Министерства науки и высшего образования Российской Федерации

федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

*Факультет программной инженерии и компьютерной техники*

**Лабораторная работа №1**

Вариант: *CreateProcess + io-thpt-write (8K) + dedup*

Выполнил:

Ценеков Дмитрий Алексеевич

Группа: P3310

г. Санкт-Петербург

2024 г.

**Оглавление**

[**Текст задания** 3](#_Toc183086182)

[**Листинг исходного кода** 4](#_Toc183086183)

[**Bench1.cpp** 4](#_Toc183086184)

[**Bench2.cpp** 5](#_Toc183086185)

[**Shell.cpp** 6](#_Toc183086186)

[**Tester.py** 8](#_Toc183086187)

[**Вывод** 17](#_Toc183086188)

## **Текст задания**

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

## **Листинг исходного код****а**

### **Bench1.cpp**

  1 │ ***// Copyright 2024 dim0n4eg***  
  2 │   
  3 │ ***#include <windows.h>***  
  4 │   
  5 │ ***#include <chrono>***  
  6 │ ***#include <cstdlib>***  
  7 │ ***#include <iostream>***  
  8 │ ***#include <vector>***  
  9 │   
 10 │ **const** size\_t BLOCK\_SIZE **=** 8192**;**  
 11 │   
 12 │ **int** main**(int** argc**,** **char\*** argv**[])** **{**  
 13 │  **if** **(**argc **!=** 3**)** **{**  
 14 │  std**::**cerr **<<** "Usage: " **<<** argv**[**0**]** **<<** " <filename> <size\_in\_MB>"  
 15 │  **<<** std**::**endl**;**  
 16 │  **return** 1**;**  
 17 │  **}**  
 18 │   
 19 │  **const** **char\*** filename **=** argv**[**1**];**  
 20 │  size\_t size\_in\_MB **=** std**::**atoi**(**argv**[**2**]);**  
 21 │  size\_t TOTAL\_SIZE **=** size\_in\_MB **\*** 1024 **\*** 1024**;**  
 22 │   
 23 │  std**::**vector**<char>** buffer**(**BLOCK\_SIZE**,** 'T'**);**  
 24 │   
 25 │  HANDLE hFile **=**  
 26 │  **CreateFile(**filename**,** GENERIC\_WRITE**,** 0**,** NULL**,** CREATE\_ALWAYS**,**  
 27 │  FILE\_FLAG\_WRITE\_THROUGH **|** FILE\_FLAG\_NO\_BUFFERING**,** NULL**);**  
 28 │   
 29 │  **if** **(**hFile **==** INVALID\_HANDLE\_VALUE**)** **{**  
 30 │  std**::**cerr **<<** "Error opening file for writing: " **<<** **GetLastError()**  
 31 │  **<<** std**::**endl**;**  
 32 │  **return** 1**;**  
 33 │  **}**  
 34 │   
 35 │  auto start\_time **=** std**::**chrono**::**high\_resolution\_clock**::**now**();**  
 36 │   
 37 │  size\_t total\_written **=** 0**;**  
 38 │  DWORD bytes\_written**;**  
 39 │   
 40 │  **while** **(**total\_written **<** TOTAL\_SIZE**)** **{**  
 41 │  **if** **(!WriteFile(**hFile**,** buffer**.**data**(),** BLOCK\_SIZE**,** **&**bytes\_written**,** NULL**))** **{**  
 42 │  std**::**cerr **<<** "Error writing to file: " **<<** **GetLastError()** **<<** std**::**endl**;**  
 43 │  **CloseHandle(**hFile**);**  
 44 │  **return** 1**;**  
 45 │  **}**  
 46 │  total\_written **+=** bytes\_written**;**  
 47 │  **}**  
 48 │   
 49 │  auto end\_time **=** std**::**chrono**::**high\_resolution\_clock**::**now**();**  
 50 │  **CloseHandle(**hFile**);**  
 51 │   
 52 │  std**::**chrono**::**duration**<double>** duration **=** end\_time **-** start\_time**;**  
 53 │   
 54 │  **double** throughput **=** **(**TOTAL\_SIZE **/** **(**1024.0 **\*** 1024.0**))** **/** duration**.**count**();**  
 55 │  std**::**cout **<<** "Write throughput: " **<<** throughput **<<** " MB/s" **<<** std**::**endl**;**  
 56 │   
 57 │  **return** 0**;**  
 58 │ **}**  
 59 │

### **Bench2.cpp**

  1 │ ***// Copyright 2024 dim0n4eg***  
  2 │   
  3 │ ***#include <iostream>***  
  4 │ ***#include <random>***  
  5 │ ***#include <set>***  
  6 │ ***#include <vector>***  
  7 │   
  8 │ std**::**vector**<int>** generateRandomData**(**size\_t size**)** **{**  
  9 │  std**::**vector**<int>** data**;**  
 10 │  std**::**random\_device rd**;**  
 11 │  std**::**mt19937 gen**(**rd**());**  
 12 │  std**::**uniform\_int\_distribution**<>** dis**(**0**,** 99**);**  
 13 │   
 14 │  **for** **(**size\_t i **=** 0**;** i **<** size**;** **++**i**)** **{**  
 15 │  data**.**push\_back**(**dis**(**gen**));**  
 16 │  **}**  
 17 │  **return** data**;**  
 18 │ **}**  
 19 │   
 20 │ size\_t deduplicate**(const** std**::**vector**<int>&** data**)** **{**  
 21 │  std**::**set**<int>** unique\_data**(**data**.**begin**(),** data**.**end**());**  
 22 │  **return** unique\_data**.**size**();**  
 23 │ **}**  
 24 │   
 25 │ **int** main**(int** argc**,** **char\*** argv**[])** **{**  
 26 │  **if** **(**argc **!=** 3**)** **{**  
 27 │  std**::**cerr **<<** "Usage: " **<<** argv**[**0**]** **<<** " <size\_in\_MB> <times>" **<<** std**::**endl**;**  
 28 │  **return** 1**;**  
 29 │  **}**  
 30 │   
 31 │  size\_t array\_size **=** std**::**stoul**(**argv**[**1**])** **/** sizeof**(int)** **\*** 1024 **\*** 1024**;**  
 32 │  **int** times **=** std**::**stoi**(**argv**[**2**]);**  
 33 │   
 34 │  size\_t unique\_count **=** 0**;**  
 35 │  **for** **(int** i **=** 0**;** i **<** times**;** **++**i**)** **{**  
 36 │  std**::**vector**<int>** data **=** generateRandomData**(**array\_size**);**  
 37 │  unique\_count **+=** deduplicate**(**data**);**  
 38 │  **}**  
 39 │   
 40 │  std**::**cout **<<** ": " **<<** unique\_count **<<** std**::**endl**;** ***// anti-optimization***  
 41 │   
 42 │  **return** 0**;**  
 43 │ **}**  
 44 │

