

University of Wolverhampton
School of Mathematics and Computer Science
Student Number:
Name:2407710
6CS005 High Performance Computing Week 3 Workshop
Revision on Multithreading
Tasks – Multithreading

You will need to refer to the Week 3 lecture sides in order to complete these tasks.

1. Write a multithreaded C program to print out all the prime numbers between 1 to 10000. Use exactly 3 threads.

```
PS C:\cistuff\HPC\week3> ./output
2357 9631 2371 6311 2377 6317 2381 6323 9643 6329 2383 6337 2389 2393 9649 2399 9661 9677 6343 9679 6353 2411 6359 2417 6361 9689 6367 9697 6373 2423 6379 9719 6389 6397 2437 6421 24
41 6427 9721 6449 2447 6451 9733 6469 9739 2459 9743 2467 9749 2473 9767 2477 9769 2503 6473 9781 2521 6481 9787 6481 9791 6521 9803 9811 2531 9817 2539 9829 6529 9833 2543 6547 2549
6551 2551 6553 2557 6563 2579 6569 9839 6571 2591 6577 9851 6581 2593 9857 2609 9859 6599 9871 6607 9883 6619 6637 6653 6659 6661 9887 9901 2617 9907 2621 9923 2633 9929 9931 9941 9
949 2647 9967 2657 9973 2659 2663 2671 2677 2683 2687 2689 2693 2699 2707 2711 2713 2719 2729 2731 2741 2749 2753 2767 2777 2789 2791 2797 2801 2803 2819 2833 2837 2841 2851 2857 286
1 2879 2887 2897 2903 2909 2917 2927 2939 2953 2957 2963 2969 2971 2999 3001 3111 3019 3023 3037 3041 3049 3061 3067 3079 3083 3089 3109 3119 3121 3137 3163 3167 3169 3181 3187 3191
3203 3209 3217 3221 3229 3251 3253 3257 3259 3271 3299 3301 3307 3313 3319 3323 3329 3331
3 6203 9539 9547 6211 9551 2269 2273 6217 6221 9587 6229 2281 6247 2287 6257 2293 6263 6269 2297 6271 2309 6277 2311 9601 9613 2333 9619 2339 2341 6287 2347 6299 9623 9629 2351 6301
2357 9631 2371 6311 2377 6317 2381 6323 9643 6329 2383 6337 2389 2393 9649 2399 9661 9677 6343 9679 6353 2411 6359 2417 6361 9689 6367 9697 6373 2423 6379 9719 6389 6397 2437 6421 24
41 6427 9721 6449 2447 6451 9733 6469 9739 2459 9743 2467 9749 2473 9767 2477 9769 2503 6473 9781 2521 6481 9787 6481 9791 6521 9803 9811 2531 9817 2539 9829 6529 9833 2543 6547 2549
6551 2551 6553 2557 6563 2579 6569 9839 6571 2591 6577 9851 6581 2593 9857 2609 9859 6599 9871 6607 9883 6619 6637 6653 6659 6661 9887 9901 2617 9907 2621 9923 2633 9929 9931 9941 9
949 2647 9967 2657 9973 2659 2663 2671 2677 2683 2687 2689 2693 2699 2707 2711 2713 2719 2729 2731 2741 2749 2753 2767 2777 2789 2791 2797 2801 2803 2819 2833 2837 2841 2851 2857 286
1 2879 2887 2897 2903 2909 2917 2927 2939 2953 2957 2963 2969 2971 2999 3001 3111 3019 3023 3037 3041 3049 3061 3067 3079 3083 3089 3109 3119 3121 3137 3163 3167 3169 3181 3187 3191
3203 3209 3217 3221 3229 3251 3253 3257 3259 3271 3299 3301 3307 3313 3319 3323 3329 3331
3 6203 9539 9547 6211 9551 2269 2273 6217 6221 9587 6229 2281 6247 2287 6257 2293 6263 6269 2297 6271 2309 6277 2311 9601 9613 2333 9619 2339 2341 6287 2347 6299 9623 9629 2351 6301
2357 9631 2371 6311 2377 6317 2381 6323 9643 6329 2383 6337 2389 2393 9649 2399 9661 9677 6343 9679 6353 2411 6359 2417 6361 9689 6367 9697 6373 2423 6379 9719 6389 6397 2437 6421 24
41 6427 9721 6449 2447 6451 9733 6469 9739 2459 9743 2467 9749 2473 9767 2477 9769 2503 6473 9781 2521 6481 9787 6481 9791 6521 9803 9811 2531 9817 2539 9829 6529 9833 2543 6547 2549
6551 2551 6553 2557 6563 2579 6569 9839 6571 2591 6577 9851 6581 2593 9857 2609 9859 6599 9871 6607 9883 6619 6637 6653 6659 6661 9887 9901 2617 9907 2621 9923 2633 9929 9931 9941 9
949 2647 9967 2657 9973 2659 2663 2671 2677 2683 2687 2689 2693 2699 2707 2711 2713 2719 2729 2731 2741 2749 2753 2767 2777 2789 2791 2797 2801 2803 2819 2833 2837 2841 2851 2857 286
1 2879 2887 2897 2903 2909 2917 2927 2939 2953 2957 2963 2969 2971 2999 3001 3111 3019 3023 3037 3041 3049 3061 3067 3079 3083 3089 3109 3119 3121 3137 3163 3167 3169 3181 3187 3191
3203 3209 3217 3221 3229 3251 3253 3257 3259 3271 3299 3301 3307 3313 3319 3323 3329 3331
all threads finished
```

