Laboratorio

Synchronization

Ciencias de la Computación VII

El objetivo de este laboratorio es recordar los conceptos sobre Proceso, Thread, paradigmas en programación y analizar analizar su comportamiento.

Documentación:

Como se ha explicado en clase existen Threads dentro del mismo proceso y que no existe un modelo explícito padre-hijo, excepto para el "Main Thread" que contiene toda la información del proceso.

A continuación describimos los elementos de la librería PThreads que utilizaremos en este laboratorio.

Ejemplo createThread.c

Estructura pthread_create:

int pthread_create(tid, attr, function, arg);

pthread_t *tid Descriptor del thread creado

const pthread_attr_t *attr Atributos del thread a crearse. Se envía NULL para utilizar los atributos por defecto.

void *(*function)(void *) Función que será mapeada al hilo (El Runnable)

void *arg Argumentos que se envían a la función. Si no se envían parámetros a la función, se especifica NULL.

Ejemplo joinThread.c

Estructura pthread_join:

int pthread_join(tid, val_ptr);

pthread_t tid Descriptor del Thread a esperar

void **val_ptr Valor de salida devuelto por un thread. Si no se espera ningún valor de retorno se especifica NULL.

Ejemplo mutexThread.c
Inicialización: **int pthread_mutex_init(mutex, attr);**
pthread_mutex_t *mutex Mutex a ser inicializado
const pthread_mutexattr_t *attr Atributos a ser establecidos a mutex
Lock: int pthread_mutex_lock(mutex);
pthread_mutex_t *mutex Mutex que intenta hacer el lock
Release: int pthread_mutex_unlock(mutex);
pthread_mutex_t *mutex Mutex a ser desbloqueado

conditionThread.c
pthread_cond_init Inicializa una variable de condición
int pthread_cond_init(cond, attr);
pthread_cond_t *cond Variable condición a ser inicializada
const pthread_condattr_t *attr Atributos a ser establecidos en la variable de condición. Si se envía NULL, se asumen los atributos por defecto.
pthread_cond_wait El thread duerme hasta que se efectúa un signal a la variable de condición

int pthread_cond_wait(cond, mutex);
pthread_cond_t *cond Variable condición a esperar
pthread_mutex_t *mutex Lock a liberar dentro del wait

pthread_cond_signal Señal que libera la variable de condición
int pthread_cond_signal(cond);
pthread_cond_t *cond Variable condición a la que se le enviará la señal
pthread_cond_broadcast Broadcast que libera la variable de condición
int pthread_cond_broadcast(cond);
pthread_cond_t *cond Variable condición a recibir la señal

Contexto:

La Galileo ha sido tomada por sorpresa por un grupo de Zombies hambrientos. Miembros de una misteriosa compañía de nombre Sombrilla han logrado sellar el acceso a la Universidad, aislando a los Zombies que se encuentran dentro. A pesar de lo acontecido, los Inges de Sistemas Operativos continúan asistiendo a clase para completar el Pintos.

Para ello han construido un puente desde la garita de entrada hasta el salón de clase. De momento solo ha quedado espacio sobre el puente para que pueda ser cruzado en una sola dirección, y como máximo puede soportar hasta 4 personas en fila, si se intenta colocar a más personas se correría el riesgo de que el puente colapse justo sobre un grupo de Zombies que han logrado posicionarse por debajo el puente esperando a que algún inge resbale y caiga entre ellos.

Instrucciones:

Implemente un sistema que permita controlar el tráfico sobre el puente, tomando en cuenta que hay inges que entran y salen del edificio en intervalos de **tiempo aleatorios**, y que el puente solo puede soportar a un **máximo de 4 personas**. Debe sincronizar a los inges de manera tal que no existan problemas de circulación sobre el puente, el tráfico debe moverse en un solo sentido a la vez. Además cada Inge tiene un tiempo de cruzar el puente. Si el primer Inge que comenzó a "cruzar" el puente aún está cruzando y llega otro que va a la misma dirección y no supera los 4 puede comenzar a cruzar.

