TEC | Tecnológico de Costa Rica

Área Académica de Ingeniería en Computadores

Arquitectura de Computadores II

Taller 1

Profesor:
Luis Alonso Barboza Artavia

Estudiante:
Jose Ignacio Granados Marín

Grupo 1

Semestre II 2022

Investigación

1. Investigue posibles métodos (bibliotecas, apis, etc) para el uso de hilos bajo el sistema operativo GNU Linux. (Por ejemplo, pthreads).

Una de las bibliotecas más utilizadas para la implementación de hilos es la librería pthreads de POSIX, la cual corresponde a un API para C o C++, basado en estádares, que permite la implementación de diversos subprocesos dentro de un procesador. Dicha herramienta permite la creación de flujos de ejecución concurrentes, acelerando de esta manera el procesamiento paralelo o distribuido de procesos [1]. Por su parte, algunos de los métodos que presenta esta biblioteca, son [2]:

- pthread_create: se utiliza para crear un nuevo hilo.
- pthread_exit: se utiliza para terminar un hilo determinado.
- pthread_join: se utiliza para esperar la terminación de un hilo.
- pthread_self: se utiliza para obtener el identificador del hilo actual.
- pthread_equal: compara si dos hilos son iguales o no.
- pthread_cancel: se utiliza para enviar una solicitud de cancelación a un hilo.
- pthread_detach: se utiliza para separar un hilo.

Por otra parte, la biblioteca de Java Threads consiste en una API que permite crear y gestionar múltiples hilos directamente como programas codificados en el lenguaje de programación de Java [3]. Dentro de los métodos que brinda esta biblioteca, se tienen los siguientes [4]:

- start: inicia el hilo.
- getState: devuelve el estado del subproceso.
- getName: devuelve el nombre del subproceso.
- getPriority: devuelve la prioridad del hilo.
- sleep: deteniene el hilo durante el tiempo especificado.
- join: deteniene el subproceso actual hasta que finalice el subproceso llamado.
- isAlive: comprueba si el hilo está vivo.

2. ¿Qué es el concepto de mutex en multiprogramación y que busca hacer?

El concepto de exclusión mutua (abreviado como mutex por mutual exclusión en inglés) consiste en que un solo proceso excluye temporalmente a todos los demás para utilizar un recurso compartido, de tal manera que se garantice la integridad del sistema en cuestión [5]. Los algoritmos codificados por medio de esta técnica, buscan evitar que fragmentos de código (secciones críticas) accedan al mismo tiempo a recursos que no deben ser compartidos [6].

3. ¿Qué sucede cuando dos hilos quieren utilizar el mismo recurso? ¿Cómo se manejan estos casos?

Aquellos hilos que comparten los mismos recursos, en conjunto con dichos recursos, conforman un determinado proceso. Cuando un hilo modifica un dato en la memoria, los otros hilos acceden dicho valor modificado inmediatamente después de haber sido escrito. Sin embargo, cuando dos hilos necesitan utilizar el mismo recurso, se recurre a la técnica de espera por parte de algún hilo, la cual consiste en “dormir” un hilo hasta que el otro hilo termine de utilizar el recurso solicitado [7].

Desarrollo

Ejercicio 1

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

// constants definition
#define MIN 0
#define MAX 255
#define SIZE 20
#define DELAY 10000 // 10000 microseconds = 10 milliseconds
#define CONSTANT 123

// this function generates 'size' 8bit random numbers between 'min' and
// 'max' range and stores them in an array
void writeRandomNumbers(int *randomArray, int min, int max, int size) {

    int i;

    for(i = 0; i < size; i++) {

        randomArray[i] = (rand() % (max - min + 1)) + min;

        printf("%d \n", randomArray[i]);

    }

    printf("\n");

}

// this function read 'size' 8 bit random numbers generated by the first
// thread
void readRandomNumbers(int *randomArray, int size, int constant) {

    int i;

    for(i = 0; i < size; i++) {

        int number = randomArray[i] & constant;

        char charNumber = number + '0';

        printf("%c \n", charNumber);

