|  |
| --- |
| #include<stdio.h> |
|  | #include<pthread.h> |
|  | #include<stdlib.h> |
|  | #include <unistd.h> |
|  |  |
|  |  |
|  | pthread\_mutex\_t lock = PTHREAD\_MUTEX\_INITIALIZER; |
|  | pthread\_cond\_t finish\_cond = PTHREAD\_COND\_INITIALIZER; |
|  | int barrier = 0; |
|  | int thread\_count; |
|  | int barrier\_size; |
|  | int counter=0; |
|  | int invoke\_barrier = 0; |
|  |  |
|  |  |
|  |  |
|  | /\* |
|  | \* params : number of threads a process is creating. |
|  | \* returns : none. |
|  | \* |
|  | \* Initialize barrier with total number of threads. |
|  | \*/ |
|  | void barrier\_init(int n\_threads) |
|  | { |
|  | if ( thread\_count < barrier\_size ) { barrier = thread\_count; return; } |
|  | barrier = n\_threads; |
|  | } |
|  |  |
|  |  |
|  |  |
|  | /\* |
|  | \* params: none. |
|  | \* returns: -1 on failure, 0 on success. |
|  | \* decrement the count by 1. |
|  | \* |
|  | \*/ |
|  | int decrement() |
|  | { |
|  | if (barrier == 0) { |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  | if(pthread\_mutex\_lock(&lock) != 0) |
|  | { |
|  | perror("Failed to take lock."); |
|  | return -1; |
|  | } |
|  |  |
|  | barrier--; |
|  |  |
|  | if(pthread\_mutex\_unlock(&lock) != 0) |
|  | { |
|  | perror("Failed to unlock."); |
|  | return -1; |
|  | } |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | /\* |
|  | \* params: none. |
|  | \* returns: int : 0 on sucess, -1 on failure. |
|  | \* |
|  | \* |
|  | \* wait for other threads to complete. |
|  | \*/ |
|  | int wait\_barrier() |
|  | { |
|  | if(decrement() < 0) |
|  | { |
|  | return -1; |
|  | } |
|  |  |
|  | while (barrier) |
|  | { |
|  | if(pthread\_mutex\_lock(&lock) != 0) |
|  | { |
|  | perror("\n Error in locking mutex"); |
|  | return -1; |
|  | } |
|  |  |
|  | if(pthread\_cond\_wait(&finish\_cond, &lock) != 0) |
|  | { |
|  | perror("\n Error in cond wait."); |
|  | return -1; |
|  | } |
|  | } |
|  |  |
|  | /\* |
|  | \* last thread will execute this. |
|  | \*/ |
|  | if(0 == barrier) |
|  | { |
|  | if(pthread\_mutex\_unlock(&lock) != 0) |
|  | { |
|  | perror("\n Error in locking mutex"); |
|  | return -1; |
|  | } |
|  | if(pthread\_cond\_signal(&finish\_cond) != 0) |
|  | { |
|  | perror("\n Error while signaling."); |
|  | return -1; |
|  | } |
|  | } |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  | void \* barrier\_point(void \*numthreads) |
|  | { |
|  |  |
|  | int r = rand() % 5; |
|  |  |
|  | printf("\nThread %d \nPerforming init task of length %d sec\n",++counter,r); |
|  | sleep(r); |
|  |  |
|  | wait\_barrier(); |
|  | if (barrier\_size!=0) { |
|  | if ((thread\_count - (invoke\_barrier++) ) % barrier\_size == 0) { |
|  | printf("\nBarrier is Released\n"); |
|  | } |
|  | printf("\nI am task after barrier\n"); |
|  |  |
|  | } |
|  | //printf("Thread completed job.\n"); |
|  |  |
|  | return NULL; |
|  | } |
|  |  |
|  |  |
|  |  |
|  |  |
|  | int main() |
|  | { |
|  |  |
|  | printf("Enter Barrier Size\n"); |
|  | scanf("%d", &barrier\_size); |
|  |  |
|  | printf("Enter no. of thread\n"); |
|  | scanf("%d", &thread\_count); |
|  |  |
|  | //Checking valid input |
|  |  |
|  | if (barrier\_size>=0 && thread\_count>=0) { |
|  | pthread\_t tid[thread\_count]; |
|  |  |
|  | barrier\_init(barrier\_size); |
|  |  |
|  | for(int i =0; i < thread\_count; i++) |
|  | { |
|  | pthread\_create(&(tid[i]), NULL, &barrier\_point, &thread\_count); |
|  | } |
|  |  |
|  |  |
|  | for(int j = 0; j < thread\_count; j++) |
|  | { |
|  | pthread\_join(tid[j], NULL); |
|  | } |
|  | } |
|  | //when user give wrong input then this section will execute. |
|  | else{ |
|  | printf("You are entering wrong data.\n"); |
|  | main(); |
|  | } |
|  |  |
|  | return 0; |
|  | } |