Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Операционные системы»

Студент: А. О. Дубинин
Преподаватель: Е. С. Миронов
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2018

Лабораторная работа №3

Цель работы: Приобретение практических навыков в управлении потоками в ОС, обеспечении синхронизации между потоками.

Задание: Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков. Ограничение потоков может быть задано или ключом запуска вашей программы, или алгоритмом. Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы.

Вариант 3: Отсортировать массив строк при помощи параллельной сортировки слиянием.

1 Описание

Потоки используются при разбивании массива на два. Количество потоков ограничено параметром алгоритма(max_number_of_threads), который не разбивает массив на два, когда колличество элементов меньше max_number_of_threads, а запускает qsort. Либо потоки ограничены системой (указано в /proc/sys/kernel/threads-max).

Согласно [1], потоки так же можно использовать при сливании двух массивов.

Из библиотеки pthreads использовались следующие функции:
pthreadcreate – создание потока.
pthreadjoin – ожидание завершения потока, переданного в аргументах.
pthread_mutex_init – Инициализация мьютекса, при успешном завершении функция возвращает 0, в противном случае - код ошибки.
pthread_mutex_lock – Захват мьютекса, при успешном завершении функция возвращает 0, в противном случае - код ошибки.
pthread_mutex_unlock – Освобождение мьютекса. При успешном завершении функция возвращает 0, в противном случае - код ошибки.

