Algoritmos de Dekker y Peterson

# Algoritmo de Dekker

#include <stdio.h>

#include <pthread.h>

#define TRUE 1

#define FALSE 0

volatile \_Bool threadAWant, threadBWant;

volatile int turn = 0;

volatile int count = 0;

volatile int limit = 0; //número de veces máximo que cada thread suma 1 a count, N = 100, 1000, 10000....

//funcion del primer thread

void\* functionThreadA(void \*arg) {

int iteration = 0;

pthread\_t tid = pthread\_self();

printf("Thread %lu\n", tid);

while (1) {

//el thread "quiere" entrar a la sección crítica

threadAWant = TRUE;

\_\_sync\_synchronize();

while (threadBWant == TRUE) {

if (turn == 1) {

threadAWant = FALSE;

while (turn != 0) {

}

threadAWant = TRUE;

}

\_\_sync\_synchronize();

}

//inicia seccion crítica

++count;

printf("In thread #%ld -> %d;\n", tid, count);

++iteration;

//fin de la seccion crÃ­tica

turn = 1;

threadAWant = FALSE;

\_\_sync\_synchronize();

if (iteration == limit)

return 0;

}

}

//funcion del segundo thread

void\* functionThreadB(void \*arg) {

long iteration = 0;

pthread\_t tid = pthread\_self();

printf("Thread %lu\n", tid);

while (1) {

//el thread "quiere" entrar a la sección crítica

threadBWant = TRUE;

\_\_sync\_synchronize();

while (threadAWant == TRUE) {

if (turn != 1) {

threadBWant = FALSE;

while (turn != 1) {

}

threadBWant = TRUE;

}

\_\_sync\_synchronize();

}

//inicia seccion crÃ­tica

++count;

printf("tid #%ld -> %d;\n", tid, count);

++iteration;

//termina sección crítica

turn = 0;

threadBWant = FALSE;

\_\_sync\_synchronize();

if (iteration == limit)

return 0;

}

}

int main(int argc, char \*argv[]) {

threadAWant = 0;

threadBWant = 0;

pthread\_t threadA, threadB;

printf("\nnumero de veces de suma (N): ");

scanf("%d", &limit);

pthread\_create(&threadA, 0, functionThreadA, 0);

pthread\_create(&threadB, 0, functionThreadB, 0);

pthread\_join(threadA, 0);

pthread\_join(threadB, 0);

printf("\nValor final de count: %d\n", count);

return 0;

}