Explanation:

This program finds prime numbers faster by using three threads instead of one. It divides the numbers from 1 to 10,000 into three equal parts, and each thread checks the numbers in its assigned section. The function that checks if a number is prime works efficiently by handling small cases, skipping even numbers, and only checking up to the square root. Each thread prints the prime numbers it finds. In the main part of the program, the threads are created and then the program waits for all of them to finish. By having multiple threads work at the same time, the program completes the task more quickly, and after all threads are done, it prints a message saying the work is complete.

2. Convert this program to prompt the user for a number and then to create the number of threads the user has specified to find the prime numbers.

```
week3 > C workshop2.c > main()
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <stdlib.h>
4
5
6  #define MAX 10000
7  #define THREAD_COUNT 3
8
9  typedef struct {
10     int start;
11     int end ;
12 }Range;
13
14
15 int isPrime(int n ){
16     if(n <2) return 0;
17     if(n ==2) return 1;
18     if(n%2 ==0) return 0;
19     for(int i =3; i*i <= n; i+=2){
20         if (n%i==0) return 0;
21     }
22     return 1;
23 }
24
25 void* findPrime(void *args){
26     Range *range = (Range *)args;
27     printf("Thread started\n", range->start, range->end);
28     for(int i= range->start; i< range->end; i++){
29         if(isPrime(i)){
30             printf("%d ", i);
31         }
32     }
33     pthread_exit(NULL);
34 }
35 int main() {
36     int maxNum, threadCount;
37
38     printf("Enter the maximum number (upper limit): ");
39     scanf("%d", &maxNum);
40
41     printf("Enter the number of threads to use: ");
42     scanf("%d", &threadCount);
43
44
45     if (maxNum < 2 || threadCount < 1) {
46         printf("Invalid input. Max number must be >= 2 and threads >= 1.\n");
47         return 1;
48     }
49     pthread_t threads[threadCount];
50     Range ranges[threadCount];
51
52     int step = maxNum / threadCount;
53     for (int i = 0; i < threadCount; i++) {
54         ranges[i].start = i * step + 1;
55         ranges[i].end = (i == threadCount - 1) ? maxNum : (i + 1) * step;
56     }
57
58     for (int i = 0; i < threadCount; i++) {
59         pthread_create(&threads[i], NULL, findPrime, (void*)&ranges[i]);
60     }
61
62     for (int i = 0; i < threadCount; i++) {
63         pthread_join(threads[i], NULL);
64     }
65     printf("\nAll threads finished.\n");
66     return 0;
67 }
```

