**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP MÔN HỌC NMHĐH**

**LAB 5**

**Nhóm:** **22** **Tổ: 01**

**Tên : Nguyễn Thị Anh Thư**

**MSSV: 51900564**

**NỘI DUNG BÁO CÁO KẾT QUẢ**

**Bài 2 :** Bài toán ước lượng giá trị số PI

Một cách giá trị π khá thú vị là sử dụng kỹ thuật Monte Carlo, liên quan đến ngẫu nhiên. Kỹ thuật

này hoạt động như sau: Giả sử bạn có một vòng tròn bán kính là 1 nội tiếp trong một hình vuông

cạnh là 2, như thể hiện trong hình sau:

- Đầu tiên, tạo một chuỗi các điểm ngẫu nhiên dưới dạng tọa độ (x, y) đơn giản. Những điểm

này phải nằm trong tọa độ Descartes bị ràng buộc hình vuông. Trong tổng số điểm ngẫu

nhiên được tạo, một số sẽ nằm trong vòng tròn.

- Tiếp theo, ước tính π bằng cách thực hiện phép tính sau: π = 4 × (số điểm trong vòng tròn)

/ (tổng số điểm)

Chương trình cần tạo ra n tiểu trình và mỗi tiểu trình sẽ sinh ra m điểm, cũng chính tiểu trình sẽ

tính khoảng cách d và cập nhật vào biến counter (là tổng số điểm nằm trong hình tròn, biến toàn

cục chia sẻ). Xem lưu đồ kèm theo.

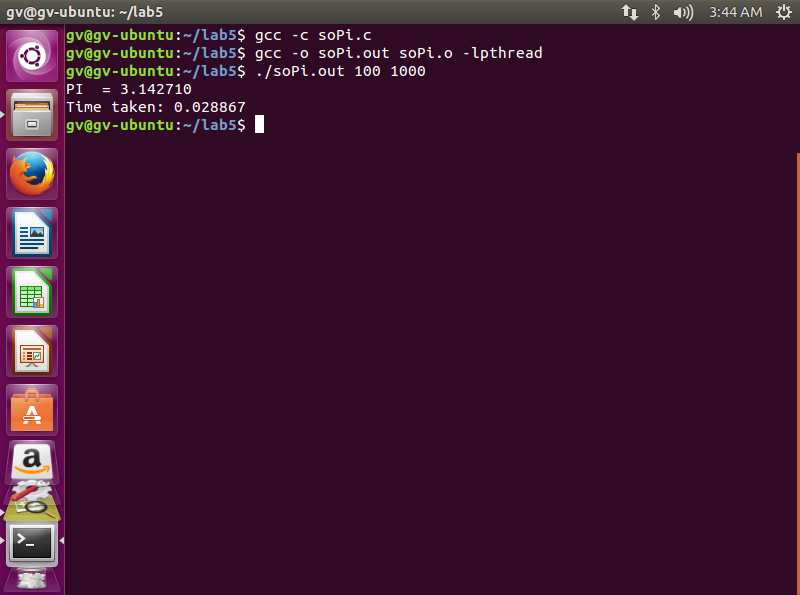
Mỗi tiểu trình cũng cần ghi giá trị các điểm sinh ra và một tập tin m\_point.txt.

**KẾT QUẢ THỰC HIỆN**

**2.1. Phần source code:**

1. #include <stdio.h>
2. #include <pthread.h>
3. #include <time.h>
4. #include <sys/time.h>
5. #include <sys/syscall.h>
6. #include <sys/types.h>
7. #include <sys/stat.h>
8. #include <fcntl.h>
9. #include <string.h>
10. #include <stdlib.h>
11. #define MAX 16
12. #define MAX\_THREAD 40
13. #define MAX\_POINT 10000
14. int counter = 0;
15. int in = 0;
16. void\* count(void\* arg)
17. {
18. int temp =0;
19. float x;
20. float y;
21. for(int i=0; i<MAX\_POINT;i++){
22. x = ((float)rand()/(float)(RAND\_MAX)) \* 2 -1;
23. y = ((float)rand()/(float)(RAND\_MAX)) \* 2 -1;
24. if(x\*x + y\*y <= 1){
25. temp++;
26. }
27. }
28. in+= temp;
29. counter += MAX\_POINT;
30. }
31. int main() {
32. clock\_t t1, t2;
33. t1 = clock();
34. pthread\_t threads[MAX\_THREAD];
35. *// Creating 4 threads*
36. for (int i = 0; i < MAX\_THREAD ; i++)
37. pthread\_create(&threads[i],NULL,&count,(void\*) &counter);
38. *// joining 4 threads i.e. waiting for all 4 threads to complete*
39. for (int i = 0; i < MAX\_THREAD ; i++)
40. pthread\_join(threads[i],NULL);
41. printf("PI = %f\n",(double)in/(double)counter \* 4);
42. t2 = clock();
43. *// time taken by merge sort in seconds*
44. printf("Time taken: %f\n",(t2 - t1) / (double)CLOCKS\_PER\_SEC);
45. return 0;
46. }