### **Shell.cpp**

  1 │ ***// Copyright 2024 dim0n4eg***  
  2 │   
  3 │ ***#include <stdlib.h>***  
  4 │ ***#include <string.h>***  
  5 │ ***#include <time.h>***  
  6 │ ***#include <windows.h>***  
  7 │   
  8 │ ***#include <cstdio>***  
  9 │   
  10 │ ***#define MAX\_ARGS 64***  
  11 │ ***#define BUFFER\_SIZE 4096***  
  12 │   
  13 │ **void** execute\_cd**(char** **\***path**,** HANDLE hOut**,** HANDLE hErr**)** **{**  
  14 │  **if** **(SetCurrentDirectory(**path**)** **==** 0**)** **{**  
  15 │  **const** **char** **\***errorMsg **=** "Error: Cannot change directory.\n"**;**  
  16 │  DWORD bytesWritten**;**  
  17 │  **WriteFile(**hErr**,** errorMsg**,** strlen**(**errorMsg**),** **&**bytesWritten**,** NULL**);**  
  18 │  **return;**  
  19 │  **}**  
  20 │ **}**  
  21 │   
  22 │ **void** execute\_ls**(**HANDLE hOut**,** HANDLE hErr**)** **{**  
  23 │  WIN32\_FIND\_DATA findFileData**;**  
  24 │  HANDLE hFind **=** **FindFirstFile(**"\*"**,** **&**findFileData**);**  
  25 │   
  26 │  **if** **(**hFind **==** INVALID\_HANDLE\_VALUE**)** **{**  
  27 │  **const** **char** **\***errorMsg **=** "Error: Unable to list directory.\n"**;**  
  28 │  DWORD bytesWritten**;**  
  29 │  **WriteFile(**hErr**,** errorMsg**,** strlen**(**errorMsg**),** **&**bytesWritten**,** NULL**);**  
  30 │  **return;**  
  31 │  **}**  
  32 │   
  33 │  **do** **{**  
  34 │  DWORD bytesWritten**;**  
  35 │  **char** outputBuffer**[**MAX\_PATH **+** 2**];**  
  36 │  snprintf**(**outputBuffer**,** sizeof**(**outputBuffer**),** "%s\n"**,**  
  37 │  findFileData**.**cFileName**);**  
  38 │   
  39 │  **WriteFile(**hOut**,** outputBuffer**,** strlen**(**outputBuffer**),** **&**bytesWritten**,** NULL**);**  
  40 │  **}** **while** **(FindNextFile(**hFind**,** **&**findFileData**)** **!=** 0**);**  
  41 │   
  42 │  **FindClose(**hFind**);**  
  43 │ **}**  
  44 │ **void** execute\_program**(char** **\***program**,** **char** **\*\***args**,** HANDLE hIn**,** HANDLE hOut**,**  
  45 │  HANDLE hErr**)** **{**  
  46 │  STARTUPINFO si**;**  
  47 │  PROCESS\_INFORMATION pi**;**  
  48 │  memset**(&**si**,** 0**,** sizeof**(**si**));**  
  49 │  si**.**cb **=** sizeof**(**si**);**  
  50 │  si**.**dwFlags **|=** STARTF\_USESTDHANDLES**;**  
  51 │  si**.**hStdInput **=** hIn**;**  
  52 │  si**.**hStdOutput **=** hOut**;**  
  53 │  si**.**hStdError **=** hErr**;**  
  54 │   
  55 │  memset**(&**pi**,** 0**,** sizeof**(**pi**));**  
  56 │   
  57 │  **char** cmd**[**1024**]** **=** ""**;**  
  58 │  snprintf**(**cmd**,** sizeof**(**cmd**),** "%s"**,** program**);**  
  59 │  **for** **(int** i **=** 1**;** args**[**i**]** **!=** NULL**;** i**++)** **{**  
  60 │  snprintf**(**cmd **+** strlen**(**cmd**),** sizeof**(**cmd**)** **-** strlen**(**cmd**),** " %s"**,** args**[**i**]);**  
  61 │  **}**  
  62 │   
  63 │  clock\_t start **=** **clock();**  
  64 │  **if** **(!CreateProcess(**NULL**,** cmd**,** NULL**,** NULL**,** TRUE**,** 0**,** NULL**,** NULL**,** **&**si**,** **&**pi**))** **{**  
  65 │  printf**(**"Error: Failed to start program %s\n"**,** program**);**  
  66 │  **return;**  
  67 │  **}**  
  68 │  **WaitForSingleObject(**pi**.**hProcess**,** INFINITE**);**  
  69 │   
  70 │  clock\_t end **=** **clock();**  
  71 │  **double** time\_spent **=** static\_cast**<double>(**end **-** start**)** **/** CLOCKS\_PER\_SEC**;**  
  72 │  printf**(**"Execution time of %s: %.3f seconds\n"**,** program**,** time\_spent**);**  
  73 │   
  74 │  **CloseHandle(**pi**.**hProcess**);**  
  75 │  **CloseHandle(**pi**.**hThread**);**  
  76 │ **}**  
  77 │   
  78 │ **void** execute\_command**(char** **\***command**,** HANDLE hIn**,** HANDLE hOut**,** HANDLE hErr**)** **{**  
  79 │  **char** **\***args**[**MAX\_ARGS**];**  