Output

```

PS C:\Listfiles\WP\Weeks\1\ -workshop
Enter the maximum number (upper limit): 18000
Enter the number of threads to use: 4
Thread started
Thread started
Thread started
7501573 Thread started
7523 7529 7537 5001 7541 7547 7549 7559 7569 7576 7573 11 13 9009 17 7577 5011 5021 10 5023 7583 5039 5051 23 9059 7589 61 31 5077 37 5081 41 7591 43 5201 67 40 5807 7607 5099 5251 7621 2531 7639 2549 7643 2549 7649 7659 7669 7675 7687 7591 7595 7599 7609 7617 7683 5131 7623 7591 5147 7637 7643 7653 7663 7673 7683 7693 7699 7703 7709 7713 7719 7723 7729 7733 7739 7743 7749 7753 7759 7763 7769 7773 7779 7783 7789 7793 7799 7803 7809 7813 7819 7823 7829 7833 7839 7843 7849 7853 7859 7863 7869 7873 7879 7883 7889 7893 7899 7903 7909 7913 7919 7923 7929 7933 7939 7943 7949 7953 7959 7963 7969 7973 7979 7983 7989 7993 7999 8000 8001 8002 8003 8004 8005 8006 8007 8008 8009 8010 8011 8012 8013 8014 8015 8016 8017 8018 8019 8020 8021 8022 8023 8024 8025 8026 8027 8028 8029 8030 8031 8032 8033 8034 8035 8036 8037 8038 8039 8040 8041 8042 8043 8044 8045 8046 8047 8048 8049 8050 8051 8052 8053 8054 8055 8056 8057 8058 8059 8060 8061 8062 8063 8064 8065 8066 8067 8068 8069 8070 8071 8072 8073 8074 8075 8076 8077 8078 8079 8080 8081 8082 8083 8084 8085 8086 8087 8088 8089 8090 8091 8092 8093 8094 8095 8096 8097 8098 8099 8100 8101 8102 8103 8104 8105 8106 8107 8108 8109 8110 8111 8112 8113 8114 8115 8116 8117 8118 8119 8120 8121 8122 8123 8124 8125 8126 8127 8128 8129 8130 8131 8132 8133 8134 8135 8136 8137 8138 8139 8140 8141 8142 8143 8144 8145 8146 8147 8148 8149 8150 8151 8152 8153 8154 8155 8156 8157 8158 8159 8160 8161 8162 8163 8164 8165 8166 8167 8168 8169 8170 8171 8172 8173 8174 8175 8176 8177 8178 8179 8180 8181 8182 8183 8184 8185 8186 8187 8188 8189 8190 8191 8192 8193 8194 8195 8196 8197 8198 8199 8200 8201 8202 8203 8204 8205 8206 8207 8208 8209 8210 8211 8212 8213 8214 8215 8216 8217 8218 8219 8220 8221 8222 8223 8224 8225 8226 8227 8228 8229 8230 8231 8232 8233 8234 8235 8236 8237 8238 8239 8240 8241 8242 8243 8244 8245 8246 8247 8248 8249 8250 8251 8252 8253 8254 8255 8256 8257 8258 8259 8260 8261 8262 8263 8264 8265 8266 8267 8268 8269 8270 8271 8272 8273 8274 8275 8276 8277 8278 8279 8280 8281 8282 8283 8284 8285 8286 8287 8288 8289 8290 8291 8292 8293 8294 8295 8296 8297 8298 8299 8300 8301 8302 8303 8304 8305 8306 8307 8308 8309 8310 8311 8312 8313 8314 8315 8316 8317 8318 8319 8320 8321 8322 8323 8324 8325 8326 8327 8328 8329 8330 8331 8332 8333 8334 8335 8336 8337 8338 8339 8340 8341 8342 8343 8344 8345 8346 8347 8348 8349 8350 8351 8352 8353 8354 8355 8356 8357 8358 8359 8360 8361 8362 8363 8364 8365 8366 8367 8368 8369 8370 8371 8372 8373 8374 8375 8376 8377 8378 8379 8380 8381 8382 8383 8384 8385 8386 8387 8388 8389 8390 8391 8392 8393 8394 8395 8396 8397 8398 8399 8400 8401 8402 8403 8404 8405 8406 8407 8408 8409 8410 8411 8412 8413 8414 8415 8416 8417 8418 8419 8420 8421 8422 8423 8424 8425 8426 8427 8428 8429 8430 8431 8432 8433 8434 8435 8436 8437 8438 8439 8440 8441 8442 8443 8444 8445 8446 8447 8448 8449 8450 8451 8452 8453 8454 8455 8456 8457 8458 8459 8460 8461 8462 8463 8464 8465 8466 8467 8468 8469 8470 8471 8472 8473 8474 8475 8476 8477 8478 8479 8480 8481 8482 8483 8484 8485 8486 8487 8488 8489 8490 8491 8492 8493 8494 8495 8496 8497 8498 8499 8500 8501 8502 8503 8504 8505 8506 8507 8508 8509 8510 8511 8512 8513 8514 8515 8516 8517 8518 8519 8520 8521 8522 8523 8524 8525 8526 8527 8528 8529 8530 8531 8532 8533 8534 8535 8536 8537 8538 8539 8540 8541 8542 8543 8544 8545 8546 8547 8548 8549 8550 8551 8552 8553 8554 8555 8556 8557 8558 8559 8560 8561 8562 8563 8564 8565 8566 8567 8568 8569 8570 8571 8572 8573 8574 8575 8576 8577 8578 8579 8580 8581 8582 8583 8584 8585 8586 8587 8588 8589 8590 8591 8592 8593 8594 8595 8596 8597 8598 8599 8600 8601 8602 8603 8604 8605 8606 8607 8608 8609 8610 8611 8612 8613 8614 8615 8616 8617 8618 8619 8620 8621 8622 8623 8624 8625 8626 8627 8628 8629 8630 8631 8632 8633 8634 8635 8636 8637 8638 8639 8640 8641 8642 8643 8644 8645 8646 8647 8648 8649 8650 8651 8652 8653 8654 8655 8656 8657 8658 8659 8660 8661 8662 8663 8664 8665 8666 8667 8668 8669 8670 8671 8672 8673 8674 8675 8676 8677 8678 8679 8680 
```

