanzará al número 1:

- Si el número es par, se divide por 2.
- Si el número es impar, se multiplica por 3 y se le suma 1.

$$f(n) = \begin{cases} \frac{n}{2}, & \text{si } n \text{ es par} \\ 3n+1, & \text{si } n \text{ es impar} \end{cases}$$

Por ejemplo, empezando en ***n*** = 11, la sucesión es: 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1, mientras que si empezamos con ***n*** = 27, la sucesión tiene 111 elementos! Más información encontrará en (https://es.wikipedia.org/wiki/Conjetura_de_Collatz).

Se solicita generar un programa en C que implementa lo siguiente:

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

ELO 321
Teoría de Sistemas Operativos
Tarea #01
Octubre 11, 2020

DEPARTAMENTO DE ELECTRONICA

0. Recibe como **argumento desde línea de comando** un número entero positivo, lo almacena en una variable local y, antes de proceder, realiza un control sobre su valor para validar que se trata de un número positivo.

De no ser así, el programa debe terminar su ejecución. De lo contrario, se realiza lo siguiente:

1. Genera uno (o más) proceso(s) hijo(s), según sea necesario, a fin de **ejecutar una(s) llamada(s) a sistema** e imprimir por pantalla **la hora y fecha** de ejecución del programa. El proceso **padre debe esperar** que el/los proceso(s) hijo(s) termine(n). Para una lista de *system calls* revisen la siguiente página web (https://man7.org/linux/man-pages/man2/syscalls.2.html).

2. Establece un espacio de **memoria compartida** (ver ejemplos POSIX en clases).

3. Genera un proceso hijo el cual ejecuta repetitivamente la siguiente función:

```
int sucesion_Collatz (unsigned int n);
```

la cual recibe el número positivo como argumento y retorna el siguiente número generado de la sucesión Collatz, implementando el algoritmo antes mencionado.

El **proceso hijo debe escribir al espacio de memoria compartida** cada nuevo número generado hasta llegar al No. 1, que es cuando debería terminar su ejecución.

4. El proceso **padre debe esperar** que el proceso hijo termine su ejecución antes de proceder. Luego, debe imprimir por pantalla la sucesión de números generada por el proceso hijo, leyendo desde el espacio de memoria compartida.

5. Al final, el proceso padre debe eliminar el espacio de memoria compartida.

6. En el **punto 4**, agregue en su programa un llamado a `sleep()` justo antes de `wait()`, pasando como argumento la cantidad de "**segundos**" que el proceso padre debe permanecer inactivo, de modo de asegurar que el proceso hijo pase a ser un "zombie".

Ejecute nuevamente su programa desde la línea de comando agregando al final del comando el operador & para ejecutarlo en el background. A continuación, ejecutando el comando `ps-l` podrá identificar el proceso hijo y comprobar que efectivamente se mantiene en el sistema como **zombie**

durante la cantidad de segundos que Ud. indicó en `sleep()`, antes que el proceso padre retome su ejecución. Explique cómo identificó dicho proceso y **agregue un "pantallazo" a su report final**.

Parte B: ***Hilos*** (**55 Pts**)

Se solicita generar un programa en C utilizando Pthreads API, que implementa lo siguiente:

0. Tiene las siguientes variables definidas globalmente:

```
float avg_value;
int min_value;
int max_value;
```

y un arreglo global de 50 números enteros:

```
int numArray [50];
```

el cual su programa lo debe inicializar a con valores aleatorios en el rango de 1 a 100.

1. Luego el programa **genera 3 hilos (threads)** cuyo propósito será calcular diferentes propiedades estadísticas de los valores almacenados en `numArray[]`. En particular:

- el primer thread calculará el **valor promedio**
- el segundo thread buscará el **valor mínimo**
- el tercer thread buscará el **valor máximo**

Para esto, cada thread ejecutará la función que corresponde de las que se indican a continuación, las cuales serán implementadas en su programa con el siguiente prototipo:

```
void* return_Avg(void*);
void* return_Min(void*);
void* return_Max(void*);
```

Ya que el proceso padre y los threads comparten variables globales, dichas funciones deben atravesar el arreglo `numArray[]` y actualizar el valor de las tres variables globales.

En la definición de dichas funciones, los threads que las ejecutan deben imprimir por pantalla un mensaje identificando la operación realizada y su propio **ID** (para eso, ver documentación de `pthread_self()`), en la siguiente forma, justo antes de terminar:

```
Thread ID: #### calculó promedio.
```

2. El proceso padre debe almacenar en un arreglo local:

```
pthread_t threadID [3];
```

los ID de todos los threads generados y **debe esperar (`join`) que todos los threads terminen** su ejecución, para luego imprimir por pantalla los resultados.

3. **Modifique su programa** haciendo ahora que las 3 variables (`avg_value`, `min_value`, `max_value`) sean locales en `main()` (el proceso padre) y que cada función ejecutada por los threads reciba como argumento la variable en cuyo valor debe almacenar el resultado calculado. Revisen la documentación de `pthread_create()` en (https://man7.org).

Indicación

Las tareas se realizan **en grupos de 2 personas como máximo**. Cualquier acción que pueda beneficiar de forma injusta la calificación de su tarea está prohibida, incluyendo la presentación de cualquier componente que no es de su autoría, o la facilitación de esto para otros. Es aceptable discutir -*en líneas generales*- los métodos y resultados con sus compañeros, pero **se prohíbe compartir soluciones de código. Utilizar código de internet que no es de su autoría, también es considerado plagio, a menos que se indique la fuente**.

Consideraciones y Formato de Entrega

- Utilice **/*comentarios*/** para documentar lo que se hace en cada etapa de su código. En la parte superior de su(s) archivo(s) de código incluir lo siguiente:

```
/*
* @file         : <nombre del archivo: por ejemplo, "tarea01.c">
* @author       : <integrantes del grupo, en líneas diferentes>
* @date         : 11/10/2020
* @brief        : Código para tarea 01 en ELO 321, semestre 2020-2
*/
```

- El código deberá estar perfectamente *indentado* **con espacios**, no tabuladores
- Toda tarea debe ser correctamente **compilada** y ejecutada **en el servidor Aragorn (**tareas que **no compilan** se evaluarán cualitativamente y **tendrán como máximo nota 54)**.
    - Asegúrense de comprobar con anticipación que tienen acceso al servidor Aragorn.
    - **Prioricen entregar algo funcional**, aunque incompleto, que entregar algo que no compila.

- Para la entrega envíe **un único archivo** comprimido (`.zip, .rar, .tar.gz`, etc.) que contenga sus archivos `.c, .h` utilizados, más cualquier archivo adicional que de respuesta a las preguntas de la tarea, y un archivo **`README.txt`** donde se especifican:
    - los **datos de los integrantes del grupo** (nombres, apellidos, ROL USM, y emails)
    - qué es cada archivo de su programa, cómo debe ser compilado y cómo debe ser ejecutado

La tarea se entrega vía **AULA** hasta: **viernes 06 de noviembre 2020, 23:55hrs**