**MThreadFac.c**

|  |
| --- |
| #include <stdint.h> |
|  | #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <getopt.h> |
|  | #include <stdbool.h> |
|  | #include <limits.h> |
|  | #include <sys/time.h> |
|  | #include <pthread.h> |
|  |  |
|  | pthread\_mutex\_t mut = PTHREAD\_MUTEX\_INITIALIZER; |
|  | static struct option options[] = {{"k", required\_argument, 0, 0}, |
|  | {"mod", required\_argument, 0, 0}, |
|  | {"pnum", required\_argument, 0, 0}, |
|  | {0, 0, 0, 0}}; |
|  |  |
|  | int getArguments(int argc, char \*\*argv, int \*arg) |
|  | { |
|  |  |
|  | while (true) { |
|  | int option\_index = 0; |
|  | int c = getopt\_long(argc, argv, "", options, &option\_index); |
|  |  |
|  | if (c == -1) |
|  | break; |
|  |  |
|  | switch (c) { |
|  | case 0: { |
|  | switch (option\_index) { |
|  | case 0: { |
|  | arg[0] = atoi(optarg); |
|  | if (arg[0] < 0) { |
|  | printf("The factorial argument must be a positive number or 0. Now factorial argument = %d\n", arg[0]); |
|  | return -1; |
|  | } |
|  | break; |
|  | } |
|  |  |
|  | case 1: |
|  | arg[1] = atoi(optarg); |
|  | if (arg[1] < 0) { |
|  | printf("The factorial modul's must be a positive number. Now modul = %d\n", arg[1]); |
|  | return -1; |
|  | } |
|  | break; |
|  |  |
|  | case 2: { |
|  | arg[2] = atoi(optarg); |
|  | if (arg[2] <= 0) { |
|  | printf("Threads\_num must be 1 or more. Now threads\_num = %d\n", arg[2]); |
|  | return -1; |
|  | } |
|  |  |
|  | break; |
|  | } |
|  | } |
|  | break; |
|  | } |
|  |  |
|  | case '?': |
|  | break; |
|  |  |
|  | default: |
|  | printf("getopt returned character code 0%o?\n", c); |
|  | } |
|  | } |
|  | return 0; |
|  | } |
|  |  |
|  | void Multy\_thread\_factorial(int\* arg) |
|  | { |
|  | int n = arg[0]; |
|  | int p = arg[1]; |
|  | int res = 1; |
|  | printf("\nF:%d %d %d",n,p,res); |
|  | while (n > 1) { |
|  | pthread\_mutex\_lock(&mut); |
|  | if (n % p == 0) |
|  | res \*= n/p; |
|  | else |
|  | res \*= n%p; |
|  | n--; |
|  | pthread\_mutex\_unlock(&mut); |
|  | } |
|  | printf("\n%d %d %d",n,p,res); |
|  | arg[2] = res % p; |
|  | } |
|  |  |
|  | int main(int argc, char \*\*argv) { |
|  |  |
|  | int k = -1; |
|  | int mod = -1; |
|  | int pnum = -1; |
|  | int arg[3]; |
|  |  |
|  | if (getArguments(argc, argv, arg)) |
|  | return -1; |
|  |  |
|  | k = arg[0]; |
|  | mod = arg[1]; |
|  | pnum = arg[2]; |
|  |  |
|  |  |
|  | if (optind < argc) { |
|  | printf("Has at least one no option argument\n"); |
|  | return 1; |
|  | } |
|  |  |
|  | if (k == -1 || mod == -1 || pnum == -1 ) { |
|  | printf("Usage: %s --seed \"num\" --array\_size \"num\" --pnum \"num\" \n", |
|  | argv[0]); |
|  | return 1; |
|  | } |
|  |  |
|  | pthread\_t Threads\_mass[pnum]; |
|  |  |
|  | printf("\n%d %d %d",arg[0],arg[1],arg[2]); |
|  | for(int i = 0; i < pnum; i++) |
|  | if (pthread\_create(Threads\_mass + i, NULL, (void \*)Multy\_thread\_factorial, |
|  | (void \*)arg) != 0) { |
|  | perror("pthread\_create"); |
|  | exit(1); |
|  | } |
|  |  |
|  | for(int i = 0; i < pnum; i++) |
|  | if (pthread\_join(Threads\_mass[i], NULL) != 0) { |
|  | perror("pthread\_join"); |
|  | exit(1); |
|  | } |
|  |  |
|  | printf("\n%d!mod(%d) = %d\n", k, mod, arg[2]); |
|  |  |
|  | return 0; |
|  | } |