Explanation:

This program is an interactive multithreaded prime number finder written in C using POSIX threads. It first asks the user how many threads they want to use, then splits the numbers from 1 to 10,000 into roughly equal parts so each thread gets its own section to work on. Every thread runs the `findPrimes` function, which prints the prime numbers in its assigned range. To check whether a number is prime, the program uses the `isPrime` function, which works efficiently by handling small cases, skipping even numbers, and only checking divisibility up to the square root. The threads are created with `pthread_create`, and the main program waits for them to finish using `pthread_join`. While running, the program shows which thread is responsible for each part of the range, and when all threads are done, it prints a final completion message. This setup demonstrates parallel computation and performs better than checking all numbers with a single thread.

3. Convert the program in (2) so that each thread returns the number of prime numbers that it has found using `pthread_exit()` and for main program to print out the number of prime number that each thread has found.

```
week3 > C workshop3.c > main()
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <stdlib.h>
4
5  struct Range {
6      int start;
7      int end;
8  };
9
10 int is_prime(int n) {
11     if (n <= 1) return 0;
12     for (int i = 2; i * i <= n; i++) {
13         if (n % i == 0) return 0;
14     }
15     return 1;
16 }
17
18 void* find_primes(void* arg) {
19     struct Range* r = (struct Range*)arg;
20     int* count = malloc(sizeof(int));
21     *count = 0;
22
23     for (int i = r->start; i <= r->end; i++) {
24         if (is_prime(i)) (*count)++;
25     }
26
27     pthread_exit(count);
28 }
29
30 int main() {
31     int n, limit;
32     printf("Enter number of threads: ");
33     scanf("%d", &n);
34     printf("Enter range (up to): ");
35     scanf("%d", &limit);
36
37     pthread_t threads[n];
38     struct Range ranges[n];
39
40     int range = limit / n;
41
42     for (int i = 0; i < n; i++) {
43         ranges[i].start = i * range + 1;
44         ranges[i].end = (i == n - 1) ? limit : (i + 1) * range;
45         pthread_create(&threads[i], NULL, find_primes, &ranges[i]);
46     }
47
48     int total_primes = 0;
49     for (int i = 0; i < n; i++) {
50         int* count;
51         pthread_join(threads[i], (void**)&count);
52         printf("Thread %d found %d prime numbers.\n", i + 1, *count);
53         total_primes += *count;
54         free(count);
55     }
56
57     printf("Total prime numbers: %d\n", total_primes);
58     return 0;
59 }
```

Output

```
PS C:\Clzstuffs\HPC\week3> gcc workshop3.c -o workshop3 -pthread
PS C:\Clzstuffs\HPC\week3> ./workshop3
PS C:\Clzstuffs\HPC\week3> ./workshop3
Enter number of threads: 4
Enter number of threads: 4
Enter range (up to): 1000
Enter range (up to): 1000
Thread 1 found 53 prime numbers.
Thread 1 found 53 prime numbers.
Thread 2 found 42 prime numbers.
Thread 3 found 37 prime numbers.
Thread 4 found 36 prime numbers.
Total prime numbers: 168
PS C:\Clzstuffs\HPC\week3> █
```

Explanation:

Here is an even simpler, clearer version in one smooth paragraph:

This program is an interactive multithreaded prime number finder written in C using POSIX threads. It asks the user how many threads they want to use, then divides the numbers from 1 to 10,000 into equal parts so each thread gets its own section to check. Each thread runs a function that finds and prints the prime numbers in its assigned range. To check if a number is prime, the program uses an efficient method that handles small cases, skips even numbers, and only checks divisibility up to the square root. The threads are created with `pthread_create`, and the main program waits for them to finish using `pthread_join`. As the program runs, it shows which thread is responsible for which part of the number range, and after all threads finish, it prints a completion message. Using multiple threads allows the program to check numbers in parallel, making it faster than using just one thread.