  80 │  **char** **\***saveptr**;**  
  81 │  **char** **\***token **=** strtok\_r**(**command**,** " "**,** **&**saveptr**);**  
  82 │  **int** argCount **=** 0**;**  
  83 │  **while** **(**token **!=** NULL**)** **{**  
  84 │  args**[**argCount**++]** **=** token**;**  
  85 │  token **=** strtok\_r**(**NULL**,** " "**,** **&**saveptr**);**  
  86 │  **}**  
  87 │  args**[**argCount**]** **=** NULL**;**  
  88 │   
  89 │  **if** **(**strcmp**(**args**[**0**],** "cd"**)** **==** 0**)** **{**  
  90 │  **if** **(**argCount **>** 1**)** **{**  
  91 │  execute\_cd**(**args**[**1**],** hOut**,** hErr**);**  
  92 │  **}**  
  93 │  **}** **else** **if** **(**strcmp**(**args**[**0**],** "ls"**)** **==** 0**)** **{**  
  94 │  execute\_ls**(**hOut**,** hErr**);**  
  95 │  **}** **else** **if** **(**args**[**0**][**0**]** **==** '.' **&&** args**[**0**][**1**]** **==** '/'**)** **{**  
  96 │  execute\_program**(**args**[**0**],** args**,** hIn**,** hOut**,** hErr**);**  
  97 │  **}** **else** **{**  
  98 │  fprintf**(**stderr**,** "Command not recognized: %s\n"**,** args**[**0**]);**  
  99 │  **}**  
 100 │ **}**  
 101 │   
 102 │ HANDLE open\_file**(char** **\***filename**,** DWORD accessMode**,** DWORD creationDisposition**)** **{**  
 103 │  SECURITY\_ATTRIBUTES sa**;**  
 104 │  sa**.**nLength **=** sizeof**(**sa**);**  
 105 │  sa**.**lpSecurityDescriptor **=** NULL**;**  
 106 │  sa**.**bInheritHandle **=** TRUE**;**  
 107 │   
 108 │  HANDLE fileHandle **=**  
 109 │  CreateFileA**(**filename**,** accessMode**,** FILE\_SHARE\_WRITE**,** **&**sa**,**  
 110 │  creationDisposition**,** FILE\_ATTRIBUTE\_NORMAL**,** NULL**);**  
 111 │  **if** **(**fileHandle **==** INVALID\_HANDLE\_VALUE**)** **{**  
 112 │  fprintf**(**stderr**,** "Failed to open file: %s!\n"**,** filename**);**  
 113 │  **}**  
 114 │  **return** fileHandle**;**  
 115 │ **}**  
 116 │   
 117 │ **void** parse\_redirection**(char** **\***command**,** HANDLE **\***hOut**,** HANDLE **\***hErr**)** **{**  
 118 │  **char** **\***redirect**;**  
 119 │  **char** **\***filename**;**  
 120 │   
 121 │  **while** **((**redirect **=** strpbrk**(**command**,** "12>"**)))** **{**  
 122 │  **int** type\_of\_redirect **=** 0**;**  
 123 │  **if** **(\***redirect **==** '>'**)** **{**  
 124 │  type\_of\_redirect **=** 1**;**  
 125 │  filename **=** redirect **+** 1**;**  
 126 │  **}** **else** **if** **((\***redirect **==** '1' **||** **\***redirect **==** '2'**)** **&&**  
 127 │  **\*(**redirect **+** 1**)** **==** '>'**)** **{**  
 128 │  **if** **(\***redirect **==** '1'**)**  
 129 │  type\_of\_redirect **=** 1**;**  
 130 │  **else**  
 131 │  type\_of\_redirect **=** 2**;**  
 132 │  filename **=** redirect **+** 2**;**  
 133 │  **}** **else** **{**  
 134 │  command **=** redirect **+** 1**;**  
 135 │  **continue;**  
 136 │  **}**  
 137 │   
 138 │  **while** **(\***filename **&&** isspace**(\***filename**))** **{**  
 139 │  filename**++;**  
 140 │  **}**  
 141 │   
 142 │  **char** **\***end **=** filename**;**  
 143 │  **while** **(\***end **&&** **!**isspace**(\***end**))** **{**  
 144 │  end**++;**  
 145 │  **}**  
 146 │   
 147 │  **char** saved **=** **\***end**;**  
 148 │  **\***end **=** '\0'**;**  
 149 │  **if** **(**type\_of\_redirect **==** 1**)** **{**  
 150 │  **\***hOut **=** open\_file**(**filename**,** GENERIC\_WRITE**,** CREATE\_ALWAYS**);**  
 151 │  **if** **(\***hOut **==** INVALID\_HANDLE\_VALUE**)** **{**  
 152 │  **\***hOut **=** **GetStdHandle(**STD\_OUTPUT\_HANDLE**);**  
 153 │  **}**  
 154 │  **}** **else** **if** **(**type\_of\_redirect **==** 2**)** **{**  
 155 │  **\***hErr **=** open\_file**(**filename**,** GENERIC\_WRITE**,** CREATE\_ALWAYS**);**  
