Gilbert Condon

2/22/2023

Lab4

1)

```
gilbert@gilbert-VirtualBox:~$ ln empty.txt hardlink1.txt
gilbert@gilbert-VirtualBox:~$ ls -l
total 316
drwxr-xr-x 2 gilbert gilbert  4096 Jan 30 17:31 Desktop
drwxr-xr-x 2 gilbert gilbert  4096 Jan 11 18:19 Documents
drwxr-xr-x 2 gilbert gilbert  4096 Jan 11 18:19 Downloads
-rw-rw-r-- 2 gilbert gilbert   100 Feb 13 18:25 empty.txt
-rwxr-xr-x 1 root    root    17856 Jan 30 18:54 file
-rw-rw-r-- 1 gilbert gilbert   624 Jan 30 18:53 file.c
-rw-rw-r-- 2 gilbert gilbert   100 Feb 13 18:25 hardlink1.txt
-rw-rw-r-- 2 gilbert gilbert    25 Feb 13 17:46 hardlink.txt
-rw-rw-r-- 1 gilbert gilbert     0 Jan 25 18:22 lab2.c
-rw-rw-r-- 1 gilbert gilbert   262 Feb  1 18:05 lab2v2.c
-rwxrwxr-x 1 gilbert gilbert 17896 Feb 13 18:25 lab4
-rw-rw-r-- 1 gilbert gilbert   656 Feb 13 18:24 lab4.c
drwxr-xr-x 3 root    root     4096 Jan 25 18:30 mnt
drwxr-xr-x 2 gilbert gilbert  4096 Jan 11 18:19 Music
-rwxrwxr-x 1 gilbert gilbert 17952 Feb 13 17:55 new
-rwxrwxr-x 1 gilbert gilbert 19144 Feb 13 18:17 new2
-rw-rw-r-- 1 gilbert gilbert  1877 Feb 13 18:16 new2.c
-rw-rw-r-- 1 gilbert gilbert   280 Feb 13 17:54 new.c
drwxr-xr-x 2 gilbert gilbert  4096 Jan 11 18:19 Pictures
```

```
gilbert@gilbert-VirtualBox:~$ ln -s testfile1.txt softlink2.txt
```

```
lrwxrwxrwx 1 gilbert gilbert    13 Feb 22 16:21 softlink2.txt -> testfile1.txt
```

2) This program uses 3 threads to calculate the three computations at once. The manner of calculating the average, min and max is nothing extradordinary the interesting portion is the way we use pthread_create() to call the three functions at once.

```
gilbert@gilbert-VirtualBox:~$ ./lab4v1
The average value is 82.86
The minimum value is 72
The maximum value is 95
```

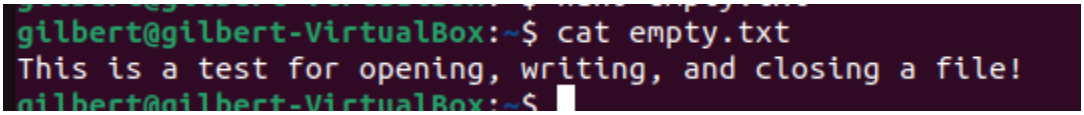
```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#define NUM_THREADS 3

int numbers[] = {90, 81, 78, 95, 79, 72, 85};
int num_count = sizeof(numbers) / sizeof(int);
double average;
int max, min;
void *calc_average(void *arg)
{
```

```
   double sum = 0.0;
   for (int i = 0; i < num_count; i++)
   {
      sum += numbers[i];
   }
   average = sum / num_count;
   pthread_exit(NULL);
}
void *calc_max(void *arg)
{
   max = numbers[0];
   for (int i = 1; i < num_count; i++)
   {
      if (numbers[i] > max)
      {
         max = numbers[i];
      }
   }
   pthread_exit(NULL);
}
void *calc_min(void *arg)
{
   min = numbers[0];
   for (int i = 1; i < num_count; i++)
   {
      if (numbers[i] < min)
      {
         min = numbers[i];
      }
   }
   pthread_exit(NULL);
}
int main(int argc, char *argv[])
{
   pthread_t threads[NUM_THREADS];
   int rc;
   rc = pthread_create(&threads[0], NULL, calc_average, NULL);
   if (rc)
   {
      printf("Error: Unable to create thread.\n");
      exit(-1);
   }
   rc = pthread_create(&threads[1], NULL, calc_max, NULL);
   if (rc)
   {
      printf("Error: Unable to create thread.\n");
      exit(-1);
   }
   rc = pthread_create(&threads[2], NULL, calc_min, NULL);
   if (rc)
   {
      printf("Error: Unable to create thread.\n");
      exit(-1);
   }
   for (int i = 0; i < NUM_THREADS; i++)
   {
      rc = pthread_join(threads[i], NULL);
      if (rc)
      {
         printf("Error: Unable to join thread.\n");
         exit(-1);
      }
   }
   printf("The average value is %.2f\n", average);
   printf("The minimum value is %d\n", min);
   printf("The maximum value is %d\n", max);
```

```
  pthread_exit(NULL);
}
```