* © 2021 GitHub, Inc.

|  |
| --- |
| **deadlock.c**  #include <stdint.h> |
|  | #include <stdio.h> |
|  | #include <stdlib.h> |
|  | #include <getopt.h> |
|  | #include <stdbool.h> |
|  | #include <limits.h> |
|  | #include <sys/time.h> |
|  | #include <pthread.h> |
|  |  |
|  | pthread\_mutex\_t mut1 = PTHREAD\_MUTEX\_INITIALIZER; |
|  | pthread\_mutex\_t mut2 = PTHREAD\_MUTEX\_INITIALIZER; |
|  | int \*Acommon = 0; |
|  | int \*Bcommon = 0; |
|  |  |
|  | void F1(int \*a) |
|  | { |
|  | int b = \*a; |
|  | pthread\_mutex\_lock(&mut1); |
|  | printf("\nthread1 takes the mut1"); |
|  | Acommon = &b; |
|  | printf("\nAcommon = %d", \*Acommon); |
|  | pthread\_mutex\_lock(&mut2); |
|  | printf("\nthread1 takes the mut2"); |
|  | Bcommon = &b; |
|  | printf("\nBcommon = %d", \*Bcommon); |
|  | printf("\nthread1 is finishing..."); |
|  | //pthread\_mutex\_unlock(&mut2); |
|  | //pthread\_mutex\_unlock(&mut1); |
|  |  |
|  |  |
|  |  |
|  | } |
|  |  |
|  | void F2(int \*a) |
|  | { |
|  | int b = \*a; |
|  | pthread\_mutex\_lock(&mut2); |
|  |  |
|  | printf("\nthread2 takes the mut2"); |
|  |  |
|  | Bcommon = &b; |
|  |  |
|  | printf("\nBcommon = %d", \*Bcommon); |
|  |  |
|  | pthread\_mutex\_lock(&mut1); |
|  |  |
|  | printf("\nthread2 takes the mut1"); |
|  |  |
|  | Acommon = &b; |
|  |  |
|  | printf("\nAcommon = %d", \*Acommon); |
|  | printf("\nthread2 is finishing..."); |
|  | //pthread\_mutex\_lock(&mut1); |
|  | //pthread\_mutex\_unlock(&mut2); |
|  |  |
|  |  |
|  |  |
|  | } |
|  |  |
|  | int main() |
|  | { |
|  | printf("\na"); |
|  | pthread\_t Threads\_mass[2]; |
|  | int a = 1; |
|  | int b = 1; |
|  |  |
|  | if (pthread\_create(Threads\_mass, NULL, (void \*)F1, |
|  | (void \*)&a) != 0) { |
|  | perror("pthread\_create"); |
|  | exit(1); |
|  | } |
|  |  |
|  | if (pthread\_create(Threads\_mass + 1, NULL, (void \*)F2, |
|  | (void \*)&b) != 0) { |
|  | perror("pthread\_create"); |
|  | exit(1); |
|  | } |
|  |  |
|  | for(int i = 0; i < 2; i++) |
|  | if (pthread\_join(Threads\_mass[i], NULL) != 0) { |
|  | perror("pthread\_join"); |
|  | exit(1); |
|  | } |
|  | printf("\nb"); |
|  | return 0; |
|  | } |