**2.2. Kết quả chạy chương trình:**

******

**Bài 3:**

Merge Sort is a popular sorting technique which divides an array or list into two halves

and then start merging them when sufficient depth is reached. Time complexity of

merge sort is O(nlogn).

Threads are lightweight processes and threads shares with other threads their code

section, data section and OS resources like open files and signals. But, like process, a

thread has its own program counter (PC), a register set, and a stack space.

Multi-threading is way to improve parallelism by running the threads simultaneously in

different cores of your processor. In this program, we’ll use 4 threads but you may

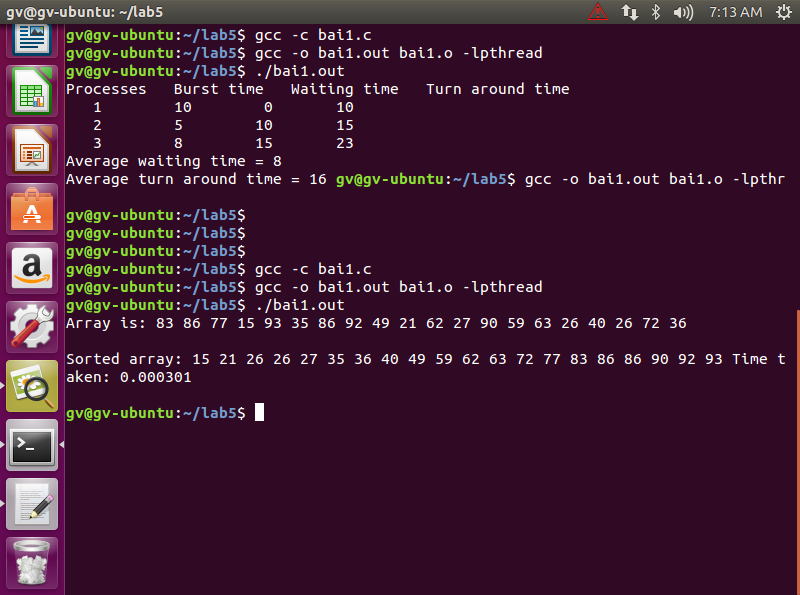
change it according to the number of cores your processor has.