3) This program opens, writes text to a text file and then closes that file. We use open() to open a file we save the text we are going to write to the file to a variable called buff and then we use write() to write the text to a file and use to exit() to exit the file.




```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

int main()

{

   int fd;

   char buf[100] = "This is a test for opening, writing, and closing a file!";

   ssize_t n;

   fd = open("empty.txt", O_WRONLY | O_CREAT, 0644);

   if (fd == -1)

   {

      perror("open");

      exit(EXIT_FAILURE);

   }

   n = write(fd, buf, sizeof(buf));

   if (n == -1)

   {

      perror("write");

      exit(EXIT_FAILURE);

   }

   if (close(fd) == -1)

   {

      perror("close");

      exit(EXIT_FAILURE);

   }
```

```
    return 0;
}
```

4) For addition each thread computes its own portion of the matrix addition process, for subtraction likewise each thread computes its own process and for multiplaction once again we create separate threads for each element in the resultant matrix. We use pthread_exit() to return the computed value of each thread which we collect by pthread_join().

```
gilbert@gilbert-VirtualBox:~/CS480/lab4$ ./lab4v3

Matrix A:
3 7 3 6
9 2 0 3
0 2 1 7
2 2 7 9

Matrix B:
6 5 5 2
1 7 9 6
6 6 8 9
0 3 5 2

 Sum of Matrix A and B:
9   12   8   8
10   9   9   9
6   8   9   16
2   5   12   11

 Subtraction of Matrix A and B:
-3   2   -2   4
8   -5   -9   -3
-6   -4   -7   -2
2   -1   2   7

Multiplcation of A and B is :
43 100 132 87
56 68 78 36
8 41 61 35
56 93 129 97
gilbert@gilbert-VirtualBox:~/CS480/lab4$
```

```
#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

#include <stdlib.h>

#define MAX 4

#define CORE 4

#define MAX 4
```

```c
pthread_t thread[CORE * 2];

int mat_A[MAX][MAX], mat_B[MAX][MAX], sum[MAX][MAX], sub[MAX][MAX];

void *subtraction(void *arg)

{

    int i, j;

    int core = (int)arg;

    for (i = core * MAX / 4; i < (core + 1) * MAX / 4; i++)

    {

        for (j = 0; j < MAX; j++)

        {

            // computes subtraction row wise

            sub[i][j] = mat_A[i][j] - mat_B[i][j];
        }
    }
}

void *addition(void *arg)

{

    int i, j;

    int core = (int)arg;

    for (i = core * MAX / 4; i < (core + 1) * MAX / 4; i++)

    {

        for (j = 0; j < MAX; j++)

        {

            // computation of sum row wise

            sum[i][j] = mat_A[i][j] + mat_B[i][j];
        }
    }
}

void *mult(void *arg)

{
```

```c
    int *data = (int *)arg;

    int k = 0, i = 0;

    int x = data[0];

    for (i = 1; i <= x; i++)

        k += data[i] * data[i + x];

    int *p = (int *)malloc(sizeof(int));

    *p = k;

    // terminates thread and returns value

    pthread_exit(p);
}

int main()

{

    int i, j, k, row1 = MAX, row2 = MAX, column1 = MAX, column2 = MAX, step = 0;

    for (i = 0; i < MAX; i++)

    {

        for (j = 0; j < MAX; j++)

        {

            mat_A[i][j] = rand() % 10;

            mat_B[i][j] = rand() % 10;
        }
    }

    printf("\nMatrix A:\n");

    for (i = 0; i < MAX; i++)

    {

        for (j = 0; j < MAX; j++)

        {

            printf("%d ", mat_A[i][j]);
        }

        printf("\n");
```

```
    }

    printf("\nMatrix B:\n");

    for (i = 0; i < MAX; i++)

    {

        for (j = 0; j < MAX; j++)

        {

            printf("%d ", mat_B[i][j]);
        }

        printf("\n");
    }

    for (i = 0; i < CORE; i++)

    {

        pthread_create(&thread[i], NULL, &addition, (void *)step);

        pthread_create(&thread[i + CORE], NULL, &subtraction, (void *)step);

        step++;
    }

    for (i = 0; i < CORE * 2; i++)

    {

        pthread_join(thread[i], NULL);
    }

    printf("\n Sum of Matrix A and B:\n");

    for (i = 0; i < MAX; i++)

    {

        for (j = 0; j < MAX; j++)

        {

            printf("%d   ", sum[i][j]);
        }

        printf("\n");
    }

    printf("\n Subtraction of Matrix A and B:\n");
```

```
    for (i = 0; i < MAX; i++)

    {

        for (j = 0; j < MAX; j++)

        {

            printf("%d   ", sub[i][j]);
        }

        printf("\n");
    }

    printf("\n");

    int max = row1 * column2;

    pthread_t *threads;

    threads = (pthread_t *)malloc(max * sizeof(pthread_t));

    int count = 0;

    int *data = NULL;

    for (i = 0; i < row1; i++)

        for (j = 0; j < column2; j++)

        {

            data = (int *)malloc((20) * sizeof(int));

            data[0] = column1;

            for (k = 0; k < column1; k++)

                data[k + 1] = mat_A[i][k];

            for (k = 0; k < row2; k++)

                data[k + column1 + 1] = mat_B[k][j];

            pthread_create(&threads[count++], NULL,

                           mult, (void *)(data));
        }

    printf("Multiplcation of A and B is : \n");

    for (i = 0; i < max; i++)

    {
```

```
        void *k;

        pthread_join(threads[i], &k);

        int *p = (int *)k;

        printf("%d ", *p);

        if ((i + 1) % column2 == 0)

            printf("\n");
    }

    return 0;
}
```