Prácticas de Sistemas Operativos

Práctica 1
Gestión de procesos y threads

Objetivos:

- **Practicar la gestión de procesos y threads concurrentes**

Descripción:

- **Desarrollar un programa que cree varios procesos concurrentes, que escriban periódicamente mensajes diferentes en la pantalla**
- **Hacer lo mismo con threads periódicos. Pasar a cada thread como parámetros el periodo y el mensaje a escribir. Los threads se crearán en estado "independiente"**

Práctica 2
Ficheros

UNIVERSIDAD DE CANTABRIA

Objetivos

- **Hacer operaciones simples con ficheros, usando las llamadas al sistema operativo**

Descripción

- **Escribir un programa que copie los contenidos de un fichero en otro. Se pasarán los nombres como argumentos al programa.**
- **Modificarlo para que ponga el estado del fichero generado según los parámetros de entrada que se le pasen los permisos del usuario (rwx)**
- **Asimismo, deberá presentar información del estado de los ficheros (original y copia) en pantalla (fechas, tamaño, etc.)**

UNIVERSIDAD DE CANTABRIA

Práctica 3

Objetivos

- Practicar con la sincronización de acceso mutuamente exclusivo.

Descripción

- Escribir un programa con tres *threads* que se sincronicen para acceder de manera mutuamente exclusiva a un conjunto de datos que representan las coordenadas `x,y,z` de un cuerpo en el espacio
- Uno de los *threads* deberá modificar los datos de manera periódica según las siguientes ecuaciones (una recta):

$$x = x + 0,1 \qquad y = 3x + 8 \qquad z = 4x - 2y + 3$$

Práctica 3 (cont.)

- Otro *thread* consulta esa información y muestra `x,y,z` y también `(y-3x)`, que debe ser 8
- Otro *thread* consulta esa información y muestra `x ,y,z` y también `(z-4x+2y)`, que debe ser 3
- Para la protección de los datos se usará un *mutex*.
- El programa se ejecutará con y sin protección, y así podrá observarse la inconsistencia del conjunto de datos cuando no hay protección
- Puede utilizarse el "come tiempos" que se encuentra en los ficheros `load.c` y `load.h` disponibles en el servidor. Cópialos en tu directorio de trabajo.

Práctica 4

Objetivo:

- Practicar con la sincronización de espera.

Descripción:

- Escribir un programa con un *thread* productor de datos y un *thread* consumidor de estos datos. Los datos serán strings.
- Tanto el productor como el consumidor tendrán tiempos de ejecución aleatorios dentro de un rango (usar `load.h`)
- Utilizarán una variable compartida para la comunicación
  - contiene un string
  - y un booleano que dice si hay un dato válido o no
- Utilizarán un mutex para acceder a la variable

Práctica 4 (cont.)

- Utilizarán dos variables condicionales.
  - Una para que el productor espere cuando la variable compartida está ocupada
  - Otra para que el consumidor espere cuando la variable compartida está vacía.
- Una vez realizado el programa, jugar con él variando los rangos de los tiempos; practicar con un productor muy rápido o muy lento, consumidores iguales, o uno más lento, o más rápido, etc.

Práctica 5

Objetivo:

- Practicar con la recepción y tratamiento de señales para notificar eventos

Descripción:

- Escribir un programa que responda a todas las señales desde un *thread* que actúa como manejador de señal, poniendo un mensaje en la pantalla que indique qué señal se recibió. El programa principal se queda dormido.
- Probar el programa enviándole señales desde una *shell* (con la orden `kill -num_señal pid`)

Nota: Según la versión de Linux se pueden crear (incorrectamente) varios procesos si hay manejadores de señal.

UNIVERSIDAD DE CANTABRIA

Práctica 6

Objetivo:

- Averiguar qué tipo de planificación por omisión utilizan los threads

Descripción:

- Escribir un programa que cree dos threads periódicos concurrentes. A cada thread se le pasará como parámetro:
  - periodo
  - tiempo de ejecución
  - número de identificación
- Para hacer threads periódicos se usará la función `clock_nanosleep()`
  - inicialmente se pueden hacer pseudo-periódicos con `sleep()`

Práctica 6 (cont.)

- Para simular el tiempo de ejecución se usa la función `eat()` suministrada
- Cada décima de segundo de tiempo de ejecución consumido se pondrá un mensaje en la pantalla con el identificador del thread
- También se pondrán mensajes al inicio y final del periodo
- A partir de la secuencia de ejecución dibujar un diagrama temporal de la ejecución
- Tratar de inferir a partir del diagrama el tipo de planificación (expulsora o no; cíclica, FIFO, ...); probar con dos combinaciones diferentes de periodos y tiempos de ejecución

Práctica 6 (cont.)

Compilar y ejecutar el programa desarrollado para Linux en el sistema operativo MaRTE OS sobre máquina desnuda:

- **Hacer las mismas pruebas que en el caso anterior**

Para ello:

- **Ajustar la variable PATH ejecutando**
  - `. /usr/local/marte-1.9/martevars.sh`
- **Compilar el programa con `mgcc`**
- **Copiar el programa en el servidor**
  - `cp el_programa /net/lctrserver/mprogram_HOSTNAME`
- **Encender el computador (PC industrial)**

UNIVERSIDAD DE CANTABRIA

Práctica 7

Objetivo:

- Observar el fenómeno de la inversión de prioridad e implementar un mecanismo para evitarla.

Descripción:

- Escribir un programa que cree varios *threads* periódicos planificados por prioridades fijas, y que se sincronizan para utilizar un recurso compartido.
- Inicialmente se protegerá este recurso compartido por medio de un *mutex* convencional.
- Los *threads* calcularán sus tiempos de respuesta de peor caso y cada vez que se produzca un tiempo peor lo pintarán en la pantalla.

Práctica 7 (cont.)

- Eligiendo de manera adecuada los tiempos de ejecución y el instante de comienzo de cada tarea se podrá observar el fenómeno de la inversión de prioridad no acotada, por el que una tarea de alta prioridad tiene que esperar a la finalización de una tarea de baja prioridad.
- A continuación se modificará el programa anterior que sufría inversión de prioridad para evitarla por medio del uso de un *mutex*:
  - Usar el protocolo de protección de prioridad.
  - Observar cómo la inversión de prioridad se minimiza, y el *thread* de alta prioridad tiene un tiempo de respuesta mucho menor.
- Comprobar el comportamiento en MaRTE OS

Práctica 8

Objetivo:

- Escribir un *script* sencillo

Descripción:

- Escribir un *script* para ser ejecutado por una *shell*:
  - que permita cambiar de nombre a todos los ficheros de un directorio añadiéndoles a todos una extensión
  - la extensión y el nombre del directorio serán parámetros que se pasen al *script*
- Probarlo creando un directorio de prueba con varios ficheros

Práctica 8 (cont.)

- Escribir un segundo script que invoque al primero para todos los directorios contenidos en el directorio de trabajo
  - Este segundo script tendrá un único argumento: la extensión a añadir al nombre de cada fichero
- Probarlo creando un directorio de prueba con varios directorios, que a su vez tengan varios ficheros