Formato en Consola: (como ejemplo los Inges van en orden)

```
C:\user\CC7> make bridge
¿Cuántas personas cruzaran el puente?  9
Inge 01 camina hacia la Derecha
Inge 02 camina hacia la Derecha
Inge 03 camina hacia la Izquierda
Inge 04 camina hacia la Izquierda
Inge 05 camina hacia la Izquierda
Inge 06 camina hacia la Izquierda
Inge 07 camina hacia la Izquierda
Inge 08 camina hacia la Derecha
Inge 09 camina hacia la Derecha

Llega Inge 01 a Cola Derecha
Cola Derecha: Inge 01
Cola Izquierda:
Inge 01 cruza puente

Llega Inge 02 a Cola Derecha
Cola Derecha: Inge 01, Inge 02
Cola Izquierda:
Inge 01 cruza puente
Inge 02 cruza puente

Llega Inge 03 a Cola Izquierda
Cola Derecha: Inge 01, Inge 02
Cola Izquierda: Inge 03
Inge 01 cruza puente
Inge 02 cruza puente

Llega Inge 04 a Cola Izquierda
Cola Derecha: Inge 01, Inge 02
Cola Izquierda: Inge 03, Inge 04
Inge 01 cruza puente
Inge 02 cruza puente

Llega Inge 05 a Cola Izquierda
Cola Derecha:
Cola Izquierda: Inge 03, Inge 04, Inge 05
Inge 01 sale puente
Inge 02 sale puente

Llega Inge 06 a Cola Izquierda
Cola Derecha:
Cola Izquierda: Inge 03, Inge 04, Inge 05, Inge 06
Inge 03 cruza puente
Inge 04 cruza puente
Inge 05 cruza puente
Inge 06 cruza puente

Llega Inge 07 a Cola Izquierda
Cola Derecha:
Cola Izquierda: Inge 03, Inge 04, Inge 05, Inge 06, Inge 07
Inge 03 cruza puente
Inge 04 cruza puente
Inge 05 cruza puente
Inge 06 cruza puente

Llega Inge 08 a Cola Derecha
Cola Derecha: Inge 08
Cola Izquierda: Inge 03, Inge 04, Inge 05, Inge 06, Inge 07
Inge 03 cruza puente
Inge 04 cruza puente
Inge 05 cruza puente
Inge 06 cruza puente

Llega Inge 09 a Cola Derecha
Cola Derecha: Inge 08, Inge 09
Cola Izquierda: Inge 07
Inge 03 sale puente
Inge 04 sale puente
Inge 05 sale puente
Inge 06 sale puente
```

Tip:
Implemente un **brige** que contenga métodos **accessBridge** y **exitBridge** Si una persona quiere cruzar el puente debe llamar al método **accessBridge** donde dirección es **0** si la persona va hacia la derecha y **1** si la persona va hacia la izquierda. Este método no retorna hasta que se haya permitido el acceso de una persona al puente. Si una persona recibe la autorización para cruzar, esta se tomará un tiempo aleatorio entre 1 y 3 segundos para recorrer el puente. Luego de cruzar el puente, la persona debe llamar al método **exitBridge**, que eventualmente permitirá que otras personas puedan intentar cruzar, haga un Sinal a todos los que están esperando y así decidir cuál dirección ganará.

Cada persona está representada por un Thread. El **brige** debe solicitar al usuario la cantidad de personas que intentan cruzar, y aleatoriamente asignará la dirección de cruce, indicando en pantalla los detalles. Un thread debe esperar un tiempo aleatorio entre 0 y 3 segundos antes de que intente llamar al método **accessBridge**

Actividades:
Coloque sus respuestas en un archivo .txt (**respuesta.txt**)
Realice un ensayo acerca de las diferencias funcionales y de concepto que encuentra entre Java y C sobre los siguientes temas:

Creación y definición de un Thread.
Implementación de Locks
Implementación de Variables de Condición.

Entrega:

- El laboratorio debe ser entregado por medio del GES, con todos los archivos en un **ZIP**. No se calificarán laboratorios entregados tarde o por medio de URL externo.
- El laboratorio debe de contener un archivo **Makefile** ya sea que requiera o no cargar librerías para realizar la compilación.
- El laboratorio puede tener una calificación de cero si no compila o de -100 si se detecta plagio.

Para este Laboratorio se adjunta un archivo ZIP con ejemplos de código compilable, puede hacer uso de ello para realizar su laboratorio.