    }

}

// this functions belogns to thread 1
void *thread1Function(void *voidRandomArray) {

    int* randomArray;
    randomArray = (int*)voidRandomArray;

    while(1) {

        writeRandomNumbers(randomArray, MIN, MAX, SIZE);

        printf("********** \n\n");

        usleep(10000);

    }

}

// this functions belogns to thread 2
void *thread2Function(void *voidRandomArray) {

    int* randomArray;
    randomArray = (int*)voidRandomArray;

    while(1) {

        readRandomNumbers(randomArray, SIZE, CONSTANT);

        printf("********** \n\n");

        usleep(10000);

    }

}

int main() {

    int *randomArray;
    randomArray = (int*)calloc(SIZE, sizeof(int));

    pthread_t thread1;
    pthread_t thread2;

    int write = pthread_create(&thread1, NULL, thread1Function, (void*) randomArray);
    int read = pthread_create(&thread2, NULL, thread2Function, (void*) randomArray);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;

}
```

Ejercicio 2

```c
// libraries
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

// global variables
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int sharedCounter = 0;

// this functions belogns to both threads
void *threadsFunction() {

    printf("Inicio del hilo\n");

    // lock resource
    pthread_mutex_lock(&mutex);

    sharedCounter ++;

    printf("Contador compartido = %d\n", sharedCounter);

    // unlock resource
    pthread_mutex_unlock(&mutex);

    printf("Fin del hilo\n\n");

}

int main() {

    pthread_t thread1;
    pthread_t thread2;

    int write1 = pthread_create(&thread1, NULL, threadsFunction, NULL);
    int write2 = pthread_create(&thread2, NULL, threadsFunction, NULL);

    pthread_join(thread1, NULL);
    pthread_join(thread2, NULL);

    return 0;

}
```

En este ejercicio en particular, se tienen 2 hilos que quieren modificar el mismo recurso compartido. Sin embargo, esto no es posible porque es un único recurso y solo puede ser asignado a un solo proceso. Dado lo anterior, se emplea un mutex, el cual se encarga de asignar un candado o cerrojo al contador mientras un hilo lo esté utilizando, de tal manera que, en el momento en que el otro hilo quiera acceder al recurso, el bloqueo del mutex pondrá en espera a dicho hilo hasta que el recurso esté disponible para ser modificado.

Referencias

[1] GeeksforGeeks. (2021, 16 julio). *POSIX Threads in OS*. Recuperado 13 de septiembre de 2022, de https://www.geeksforgeeks.org/posix-threads-in-os/

[2] GeeksforGeeks. (2020, 23 junio). *Thread functions in C/C++*. Recuperado 13 de septiembre de 2022, de https://www.geeksforgeeks.org/thread-functions-in-c-c/

[3] Tutorials Point. (2019). *What are thread libraries?* Recuperado 14 de septiembre de 2022, de https://www.tutorialspoint.com/what-are-thread-libraries#

[4] Software Testing Help. (2022, 7 agosto). *Subprocesos Java Con Métodos Y Ciclo De Vida*. (2022, 7 agosto). Recuperado 14 de septiembre de 2022, de https://www.softwaretestinghelp.com/java/java-threads/

[5] Velázquez, D. (2022b, enero 22). *Exclusión mutua*. WebProgramacion Consultoría Informática. Recuperado 14 de septiembre de 2022, de https://webprogramacion.com/exclusion-mutua/

[6] Sandoval, K. (2011). *Sistemas Operativos, procesos concurrentes – Unidad III*. (s. f.-b). Recuperado 14 de septiembre de 2022, de https://www.aiu.edu/spanish/publications/student/spanish/180-207/SISTEMAS-OPERATIVOS-PROCESOS-CONCURRENTES-Unidad-III.html

[7] Respuestas Rápidas. (2021, 19 abril). *¿Qué sucede cuando dos hilos quieren utilizar el mismo recurso? – RESPUESTASRAPIDAS*. Recuperado 14 de septiembre de 2022, de https://respuestasrapidas.com.mx/que-sucede-cuando-dos-hilos-quieren-utilizar-el-mismo-recurso/