Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Операционные системы»
Управление потоками в ОС

Студент: А. П. Шорохов
Преподаватель: А. А. Соколов
Группа: М8О-206Б
Дата:
Оценка:
Подпись:

Москва, 2020

Лабораторная работа №3

Задача: 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 задания: Отсортировать массив строк при помощи четно-нечетной сортировки Бетчера.

1 Описание

Принцип работы алгоритма практически ничем не отличается от обычной сортировки слиянием, однако операции слияния выполняются совершено по-разному.
Если в merge sort мы заводим два индекса — в первой и во второй половине массива, где половины уже отсортированы, и на каждом шаге просто ставим в результат наименьший из текущих в каждой половине, то здесь мы просто делаем операцию compareExchange на массивах, а потом слияние получившегося массива.

Мы применяем нерекурсивную сортировку Бэтчера к входным данным, распределяя их при этом на множество потоков.

2 Исходный код

$int\ compare(std::string\&\ a,\ std::string\&\ b)$ - производит сравнение строк.
$void\ compareExchange(std::string\&\ a,\ std::string\&\ b)$ - сравнение и перестановка.
$void*\ nonRecursiveBatcherSort(void*\ arg)$ - нерекурсивная сортировка Бэтчера.
$void\ merge(void*\ arg)$ - слияние аргументов.

```
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <pthread.h>
#include <memory>

int numberOfThreads;
std::vector<std::string> vec;

class threadData_t {
  public:

  int lhs;
  int rhs;

  threadData_t() = default;
  threadData_t(int l, int r) {
    lhs = l;
    rhs = r;
  }
};


int compare(std::string& a, std::string& b) {
  if (a.size() < b.size()) {
    return -1; // left is less -1
  } else if (b.size() < a.size()) {
    return 1; // right is less 1
  }

  for (int i = 0; i < b.size(); i++) {
    if (a[i] < b[i]) {
      return -1;
    } else if (b[i] < a[i]) {
      return 1;
    }
  }

  return 0;
}
```

```
42
43 void compareExchange(std::string& a, std::string& b) {
44
45   if (b.compare(a) < 0)
46   std::swap(a, b);
47
48 }
49
50
51
52 void * nonRecursiveBatcherSort(void * arg) {
53
54   threadData_t * data = (threadData_t *) arg;
55
56   int l = data -> lhs;
57   int r = data -> rhs;
58
59   int N = r - l + 1;
60   for (int p = 1; p < N; p += p)
61   for (int k = p; k > 0; k /= 2)
62   for (int j = k % p; j + k < N; j += (k + k))
63   for (int i = 0; i < N - j - k; i++)
64   if ((j + i) / (p + p) == (j + i + k) / (p + p)){
65     compareExchange(vec[l + j + i], vec[l + j + i + k]);
66
67
68   }
69
70
71
72 }
73
74 void merge(void * arg) {
75   threadData_t * data = (threadData_t *) arg;
76
77   int l = data -> lhs;
78   int r = data -> rhs;
79
80   int mid = (l + r) / 2;
81
82   int size = r - l + 1;
83
84   std::vector<std::string> buf(size);
85
86   int first = l;
87   int second = mid + 1;
88
89   for (int i = 0; i < size; i++) {
90     if (first > mid) {
```

```
91        buf[i] = vec[second++];
92        continue;
93      }
94      if (second > r) {
95        buf[i] = vec[first++];
96        continue;
97      }
98      //if (compare(vec[first], vec[second]) < 0)
99      //buf[i] = vec[first++];
100     if (vec[second].compare(vec[first]) < 0)
101     buf[i] = vec[first++];
102     else
103     buf[i] = vec[second++];
104   }
105
106   for (int i = l, k = 0; i <= r; i++, k++) {
107     vec[i] = buf[k];
108   }
109 }
110
111 void mergeParts(void * arg) {
112
113   threadData_t * data = (threadData_t *) arg;
114
115   int l = data -> lhs;
116   int r = data -> rhs;
117
118   if (r - l > 0) {
119     int mid = (l + r) / 2;
120     threadData_t left(l, mid);
121     threadData_t right(mid + 1, r);
122     mergeParts(&left);
123     mergeParts(&right);
124
125     threadData_t merge0(l, r);
126     merge(&merge0);
127
128   }
129 }
130
131 int main(int argc, char * argv[]) {
132
133   if (argc < 2) {
134     numberOfThreads = 4;
135     std::cout << "NO ARGV GIVEN\n";
136   } else {
137     numberOfThreads = atoi(argv[1]);
138   }
139
```

```
140
141   std::ifstream input("input.txt");
142   std::ofstream output("output.txt");
143
144   int size;
145
146   input >> size;
147
148
149
150   for (int i = 0; i < size; i++) {
151     std::string buf;
152     input >> buf;
153     vec.push_back(buf);
154   }
155
156
157
158   pthread_t threads[numberOfThreads];
159
160   int step = size / numberOfThreads;
161   int l = 0;
162   int r = step;
163
164   for (int i = 0; i < numberOfThreads - 1; i++) {
165     threadData_t data(l + i * step, r + i * step);
166
167     int res = pthread_create(&threads[i], NULL, &nonRecursiveBatcherSort, &data);
168
169     if (res) {
170       std::cout << "Errors occured!\n";
171     }
172
173     pthread_join(threads[i], NULL);
174   }
175
176   threadData_t data(l + (numberOfThreads - 1) * step, size - 1);
177
178
179   int res = pthread_create(&threads[numberOfThreads - 1], NULL, &
          nonRecursiveBatcherSort, &data);
180
181   if (res) {
182     std::cout << "Errors occured!\n";
183   }
184
185   pthread_join(threads[numberOfThreads - 1], NULL);
186
187   threadData_t dataF(0, size - 1);
```

```
188   mergeParts(&dataF);
189
190   for (int i = 0; i < size; i++) {
191     output << vec[i] << '\n';
192   }
193
194
195
196   input.close();
197   output.close();
198
199
200   std::cout << "Program used " << numberOfThreads << " threads!\n";
201
202
203   return 0;
204 }
```