4. Convert the program in (3) to use `pthread_cancel()` to cancel all threads as soon as the 5th prime number has been found.

```
week3 > C workshop4.c > main()
1  #include <stdio.h>
2  #include <pthread.h>
3  #include <stdlib.h>
4
5  #define MAX 10000
6
7  pthread_t* threads;
8  int prime_count = 0;
9  pthread_mutex_t lock;
10
11 int is_prime(int n) {
12     if (n <= 1) return 0;
13     for (int i = 2; i * i <= n; i++) {
14         if (n % i == 0) return 0;
15     }
16     return 1;
17 }
18
19 void* find_primes(void* arg) {
20     int id = *(int*)arg;
21     int start = id * (MAX / 3) + 1;
22     int end = (id == 2) ? MAX : (id + 1) * (MAX / 3);
23
24     for (int i = start; i <= end; i++) {
25         if (is_prime(i)) {
26             pthread_mutex_lock(&lock);
27             prime_count++;
28             printf("Thread %d found prime: %d (count=%d)\n", id + 1, i, prime_count);
29             if (prime_count >= 5) {
30                 printf("\n5th prime found! Cancelling all threads...\n");
31                 for (int j = 0; j < 3; j++) {
32                     if (j != id) pthread_cancel(threads[j]);
33                 }
34                 pthread_mutex_unlock(&lock);
35                 pthread_exit(NULL);
36             }
37             pthread_mutex_unlock(&lock);
38         }
39     }
40     return NULL;
41 }
42
43 int main() {
44     pthread_mutex_init(&lock, NULL);
45     threads = malloc(3 * sizeof(pthread_t));
46     int ids[3] = {0, 1, 2};
47
48     for (int i = 0; i < 3; i++) {
49         pthread_create(&threads[i], NULL, find_primes, &ids[i]);
50     }
51
52     for (int i = 0; i < 3; i++) {
53         pthread_join(threads[i], NULL);
54     }
55
56     printf("\nAll threads done.\n");
57     pthread_mutex_destroy(&lock);
58     free(threads);
59     return 0;
60 }
```

Output

```

5th prime found! Cancelling all threads...
PS C:\Clzstuffs\HPC\week3> gcc workshop4.c -o workshop4 -pthread
PS C:\Clzstuffs\HPC\week3> ./workshop4
Thread 2 found prime: 3343 (count=1)
Thread 2 found prime: 3347 (count=2)
Thread 2 found prime: 3359 (count=3)
Thread 2 found prime: 3361 (count=4)
Thread 2 found prime: 3371 (count=5)

5th prime found! Cancelling all threads...
Thread 3 found prime: 6673 (count=6)

5th prime found! Cancelling all threads...
Thread 2 found prime: 3347 (count=2)
Thread 2 found prime: 3359 (count=3)
Thread 2 found prime: 3361 (count=4)
Thread 2 found prime: 3371 (count=5)

5th prime found! Cancelling all threads...
Thread 3 found prime: 6673 (count=6)

5th prime found! Cancelling all threads...
Thread 2 found prime: 3371 (count=5)

5th prime found! Cancelling all threads...
Thread 3 found prime: 6673 (count=6)

5th prime found! Cancelling all threads...
5th prime found! Cancelling all threads...
Thread 3 found prime: 6673 (count=6)

5th prime found! Cancelling all threads...
Thread 1 found prime: 2 (count=7)

Thread 3 found prime: 6673 (count=6)

5th prime found! Cancelling all threads...
Thread 1 found prime: 2 (count=7)

5th prime found! Cancelling all threads...
5th prime found! Cancelling all threads...
Thread 1 found prime: 2 (count=7)

5th prime found! Cancelling all threads...
Thread 1 found prime: 2 (count=7)

5th prime found! Cancelling all threads...
5th prime found! Cancelling all threads...

All threads done.
PS C:\Clzstuffs\HPC\week3> 

```

Explanation:

Here is a simpler and clearer version in one paragraph:

This program is a multithreaded prime number finder in C that uses POSIX threads, a shared counter, and a mutex to safely coordinate between threads. The user chooses how many threads to run, and the program divides the range from 1 to 10,000 into equal parts. Each thread runs a function that counts how many prime numbers appear in its section. All threads update a global counter that keeps track of the total primes found, and a mutex is used to prevent multiple threads from changing this counter at the same time. If the total number of primes reaches 5, any thread that notices this immediately stops using `pthread_exit()`, allowing the program to finish early. The `pthread_testcancel()` call inside the loop makes it possible to cancel threads safely when needed. After all threads are done, the program prints the final count and confirms that everything has finished. This program shows how threads can coordinate work, share data safely, and exit early when a condition is met.