**KẾT QUẢ THỰC HIỆN**

**3.1 Phần source code:**

1. *// CPP Program to implement merge sort using*
2. *// multi-threading*
3. *//#include <iostream>*
4. #include <pthread.h>
5. #include <time.h>
6. #include <stdlib.h>
7. #include<stdio.h>
8. *// number of elements in array*
9. #define MAX 20
10. *// number of threads*
11. #define THREAD\_MAX 4
12. *//using namespace std;*
13. *// array of size MAX*
14. int a[MAX];
15. int part = 0;
16. *// merge function for merging two parts*
17. void merge(int low, int mid, int high)
18. {
19. int\* left = (int\*) malloc( (mid - low + 1) \* sizeof(int));
20. int\* right = (int\*) malloc( (high - mid) \* sizeof(int));
21. *// n1 is size of left part and n2 is size*
22. *// of right part*
23. int n1 = mid - low + 1,
24. n2 = high - mid,
25. i, j;
26. *// storing values in left part*
27. for (i = 0; i < n1; i++)
28. left[i] = a[i + low];
29. *// storing values in right part*
30. for (i = 0; i < n2; i++)
31. right[i] = a[i + mid + 1];
32. int k = low;
33. i = j = 0;
34. *// merge left and right in ascending order*
35. while (i < n1 && j < n2) {
36. if (left[i] <= right[j])
37. a[k++] = left[i++];
38. else
39. a[k++] = right[j++];
40. }
41. *// insert remaining values from left*
42. while (i < n1) {
43. a[k++] = left[i++];
44. }
45. *// insert remaining values from right*
46. while (j < n2) {
47. a[k++] = right[j++];
48. }
49. free(left);
50. free(right);
51. }
52. *// merge sort function*
53. void merge\_sort(int low, int high)
54. {
55. *// calculating mid point of array*
56. int mid = low + (high - low) / 2;
57. if (low < high) {
58. *// calling first half*
59. merge\_sort(low, mid);
60. *// calling second half*
61. merge\_sort(mid + 1, high);
62. *// merging the two halves*
63. merge(low, mid, high);
64. }
65. }
66. *// thread function for multi-threading*
67. void\* merge\_sort123(void\* arg){
68. *// which part out of 4 parts*
69. int thread\_part = part++;
70. *// calculating low and high*
71. int low = thread\_part \* (MAX / THREAD\_MAX);
72. int high = (thread\_part + 1) \* (MAX / THREAD\_MAX) - 1;
73. *// evaluating mid point*
74. int mid = low + (high - low) / 2;
75. if (low < high) {
76. merge\_sort(low, mid);
77. merge\_sort(mid + 1, high);
78. merge(low, mid, high);
79. }
80. return 0;
81. }
82. *// Driver Code*
83. int main()
84. {
85. *// generating random values in array*
86. printf("Array is: ");
87. for (int i = 0; i < MAX; i++){
88. a[i] = rand() % 100;
89. printf("%d ", a[i]);
90. }clock\_t t1, t2;
91. t1 = clock();
92. pthread\_t threads[THREAD\_MAX];
93. *// creating 4 threads*
94. for (int i = 0; i < THREAD\_MAX; i++)
95. pthread\_create(&threads[i], NULL, merge\_sort123,
96. (void\*)NULL);
97. *// joining all 4 threads*
98. for (int i = 0; i < THREAD\_MAX; i++)
99. pthread\_join(threads[i], NULL);
100. *// merging the final 4 parts*
101. merge(0, (MAX / 2 - 1) / 2, MAX / 2 - 1);
102. merge(MAX / 2, MAX/2 + (MAX-1-MAX/2)/2, MAX - 1);
103. merge(0, (MAX - 1)/2, MAX - 1);
104. t2 = clock();
105. *// displaying sorted array*
106. printf("\n\nSorted array: ");
107. for (int i = 0; i < MAX; i++)
108. printf ("%d ", a[i]);
109. printf("Time taken: %f\n",(t2 - t1) / (double)CLOCKS\_PER\_SEC);
110. printf("\n");
111. return 0;
112. }

**2.2 Kết quả chạy chương trình:**

****

**Bài 1**:

**Yêu cầu 1.1**. Tính thời gian quay vòng trung bình, thời gian chờ trung bình cho thuật toán FCFS bằng cách chỉnh sửa bổ sung hàm schedule() trong tập tin schedule\_FCFS.c

**Yêu cầu 1.2.** Giả sử thời gian đến của mỗi tiến trình khác nhau và khác 0. Hãy tiếp tục chỉnh sửa hàm schedule() trong tập tin schedule\_FCFS.c. Gợi ý: Trong tập tin schedule\_FCFS.c một mảng các Task đã được dùng để chứa thông tin tiến trình theo thứ tự "First come". Hãy sắp xếp lại mảng đó với tiêu chí "Arrival Time First", rồi tính toán và xuất ra theo thứ tự mảng.

**Yêu cầu 1.3**: hiện thực tập tin schedule\_priority.c. Gợi ý: tương tự yêu cầu 2, hãy sắp mảng các Task lại theo tiêu chí "Độ ưu tiên", nếu cùng độ ưu tiên thì dùng tiêu chí phụ là FCFS.

**Yêu cầu 1.4**: hiện thực tập tin schedule\_sjf.c. Gợi ý: tương tự yêu cầu 2, hãy sắp mảng các Task lại theo tiêu chí "Burst", nếu cùng độ ưu tiên thì dùng tiêu chí phụ là FCFS.

**Yêu cầu 1.5**: hiện thực tập tin schedule\_rr.c. Gợi ý: Mảng các Task cần cập nhật lại Burst của mỗi tác vụ sau mỗi lần đáp ứng CPU, tác vụ nào có Burst giảm còn 0 thì xóa khỏi mảng Task.