**Mutex**

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* An example source module to accompany... |
|  | \* |
|  | \* "Using POSIX Threads: Programming with Pthreads" |
|  | \* by Brad nichols, Dick Buttlar, Jackie Farrell |
|  | \* O'Reilly & Associates, Inc. |
|  | \* Modified by A.Kostin |
|  | \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* |
|  | \* mutex.c |
|  | \* |
|  | \* Simple multi-threaded example with a mutex lock. |
|  | \*/ |
|  | #include <errno.h> |
|  | #include <pthread.h> |
|  | #include <stdio.h> |
|  | #include <stdlib.h> |
|  |  |
|  | void do\_one\_thing(int \*); |
|  | void do\_another\_thing(int \*); |
|  | void do\_wrap\_up(int); |
|  | int common = 0; /\* A shared variable for two threads \*/ |
|  | int r1 = 0, r2 = 0, r3 = 0; |
|  | pthread\_mutex\_t mut = PTHREAD\_MUTEX\_INITIALIZER; |
|  |  |
|  | int main() { |
|  | pthread\_t thread1, thread2; |
|  |  |
|  | if (pthread\_create(&thread1, NULL, (void \*)do\_one\_thing, |
|  | (void \*)&common) != 0) { |
|  | perror("pthread\_create"); |
|  | exit(1); |
|  | } |
|  |  |
|  | if (pthread\_create(&thread2, NULL, (void \*)do\_another\_thing, |
|  | (void \*)&common) != 0) { |
|  | perror("pthread\_create"); |
|  | exit(1); |
|  | } |
|  |  |
|  | if (pthread\_join(thread1, NULL) != 0) { |
|  | perror("pthread\_join"); |
|  | exit(1); |
|  | } |
|  |  |
|  | if (pthread\_join(thread2, NULL) != 0) { |
|  | perror("pthread\_join"); |
|  | exit(1); |
|  | } |
|  |  |
|  | do\_wrap\_up(common); |
|  |  |
|  | return 0; |
|  | } |
|  |  |
|  | void do\_one\_thing(int \*pnum\_times) { |
|  | int i, j, x; |
|  | unsigned long k; |
|  | int work; |
|  | for (i = 0; i < 50; i++) { |
|  | //pthread\_mutex\_lock(&mut); |
|  | printf("doing one thing\n"); |
|  | work = \*pnum\_times; |
|  | printf("counter = %d\n", work); |
|  | work++; /\* increment, but not write \*/ |
|  | for (k = 0; k < 500000; k++) |
|  | ; /\* long cycle \*/ |
|  | \*pnum\_times = work; /\* write back \*/ |
|  | //pthread\_mutex\_unlock(&mut); |
|  | } |
|  | } |
|  |  |
|  | void do\_another\_thing(int \*pnum\_times) { |
|  | int i, j, x; |
|  | unsigned long k; |
|  | int work; |
|  | for (i = 0; i < 50; i++) { |
|  | //pthread\_mutex\_lock(&mut); |
|  | printf("doing another thing\n"); |
|  | work = \*pnum\_times; |
|  | printf("counter = %d\n", work); |
|  | work++; /\* increment, but not write \*/ |
|  | for (k = 0; k < 500000; k++) |
|  | ; /\* long cycle \*/ |
|  | \*pnum\_times = work; /\* write back \*/ |
|  | //pthread\_mutex\_unlock(&mut); |
|  | } |
|  | } |
|  |  |
|  | void do\_wrap\_up(int counter) { |
|  | int total; |
|  | printf("All done, counter = %d\n", counter); |
|  | } |
|  | /\*при запуске без мьютекса потоки наперегонки инкрементируют |
|  | общую перемунную, но тк циклы выполняются быстро, то оба потока одновременно |
|  | присвают common одно и то же значение. если количество итераций |
|  | вложенного цикла в одной из функций увеличить или уменьшить, то |
|  | один поток будет обгонять другой, а тот поток, который отстает |
|  | будет продолжать инкрементирование с того числа, на котором закончил |
|  | предыдущий поток |
|  | при запуске с мьютексами один поток блокирует общую переменную и не отпускает |
|  | пока не завершится. потом проделывает ту же работу другой поток\*/ |

|  |  |
| --- | --- |
|  | #!/bin/bash |
|  |  |
|  | gcc -c -std=c99 mutex.c |
|  | gcc mutex.o -lpthread -o mutex |
|  | ./mutex |

|  |
| --- |
| #!/bin/bash |
|  |  |
|  |  |
|  | gcc -o MThreadFac -lpthread -std=c99 MThreadFac.c |
|  | ./MThreadFac --k 10 --pnum=10 --mod=17 |

|  |
| --- |
| #!/bin/bash |
|  |  |
|  | gcc -o deadlock -lpthread -std=c99 deadlock.c |
|  | ./deadlock |