 156 │  **if** **(\***hErr **==** INVALID\_HANDLE\_VALUE**)** **{**  
 157 │  **\***hErr **=** **GetStdHandle(**STD\_ERROR\_HANDLE**);**  
 158 │  **}**  
 159 │  **}**  
 160 │  **\***end **=** saved**;**  
 161 │   
 162 │  memset**(**redirect**,** ' '**,** end **-** redirect**);**  
 163 │  **}**  
 164 │ **}**  
 165 │   
 166 │ **void** parse\_command**(char** **\***input**,** **char** **\*\***commands**,** **int** **\***cmdCount**)** **{**  
 167 │  **char** **\***saveptr**;**  
 168 │  **char** **\***token **=** strtok\_r**(**input**,** "|"**,** **&**saveptr**);**  
 169 │  **while** **(**token **!=** NULL**)** **{**  
 170 │  commands**[(\***cmdCount**)++]** **=** token**;**  
 171 │  token **=** strtok\_r**(**NULL**,** "|"**,** **&**saveptr**);**  
 172 │  **}**  
 173 │ **}**  
 174 │   
 175 │ **void** execute\_pipeline**(char** **\***input**)** **{**  
 176 │  **char** **\***commands**[**64**];**  
 177 │  **int** cmdCount **=** 0**;**  
 178 │  parse\_command**(**input**,** commands**,** **&**cmdCount**);**  
 179 │   
 180 │  HANDLE hIn **=** **GetStdHandle(**STD\_INPUT\_HANDLE**);**  
 181 │  HANDLE hOut**,** hErr**;**  
 182 │  HANDLE hPipe**[**2**]** **=** **{**INVALID\_HANDLE\_VALUE**,** INVALID\_HANDLE\_VALUE**};**  
 183 │   
 184 │  **for** **(int** i **=** 0**;** i **<** cmdCount**;** i**++)** **{**  
 185 │  hErr **=** **GetStdHandle(**STD\_ERROR\_HANDLE**);**  
 186 │  hOut **=** **GetStdHandle(**STD\_OUTPUT\_HANDLE**);**  
 187 │   
 188 │  **if** **(**i **<** cmdCount **-** 1**)** **{**  
 189 │  SECURITY\_ATTRIBUTES saAttr**;**  
 190 │  saAttr**.**nLength **=** sizeof**(**SECURITY\_ATTRIBUTES**);**  
 191 │  saAttr**.**bInheritHandle **=** TRUE**;**  
 192 │  saAttr**.**lpSecurityDescriptor **=** NULL**;**  
 193 │  **if** **(!CreatePipe(&**hPipe**[**0**],** **&**hPipe**[**1**],** **&**saAttr**,** 0**))**  
 194 │  printf**(**"StdoutRd CreatePipe"**);**  
 195 │  hOut **=** hPipe**[**1**];**  
 196 │  **}**  
 197 │   
 198 │  parse\_redirection**(**commands**[**i**],** **&**hOut**,** **&**hErr**);**  
 199 │   
 200 │  execute\_command**(**commands**[**i**],** hIn**,** hOut**,** hErr**);**  
 201 │   
 202 │  **if** **(**hOut **!=** **GetStdHandle(**STD\_OUTPUT\_HANDLE**))** **{**  
 203 │  **CloseHandle(**hOut**);**  
 204 │  **}**  
 205 │  **if** **(**hErr **!=** **GetStdHandle(**STD\_ERROR\_HANDLE**))** **{**  
 206 │  **CloseHandle(**hErr**);**  
 207 │  **}**  
 208 │   
 209 │  **if** **(**i **<** cmdCount **-** 1**)** **{**  
 210 │  hIn **=** hPipe**[**0**];**  
 211 │  **}**  
 212 │  **}**  
 213 │   
 214 │  **if** **(**hPipe**[**0**]** **!=** INVALID\_HANDLE\_VALUE**)** **{**  
 215 │  **CloseHandle(**hPipe**[**0**]);**  
 216 │  **}**  
 217 │  **if** **(**hPipe**[**1**]** **!=** INVALID\_HANDLE\_VALUE**)** **{**  
 218 │  **CloseHandle(**hPipe**[**1**]);**  
 219 │  **}**  
 220 │ **}**  
 221 │   
 222 │ **void** print\_prompt**()** **{**  
 223 │  **char** cwd**[**1024**];**  
 224 │  **if** **(GetCurrentDirectory(**sizeof**(**cwd**),** cwd**))** **{**  
 225 │  printf**(**"%s$ "**,** cwd**);**  
 226 │  **}** **else** **{**  
 227 │  printf**(**"$ "**);**  
 228 │  **}**  
 229 │ **}**  
 230 │   
 231 │ **int** main**()** **{**  
 232 │  **char** input**[**1024**];**  
 233 │   
 234 │  **while** **(**1**)** **{**  
 235 │  print\_prompt**();**  
 236 │  fflush**(**stdout**);**  
 237 │   
 238 │  **if** **(!**fgets**(**input**,** sizeof**(**input**),** stdin**))** **break;**  
 239 │  input**[**strcspn**(**input**,** "\n"**)]** **=** 0**;**  
 240 │   
 241 │  **if** **(**strcmp**(**input**,** "exit"**)** **==** 0**)** **{**  
 242 │  **break;**  
 243 │  **}**  
 244 │   
 245 │  execute\_pipeline**(**input**);**  
 246 │  **}**  
 247 │  **return** 0**;**  
 248 │ **}**  
 249 │