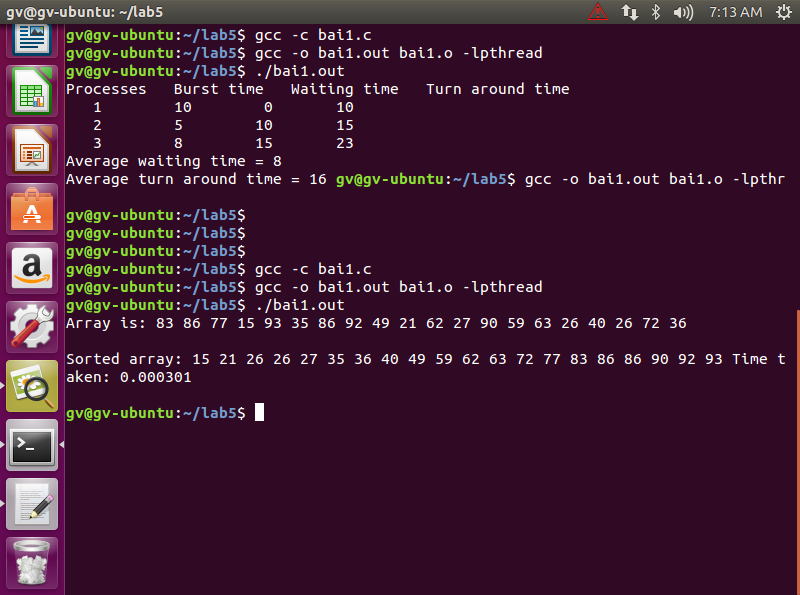
**Yêu cầu 1.6**: hiện thực tập tin schedule\_priority\_rr.c. Gợi ý: Copy lại Yêu cầu 3 và bổ sung: nếu có nhiều tác vụ cùng độ ưu tiên thì sao chép chúng và một Mảng khác và chạy RR trên mảng đó.

**KẾT QUẢ THỰC HIỆN**

* 1. **Phần source code:**

1. *// C program for implementation of FCFS*
2. *// scheduling*
3. #include<stdio.h>
4. #include <stdlib.h>
5. *// Function to find the waiting time for all*
6. *// processes*
7. void findwaitingTime(int processes[], int n,
8. int bt[], int wt[])
9. {
10. *// waiting time for first process is 0*
11. wt[0] = 0;
12. *// calculating waiting time*
13. for (int i = 1; i < n ; i++ )
14. wt[i] = bt[i-1] + wt[i-1] ;
15. }
16. *// Function to calculate turn around time*
17. void findTurnAroundTime( int processes[], int n, int bt[], int wt[], int tat[])
18. {
19. *// calculating turnaround time by adding*
20. *// bt[i] + wt[i]*
21. for (int i = 0; i < n ; i++)
22. tat[i] = bt[i] + wt[i];
23. }
24. *//Function to calculate average time*
25. void findavgTime( int processes[], int n, int bt[]) {
26. int wt[n], tat[n], total\_wt = 0, total\_tat = 0;
27. *//Function to find waiting time of all processes*
28. findwaitingTime(processes, n, bt, wt);
29. *//Function to find turn around time for all processes*
30. findTurnAroundTime(processes, n, bt, wt, tat);
31. *//Display processes along with all details*
32. printf("Processes Burst time Waiting time Turn around time\n");
33. *// Calculate total waiting time and total turn*
34. *// around time*
35. for (int i=0; i<n; i++) {
36. total\_wt = total\_wt + wt[i];
37. total\_tat = total\_tat + tat[i];
38. printf(" %d ",(i+1));
39. printf(" %d ", bt[i] );
40. printf(" %d",wt[i] );
41. printf(" %d\n",tat[i] );
42. }
43. int s=(float)total\_wt / (float)n;
44. int t=(float)total\_tat / (float)n;
45. printf("Average waiting time = %d",s);
46. printf("\n");
47. printf("Average turn around time = %d ",t);
48. }
49. *// Driver code*
50. int main() {
51. *//process id's*
52. int processes[] = { 1, 2, 3};
53. int n = sizeof processes / sizeof processes[0];
54. *//Burst time of all processes*
55. int burst\_time[] = {10, 5, 8};
56. findavgTime(processes, n, burst\_time);
57. return 0;
58. }