3 Тестирование

Входные данные

```
32
oVwYSdgkuISsvfmyoaNpciRIEJrFzLTuzguFtAfCvwdebamPjgEfEoFqrCitLFcfvNDTmuAnQHAz
aHuMmCznyftZNlbWxeONOupllkgmpxWIBongBwJkAczLpSBcxKjqakxSfIuqJIQTtaUiGUApPoijtzfGggSad
lQvmOtBhZqvnOeKcOIHVqYGuItccOYbpACnlfxJhDRzcjXjrmzTdsntt
WAUNrFOBhUArQSxhOkNJPMxnnKofICOSgHbPXeWCVwnWfsZpApboNYiEPRuAZyBhfeSzFDFOQPDpfVqInVXgv
WjQAKyYxwHWuLquwTwErunIiQtnMRDLYbIhxFboFwlyJTyZSPuvRQsxDTbplUhENSUiJhLkSVMSRNJFniPB
AVqqvEVQMXMghBvjcbjVArvhiLzAZQOkWDiLSoBVcKqquJzhZQBEgQeUTim
IrAbEhVKyYaJaVAxbnlHlGxvPHRPCNLcufilLnKNzWUs
lPwfQvBqQvkdVmoOUdIDknALIwJnzgvjKmJfOhCXbdfScXTcCysBoXsKMyKkd
TTERsfJj
ObjTSZYbAqWasFDxwKaPGuzxKEQAztAncbRHbnFVoJbpUqINcEvPnVyWiyCVWPg
RuzOnTatLQdhlSbDewPIkItxaMZjTYEKKgKPBKqmDcUWhzTGjfOZrnDA
qVaLIGAOxQRrzMbrHKJEmdcwKRHgNDLvRqRXuZlOEAKRIpVJMcpkHFbvEeziMhuPRGiNLRBg
sCNDSTLZcMbnaahDOirFjZcCSnQfjEWjmpvfbqKXZvOfoedFiMNAubgYVjakyczzdPvSJ
PCJlEjFjDxPodQtVvmDFFwOhRJLijT
aqhgteXjxkUsjlQarbow
SdzmxRISTqClJuLkgkTQLuVLQvfsYbxQgobhhAHompROH
DKujSuwhLvvbwjjvGKWvtEnXbbOBLEACSGHu
vfSZQuOHdztxiZVgbdAVsPlIlzLEMOGkKxgRdgHOqxAPdnVwkzfUkux
wwoXfRqInxjFgICghEnXObQnLqeGhadUwgbuQtZwfofsFqFQMTMTfxBCGlGRBxrLjAWQ
NBXMmxFntniEJbgmmqqavmoGJmWrmMkknXxDRasArzrXkAfwrhBlyzYiXqLG
mlDkFSPJOAdChjihpVpnzaQJcMOtqBQAlRvePwfEERllopFosdOoOnSQoTXoUVqGfiLsFygAoRCUvAosixYeE
wjkpcRHwlJlRLsPLbEviccwXFPZUNzLImqsSatqUQObNlzQwIAjtuYEAbJKitJBFQXCGKUSeTtLOROEtEtexf
MpKCbEDqQaUgH
krXjycQbYrMmHyzGaKLsySxVkSwUVLVETCRNxUijRbHhBaYwGKIxPSybOtLKVBaGpscnbJbSFpexScMURmPZI
NBivBXYhAmtPcbfjMMIqYrezdsZewBwpwmUboCHpEG
NyLnAeXKJFJRhrKnuzhNcXtTWigMWGutuqzUyUeeDEJlsFefwbQjhbmhiTIjPSGqpQeFswepqh
KTAPuZlINloHqIvdAycmJBSkIeFgEKyfGUtrGWHpx
Khjyu
EovbvfnsbSohnRVQ
WlGTiJZ
HCIzEhgPlhwMWcugvniDkue
vYobaTHEVGVknYAhKvdDNNZGoqvgDVtu
```