2 Исходный код

```
1  //
2  // main.c
3  // 10.11.18
4  //
5
6  #include <pthread.h>
7  #include <stdio.h>
8  #include <stdlib.h>
9  #include <string.h>
10
11 #define string char *
12 #define MIN_LENGTH 4
13
14 int number_of_threads;
15 int max_number_of_threads;
16 pthread_mutex_t lock_number_of_threads;
17
18 typedef struct {
19     string* array;
20     int left;
21     int right;
22 } thread_data_t;
23
24
25 int cstring_cmp( const void* a, const void* b );
26
27 int string_count( FILE* f );
28
29 void read_to_array( FILE* f, string* arr, int size );
30
31 void print_array( FILE* f, string* arr, int size );
32
33 void merge( string* data, int left, int right );
34
35 void* merge_sort_threaded( void* arg );
36
37 void merge_sort( string* array, int left, int right );
38
39 int main( int argc, char** argv ) {
40
41     if ( argc < 2 ) {
42         fprintf( stdout, "Error: enter input file name and count of threads.\n" );
43         return 0;
44     }
45     FILE* input = fopen( argv[1], "r" );
46     FILE* output = stdout;
47     max_number_of_threads = 25;
```

```
48      if( argc == 3 )
49          max_number_of_threads = atoi(argv[2]);
50
51      int size = string_count( input );
52      //initialize_string_array(arr, 65, size);
53      string* arr = ( string* ) malloc( sizeof( string* ) * size );
54      for ( int i = 0; i < size; ++i ) {
55          arr[i] = ( string ) malloc( sizeof( char ) * 65 );
56          arr[i][64] = '\0';
57      }
58
59      read_to_array( input, arr, size );
60
61      merge_sort( arr, 0, size - 1 );
62
63      print_array( output, arr, size );
64
65      fclose( input );
66      fclose( output );
67
68      //getchar();
69      //getchar();
70
71      return 0;
72  }
73
74  int cstring_cmp( const void* a, const void* b ) {
75      const char** ia = ( const char** ) a;
76      const char** ib = ( const char** ) b;
77      return strcmp( *ia, *ib );
78      /* strcmp functions works exactly as expected from
79      comparison function */
80  }
81
82  int string_count( FILE* f ) {
83      int i = 0;
84      string s = ( string ) malloc( sizeof( char ) * 65 );
85      while ( !feof( f )) {
86          fscanf( f, "%s", s );
87          i++;
88      }
89      rewind( f );
90      return i;
91  }
92
93  void read_to_array( FILE* f, string* arr, int size ) {
94      for ( int i = 0; i < size; ++i ) {
95          fscanf( f, "%s", arr[i] );
96      }
```

```
97  }
98
99  void print_array( FILE* f, string* arr, int size ) {
100     for ( int i = 0; i < size; ++i ) {
101         fprintf( f, "%s\n", arr[i] );
102     }
103 }
104
105 void merge( string* data, int left, int right ) {
106     int ctr = 0;
107     int i = left;
108     int mid = left + (( right - left ) / 2 );
109     int j = mid + 1;
110     string* c = ( string* ) malloc(( right - left + 1 ) * sizeof( string ));
111     for ( int k = 0; k < right - left + 1; k++ ) {
112         c[k] = ( string ) malloc( sizeof( char ) * 65 );
113     }
114     while ( i <= mid && j <= right ) {
115         if ( strcmp( data[i], data[j] ) <= 0 ) {
116             strcpy( c[ctr++], data[i++] );
117         } else {
118             strcpy( c[ctr++], data[j++] );
119         }
120     }
121
122     if ( i == mid + 1 ) {
123         while ( j <= right ) {
124             strcpy( c[ctr++], data[j++] );
125         }
126     } else {
127         while ( i <= mid ) {
128             strcpy( c[ctr++], data[i++] );
129         }
130     }
131
132     i = left;
133     ctr = 0;
134     while ( i <= right ) {
135         strcpy( data[i++], c[ctr] );
136         free( c[ctr++] );
137     }
138     free( c );
139     return;
140 }
141
142 void* merge_sort_threaded( void* arg ) {
143     //getchar();
144     thread_data_t *data = (thread_data_t *) arg;
145     int l = data->left;
```

```
146     int r = data->right;
147
148
149
150     if (r - l + 1 <= MIN_LENGTH) {
151         qsort(data->array + l, r - l + 1, sizeof(string), cstring_cmp);
152     } else {
153         // Try to create two threads
154         int m = l + ((r - l) / 2);
155         thread_data_t data_0;
156         data_0.left = l;
157         data_0.right = m;
158         data_0.array = data->array;
159
160
161
162         int rc;
163         int created_thread_0;
164         int created_thread_1;
165         pthread_t thread0;
166         if( number_of_threads < max_number_of_threads ) {
167             pthread_mutex_lock(&lock_number_of_threads);
168             ++number_of_threads;
169             pthread_mutex_unlock(&lock_number_of_threads);
170             rc = pthread_create(&thread0, NULL, merge_sort_threaded, &data_0);
171             created_thread_0 = 1;
172         }
173         else if (rc || number_of_threads >= max_number_of_threads) {
174             //Failed to creade thread
175             created_thread_0 = 0;
176             qsort(data->array + l, m - l + 1, sizeof(string), cstring_cmp);
177         }
178
179         thread_data_t data_1;
180         data_1.left = m + 1;
181         data_1.right = r;
182         data_1.array =data->array;
183
184         pthread_t thread1;
185         if( number_of_threads < max_number_of_threads ) {
186             pthread_mutex_lock(&lock_number_of_threads);
187             ++number_of_threads;
188             pthread_mutex_unlock(&lock_number_of_threads);
189             rc = pthread_create(&thread1, NULL, merge_sort_threaded, &data_1);
190             created_thread_1 = 1;
191         }
192         else if (rc || number_of_threads >= max_number_of_threads) {
193             //Failed to create thread
194             created_thread_1 = 0;
```

```
195             qsort(data->array +m + 1, r - m, sizeof(string), cstring_cmp);
196         }
197 
198         if (created_thread_0) {
199             pthread_join(thread0, NULL);
200         }
201         if (created_thread_1) {
202             pthread_join(thread1, NULL);
203         }
204         merge(data->array, l, r);
205     }
206     pthread_exit(NULL);
207 }
208 
209 void merge_sort( string* array, int left, int right ) {
210     thread_data_t data;
211     data.array = array;
212     data.left = left;
213     data.right = right;
214 
215     number_of_threads = 0;
216     pthread_mutex_init(&lock_number_of_threads, NULL);
217 
218     pthread_mutex_lock(&lock_number_of_threads);
219     ++number_of_threads;
220     pthread_mutex_unlock(&lock_number_of_threads);
221     pthread_t thread;
222     int rc = pthread_create( &thread, NULL, merge_sort_threaded, &data );
223 
224     if ( rc ) {
225         //Failed
226         printf( "Sort failed, start qsort\n" );
227         qsort( array + left, right - left + 1, sizeof( string ), cstring_cmp );
228     }
229     pthread_join( thread, NULL);
230     return;
231 }
```