### **Tester.py**

  1 │ **import** os  
  2 │ **import** subprocess  
  3 │ **import** time  
  4 │ from pathlib **import** **Path**  
  5 │ **import** re  
  6 │   
  7 │ **def** compile\_sources**(**output\_filename**,** input\_array\_cpp\_files**):**  
  8 │  build\_dir **=** **Path(**"build"**)**  
  9 │  build\_dir**.mkdir(**exist\_ok**=True)**  
 10 │  compile\_arr **=** **[**"g++"**,** "-o"**,** str**(**build\_dir **/** output\_filename**)]** **+** input\_array\_cpp\_files  
 11 │  **print(**" "**.join(**compile\_arr**))**  
 12 │  result **=** subprocess**.**run**(**compile\_arr**,** capture\_output**=True,** text**=True)**  
 13 │  **if** result**.**returncode **!=** 0**:**  
 14 │  **print(**"Compilation error:"**)**  
 15 │  **print(**result**.**stderr**)**  
 16 │  exit**(**1**)**  
 17 │  **else:**  
 18 │  **return** **True**  
 19 │   
 20 │ **def** run\_and\_validate**(**output\_filename**,** input\_data**,** validate\_output**):**  
 21 │  executable\_path **=** **Path(**"build"**)** **/** output\_filename  
 22 │  **if** not executable\_path**.**exists**():**  
 23 │  **print(**f"Executable file {output\_filename} not found."**)**  
 24 │  **return**  
 25 │    
 26 │  **for** i**,** input\_value **in** enumerate**(**input\_data**):**  
 27 │  input\_lines **=** input\_value**.**splitlines**()**  
 28 │  process **=** subprocess**.Popen([**str**(**executable\_path**)],** stdin**=**subprocess**.**PIPE**,** stdout**=**subprocess**.**PIPE**,** stderr**=**subprocess**.**PIPE**,** text**=True)**  
 29 │  **for** line **in** input\_lines**:**  
 30 │  process**.**stdin**.write(**line **+** '\n'**)**  
 31 │  process**.**stdin**.**flush**()**  
 32 │  time**.**sleep**(**0.5**)**  
 33 │  process**.**stdin**.**close**()**  
 34 │  stdout**,** stderr **=** process**.**communicate**()**  
 35 │  **if** process**.**returncode **!=** 0**:**  
 36 │  **print(**f"Error during test {(i+1)} with input data: {input\_value}"**)**  
 37 │  **print(**stderr**)**  
 38 │  exit**(**1**)**  
 39 │  validate\_output**(**i**+**1**,** input\_value**,** stdout**)**  
 40 │   
 41 │ **def** main**():**  
 42 │  output\_filename **=** "main.exe"  
 43 │  output\_benchmark **=** "benchmark.exe"  
 44 │  output\_benchmark2 **=** "benchmark2.exe"  
 45 │  cpp\_source **=** **[**str**(**file**)** **for** file **in** **Path(**"./source"**).**rglob**(**"\*.cpp"**)]**  
 46 │  cpp\_benchmark **=** **[**str**(**file**)** **for** file **in** **Path(**"./benchmark"**).**rglob**(**"\*1.cpp"**)]**  
 47 │  cpp\_benchmark2 **=** **[**str**(**file**)** **for** file **in** **Path(**"./benchmark"**).**rglob**(**"\*2.cpp"**)]**  
 48 │  input\_data **=** **[**"cd build\n./benchmark.exe output\_file.txt 1\nexit\n"**,** "cd build\n./benchmark2.exe 10 2\nexit\n"**]**  
 49 │  output\_data **=** **[**"\_\_PATH\_\_$ \_\_PATH\_\_\\build$ Write throughput: \_\_NUM\_\_ MB/s\nExecution time of ./benchmark.exe: \_\_NUM\_\_ seconds\n\_\_PATH\_\_\\build$ "**,** "\_\_PATH\_\_$ \_\_PATH\_\_\\build$ : \_\_NUM\_\_\nExecution time of ./benchmark2.exe: \_\_NUM\_\_ seconds\n\_\_PATH\_\_\\build$ "**]**  
 50 │    
 51 │  **def** validate\_output**(**test\_number**,** input\_value**,** output\_value**):**  
 52 │  cout **=** output\_value**.**replace**(**os**.**path**.**dirname**(**os**.**path**.**abspath**(**\_\_file\_\_**)),** "\_\_PATH\_\_"**)**  
 53 │  cout **=** re**.**sub**(**r": \d+(\.\d+)?"**,** ": \_\_NUM\_\_"**,** cout**)**  
 54 │  **if** cout **==** output\_data**[**test\_number**-**1**]:**  
 55 │  **print(**f"Test {test\_number} passed"**)**  
 56 │  **else:**  
 57 │  **print(**f"Test {test\_number} failed:\nInput data: {input\_value}\nResult: {cout}\nExpected: {output\_data[test\_number-1]}"**)**  
 58 │  exit**(**1**)**  
 59 │   
 60 │  **if** compile\_sources**(**output\_filename**,** cpp\_source**):**  
 61 │  **if** compile\_sources**(**output\_benchmark**,** cpp\_benchmark**):**  
 62 │  **if** compile\_sources**(**output\_benchmark2**,** cpp\_benchmark2**):**  
 63 │  run\_and\_validate**(**output\_filename**,** input\_data**,** validate\_output**)**  
 64 │   
 65 │ **if** \_\_name\_\_ **==** "\_\_main\_\_"**:**  
 66 │  main**()**  
 67 │

## **Предположения о свойствах программ-нагрузчиков**

Сделаем несколько предположений о свойствах программы-нагрузчика для измерение пропускной способности на последовательную запись

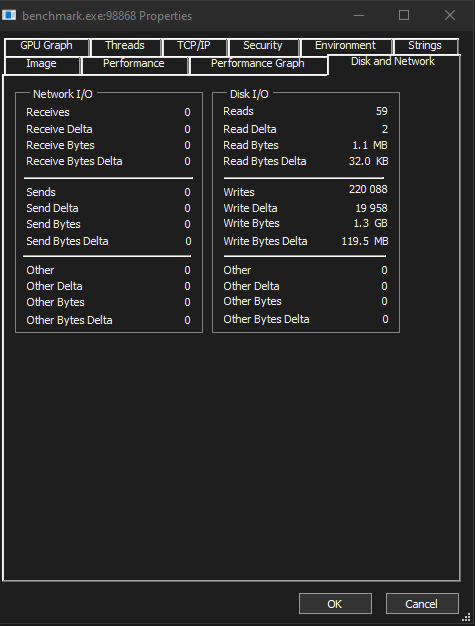
* Ожидалось бы, что программа будет иметь высокий уровень записи.
* CPU должно быть загружено на значительное время, но с преобладанием времени работы в режиме ядра из-за операций ввода-вывода.
* Высокое количество страничных ошибок, из-за большого количества IO-операций.