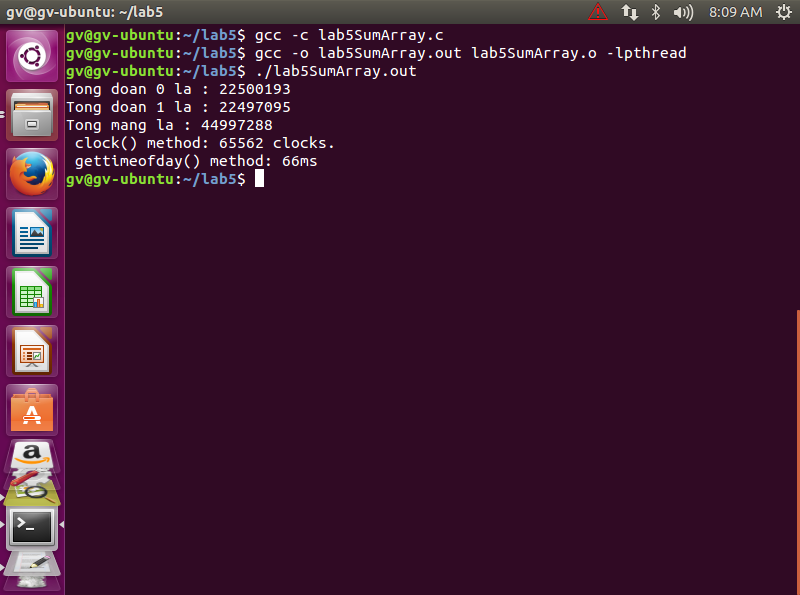
**3.2 Kết quả chạy chương trình:**

****

**Bài : Tổng mảng**

**1.Phần source code:**

1. *// CPP Program to find sum of array*
2. /\* printf \*/
3. #include <stdio.h>
4. /\* pthread\_\* \*/
5. #include <pthread.h>
6. /\* clock \*/
7. #include <time.h>
8. /\* gettimeofday \*/
9. #include <sys/time.h>
10. /\* syscall \*/
11. *//#include <sys/syscall.h>*
12. /\* pid\_t \*/
13. #include <sys/types.h>
14. /\* open \*/
15. #include <sys/stat.h>
16. #include <fcntl.h>
17. /\* strrchr \*/
18. #include <string.h>
19. #include<stdlib.h>
20. #define MAX 10000000
21. *// maximum number of threads*
22. #define MAX\_THREAD 2
23. int a[MAX];
24. int sum[4] = { 0 };
25. int part = 0;
26. void\* sum\_array(void\* arg) {
27. int thread\_part = part++;
28. int local\_sum=0;
29. for (int i = thread\_part \* (MAX / MAX\_THREAD); i < (thread\_part + 1) \* (MAX / MAX\_THREAD); i++)
30. local\_sum += a[i];
31. sum[thread\_part] = local\_sum;
32. }
33. int main() {
34. srand((unsigned int)time(NULL));
35. for(int i=0; i<MAX; i++){
36. a[i] = rand()%10;
37. }
38. clock\_t t1, t2;
39. struct timeval start2, end2;
40. t1 = clock();
41. gettimeofday(&start2, NULL);
42. pthread\_t threads[MAX\_THREAD];
43. *// Creating MAX\_THREAD threads*
44. for (int i = 0; i < MAX\_THREAD ; i++)
45. pthread\_create(&threads[i], NULL, sum\_array, (void\*)NULL);
46. *// joining MAX\_THREAD threads*
47. for (int i = 0; i < MAX\_THREAD ; i++)
48. pthread\_join(threads[i],NULL);
49. int total\_sum = 0;
50. for (int i = 0; i < MAX\_THREAD; i++)
51. { printf("Tong doan %d la : %d\n", i, sum[i]);
52. total\_sum += sum[i]; }
53. printf("Tong mang la : %d\n", total\_sum);
54. t2 = clock();
55. gettimeofday(&end2, NULL);
56. printf(" clock() method: %ld clocks.\n", (t2 - t1) );
57. printf(" gettimeofday() method: %ldms\n", (end2.tv\_sec - start2.tv\_sec) \*1000 + (end2.tv\_usec - start2.tv\_usec)/1000);
58. return 0;
59. }
60. **Kết quả chạy chương trình:**



**KẾT LUẬN**

Qua bài thực hành lab5 , em học được cách đo thời gian, lập lịch CPU, sắp xếp mảng merge sort bằng Thread, ước lượng số Pi. Tính thời giant rung bình cho thuật toán FCFS, sắp xếp các p theo FCFS. Priority,arrival time first, round robin.