3 Тесты

```
art@mars:~/study/semester_3/OS/lab_3$ cat text.txt
Lorem
Aliquam
Vestibulum
Nunc
Cras
Vivamus
Praesent
Fusce
Integer
Vestibulum
Ut
Cras
Donec
Lorem
Aliquam
Vestibulum
Nunc
Cras
Vivamus
Praesent
art@mars:~/study/semester_3/OS/lab_3$ gcc -pthread merge_sort.c -o merge
art@mars:~/study/semester_3/OS/lab_3$ ./merge text.txt
Aliquam
Aliquam
Cras
Cras
Cras
Donec
Fusce
Integer
Lorem
Lorem
Nunc
Nunc
Praesent
Praesent
Ut
Vestibulum
```

Vestibulum
Vestibulum
Vivamus
Vivamus

4 Диагностика

```
art@mars:~/workdir/OS/lab_3/cmake-build-debug$ strace -fc ./lab_3 ../text.txt
5strace: Process 15328 attached
strace: Process 15329 attached
strace: Process 15330 attached
strace: Process 15331 attached
strace: Process 15332 attached

Вдруг
Ворона
Вороне
Да
Лиса
Лису
На
На
Позавтракать-было
а
беду
бежала;
близехонько
бог
взгромоздясь,
во
где-то
держала.
дух
ель
кусочек
остановил:
позадумалась,
послал
рту
собралась,
совсем
сыр
сырный
сыру;
ту
уж
```

```
% time     seconds  usecs/call     calls    errors syscall
------------------------------------------------------------------
0.00    0.000000           0         7           read
0.00    0.000000           0        33           write
0.00    0.000000           0         7           close
0.00    0.000000           0         7           fstat
0.00    0.000000           0         1           lseek
0.00    0.000000           0        22           mmap
0.00    0.000000           0        18           mprotect
0.00    0.000000           0         7           munmap
0.00    0.000000           0         3           brk
0.00    0.000000           0         2           rt_sigaction
0.00    0.000000           0         1           rt_sigprocmask
0.00    0.000000           0         5         5 access
0.00    0.000000           0         5           madvise
0.00    0.000000           0         5           clone
0.00    0.000000           0         1           execve
0.00    0.000000           0         1           arch_prctl
0.00    0.000000           0         8           futex
0.00    0.000000           0         1           set_tid_address
0.00    0.000000           0         6           openat
0.00    0.000000           0         6           set_robust_list
0.00    0.000000           0         1           prlimit64
------------------------------------------------------------------
100.00    0.000000                   147         5 total
art@mars:~/workdir/OS/lab_3/cmake-build-debug$
```

5 Выводы

Выполнив лабораторную работу, я приобрёл практические навыки в управлении потоками в ОС, обеспечении синхронизации между потоками. Мною были изучены и применены стандарты работы с потоками в POSIX. Понял, что отладка многопоточных приложений сложна и требует дополнительных знаний. На практике я увидел, что потоки имеют общую память и с этим могут возникнуть трудности, которые решаются с помощью средств синхронизации.

Список литературы

[1] *ИТМО - многопоточная сортировка слиянием*
URL: https://neerc.ifmo.ru/wiki/index.php?title=Многопоточная_сортировка_слиянием