Сделаем несколько предположений о свойствах программы-нагрузчика для дедупликация элементов в массиве:

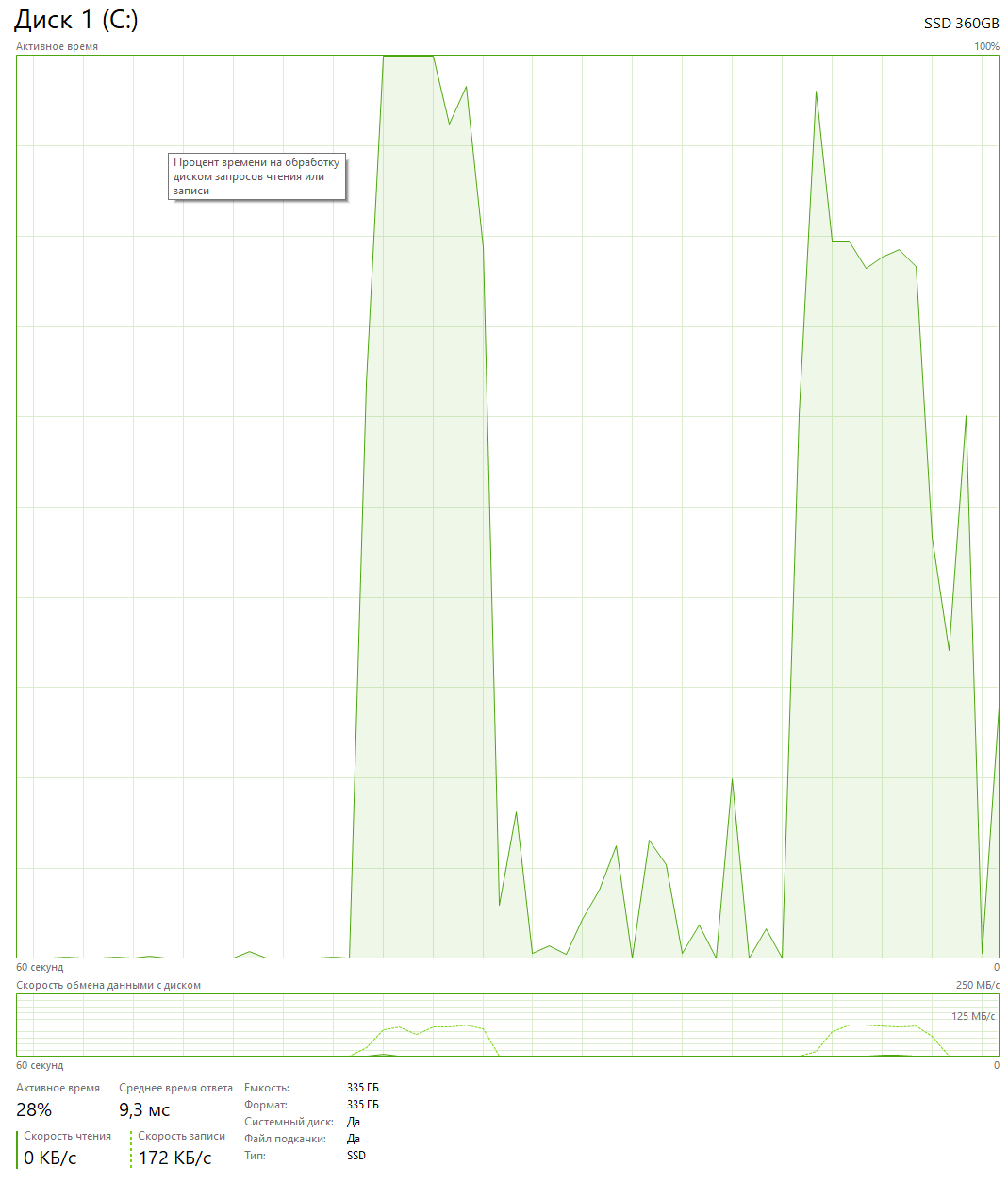
* Ожидалось бы, что программа будет активно использовать CPU с высоким значением User Time.
* Временные затраты на операции ввода-вывода должны отсутствовать, так как все данные уже находятся в оперативной памяти.
* Высокое количество страничных ошибок, так как процесс во время генерации массива будет обращаться к вновь выделенным страницам в первый раз.

## **Результаты** **измерений и метрик программ-нагрузчиков, полученных инструментами мониторинга**

Сделаем несколько измерений программы-нагрузчика для измерение пропускной способности на последовательную запись







1. Disk I/O:

* Поскольку программа работает с внешней памятью, ожидается значительное количество операций чтения и записи. Число чтений (0) и записей (131 073, 1073741824 байт/8192 байт = 131072 операций записи) указывает на то, что программа активно использует диск для загрузки и сохранения данных.
* Объем записанных данных: 1.3 Gb (Записывал 1Gb).