Выходные данные

wwoXfRqInxjFgICghEnXObQnLqeGhadUwgbuQtZwfofsFqFQMTMTfxBCGlGRBxrLjAWQ
wjkpcRHwlJlRLsPLbEviccwXFPZUNzLImqsSatqUQObNlzQwIAjtuYEAbJKitJBFQXCGKUSeTtLOROEtEtexfI
vfSZQuOHdztxiZVgbdAVsPlIlzLEMOGkKxgRdgHOqxAPdnVwkzfUkux
vYobaTHEVGVknYAhKvdDNNZGoqvgDVtu
sCNDSTLZcMbnaahDOirFjZcCSnQfjEWjmpvfbqKXZvOfoedFiMNAubgYVjakyczzdPvSJ
qVaLIGAOxQRrzMbrHKJEmdcwKRHgNDLvRqRXuZlOEAKRIpVJMcpkHFbvEeziMhuPRGiNLRBg
oVwYSdgkuISsvfmyoaNpciRIEJrFzLTuzguFtAfCvwdebamPjgEfEoFqrCitLFcfvNDTmuAnQHAz
mlDkFSPJOAdChjihpVpnzaQJcMOtqBQAlRvePwfEERllopFosdOoOnSQoTXoUVqGfiLsFygAoRCUvAosixYeEV
lQvmOtBhZqvnOeKcOIHVqYGuItccOYbpACnlfxJhDRzcjXjrmzTdsntt
lPwfQvBqQvkdVmoOUdIDknALIwJnzgvjKmJfOhCXbdfScXTcCysBoXsKMyKkd
krXjycQbYrMmHyzGaKLsySxVkSwUVLVETCRNxUijRbHhBaYwGKIxPSybOtLKVBaGpscnbJbSFpexScMURmPZIh
aqhgteXjxkUsjlQarbow
aHuMmCznyftZNlbWxeONOupllkgmpxWIBongBwJkAczLpSBcxKjqakxSfIuqJIQTtaUiGUApPoijtzfGggSadT
WlGTiJZ
WjQAKyYxwHWuLquwTwErunIiQtnMRDLYbIhxFboFwlyJTyZSPuvRQsxDTbplUhENSUiJhLkSVMSRNJFniPB
WAUNrFOBhUArQSxhOkNJPMxnnKofICOSgHbPXeWCVwnWfsZpApboNYiEPRuAZyBhfeSzFDFOQPDpfVqInVXgvS
TTERsfJj
SdzmxRISTqClJuLkgkTQLuVLQvfsYbxQgobhhAHompROH
RuzOnTatLQdhlSbDewPIkItxaMZjTYEKKgKPBKqmDcUWhzTGjfOZrnDA
PCJlEjFjDxPodQtVvmDFFwOhRJLijT
ObjTSZYbAqWasFDxwKaPGuzxKEQAztAncbRHbnFVoJbpUqINcEvPnVyWiyCVWPg
NyLnAeXKJFJRhrKnuzhNcXtTWigMWGutuqzUyUeeDEJlsFefwbQjhbmhiTIjPSGqpQeFswepqh
NBivBXYhAmtPcbfjMMIqYrezdsZewBwpwmUboCHpEG
NBXMmxFntniEJbgmmqqavmoGJmWrmMkknXxDRasArzrXkAfwrhBlyzYiXqLG
MpKCbEDqQaUgH
Khjyu
KTAPuZlINloHqIvdAycmJBSkIeFgEKyfGUtrGWHpx
IrAbEhVKyYaJaVAxbnlHlGxvPHRPCNLcufilLnKNzWUs
HCIzEhgPlhwMWcugvniDkue
EovbvfnsbSohnRVQ
DKujSuwhLvvbwjjvGKWvtEnXbbOBLEACSGHu
AVqqvEVQMXMghBvjcbjVArvhiLzAZQOkWDiLSoBVcKqquJzhZQBEgQeUTim

4 Выводы

Одно из главных отличий использования многопоточности от использования процессов в том, что потоки делят между собой одно адресное пространство, но параллельная обработка одних и тех же данных требует ответственного подхода к созданию программы.