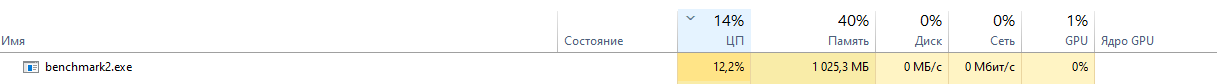
2. CPU:

* Kernel Time и User Time: Время, затраченное на выполнение в режиме ядра (Kernel Time) и в пользовательском режиме (User Time), может дать представление о том, сколько времени программа проводит в обработке данных по сравнению с операциями ввода-вывода. В данном случае, время в режиме ядра (0:02.750) значительно больше времени в пользовательском режиме (0:00.125), что может указывать на большое количество операций ввода-вывода.

3. Virtual Memory:

* Page Faults: Небольшое количество страничных ошибок (973) может указывать на то, что ядро оптимизировано на запись в файл и не требует отображения файла в Virtual Memory.

Сделаем несколько измерений программы-нагрузчика для дедупликация элементов в массиве, полученных инструментами мониторинга:





1. CPU:

* Kernel Time и User Time: программа имеет значительное время выполнения в пользовательском режиме (User Time), так как дедупликация — это вычислительно интенсивная задача.
* Cycles: Высокое количество циклов (195 271 749 758) указывает на интенсивное использование процессора.

2. I/O операции:

* Программа не выполняет операций чтения и записи.

3. Virtual Memory:

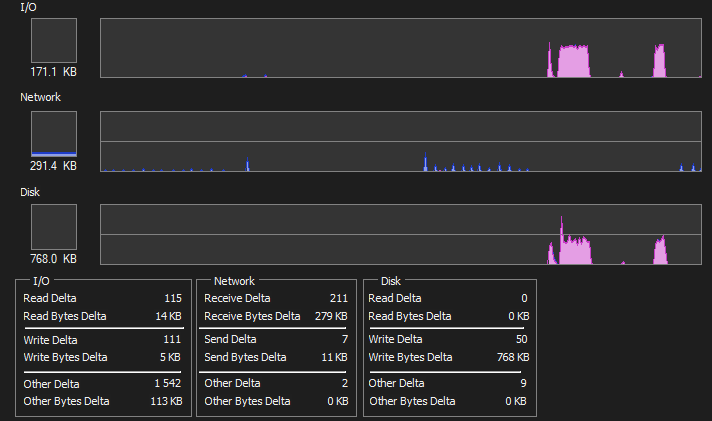
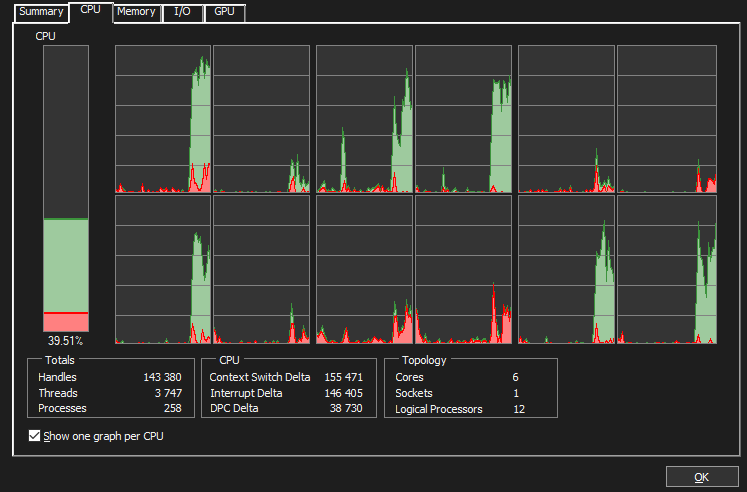
* Page Faults: Высокое количество страничных ошибок (526 316) может указывать на то, что процесс часто обращается к вновь выделенной странице в первый раз.
* Peak Private Bytes: Размер выделенной памяти в 789 844 KB (размер одного массива для дедупликация(512 000 KB) + сет для уникальных значений (?)).

## **Определение количества нагрузчиков, которое эффективно нагружает все ядра процессора**

Тест на AMD Ryzen 5 5600 (4.4 ГГЦ, 6\*2 логических процессоров) + 32GB RAM

ALL TEST

(1 \* 4+2 \* 4):



Write throughput: 12.7009 MB/s

Write throughput: 12.683 MB/s

Write throughput: 12.6486 MB/s

Write throughput: 12.626 MB/s

(benchmark1 1, 81.137666)

(benchmark2 1, 30.434909)

(benchmark1 2, 81.011103)

(benchmark2 2, 30.624453)

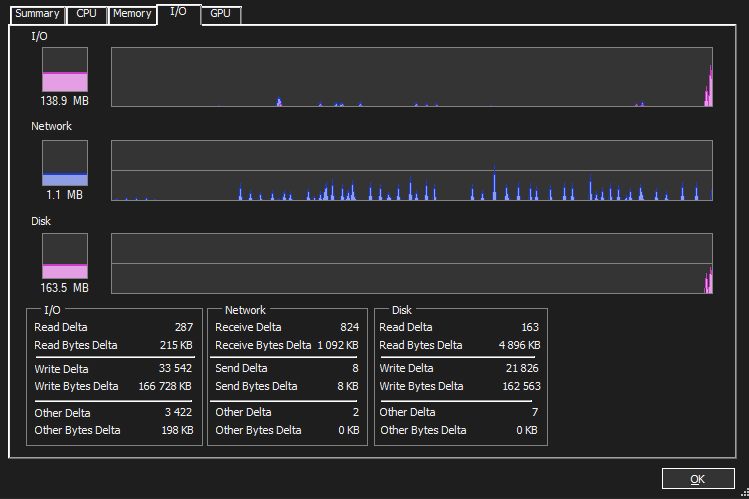
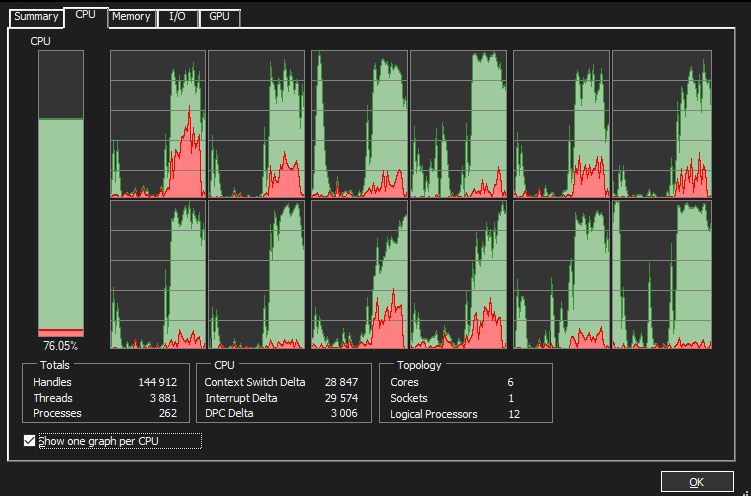
(benchmark1 3, 80.896353)

(benchmark2 3, 30.482917)

(benchmark1 4, 81.008101)

(benchmark2 4, 30.737004)

(1 \* 6+2 \* 6):



Write throughput: 4.26067 MB/s

Write throughput: 4.25887 MB/s

Write throughput: 4.25321 MB/s

Write throughput: 4.25268 MB/s

Write throughput: 4.25442 MB/s

Write throughput: 4.25224 MB/s

Write throughput: 4.25139 MB/s

Write throughput: 4.24422 MB/s

(benchmark1 1, 240.801413)

(benchmark2 1, 37.711218)

(benchmark1 2, 241.052995)

(benchmark2 2, 37.972342)

(benchmark1 3, 241.293359)

(benchmark2 3, 38.001344)

(benchmark1 4, 240.508494)

(benchmark2 4, 37.968343)

(benchmark1 5, 241.005868)

(benchmark2 5, 38.035345)

(benchmark1 6, 240.783396)

(benchmark2 6, 38.035345)

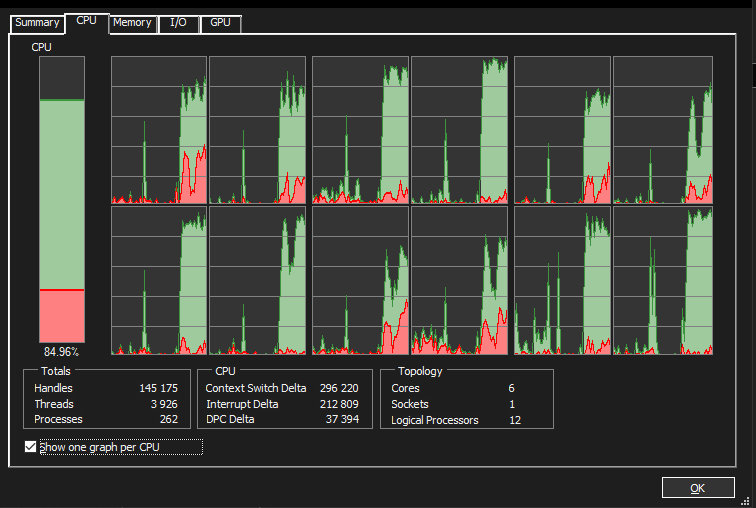
(benchmark1 7, 240.608964)

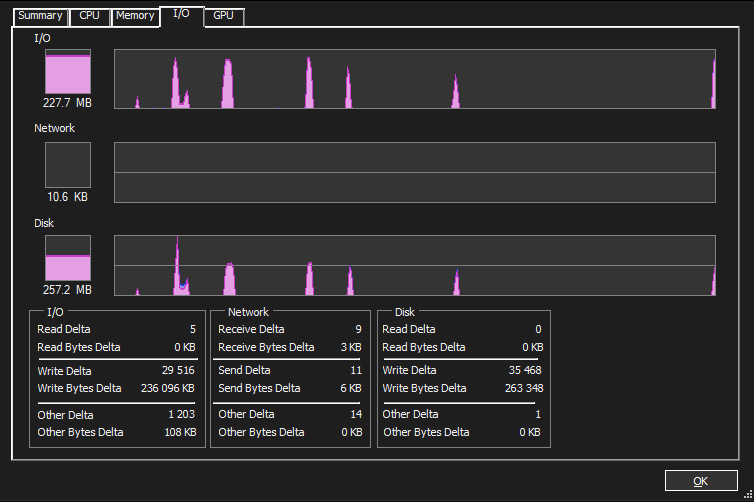
(benchmark2 7, 38.060905)

(benchmark1 8, 240.882166)

(benchmark2 8, 37.966343)

(1 \* 8 + 2 \* 8):





Write throughput: 4.62723 MB/s

Write throughput: 4.64361 MB/s

Write throughput: 4.66935 MB/s

Write throughput: 4.63479 MB/s

Write throughput: 4.66846 MB/s

Write throughput: 4.6665 MB/s

Write throughput: 4.66193 MB/s

Write throughput: 4.66566 MB/s

(benchmark1 1, 221.467758)

(benchmark2 1, 35.103577)

(benchmark1 2, 221.569782)

(benchmark2 2, 35.121574)

(benchmark1 3, 221.488618)

(benchmark2 3, 35.215576)

(benchmark1 4, 221.540312)

(benchmark2 4, 35.306575)

(benchmark1 5, 221.705564)

(benchmark2 5, 35.150575)

(benchmark1 6, 221.647285)

(benchmark2 6, 35.155577)

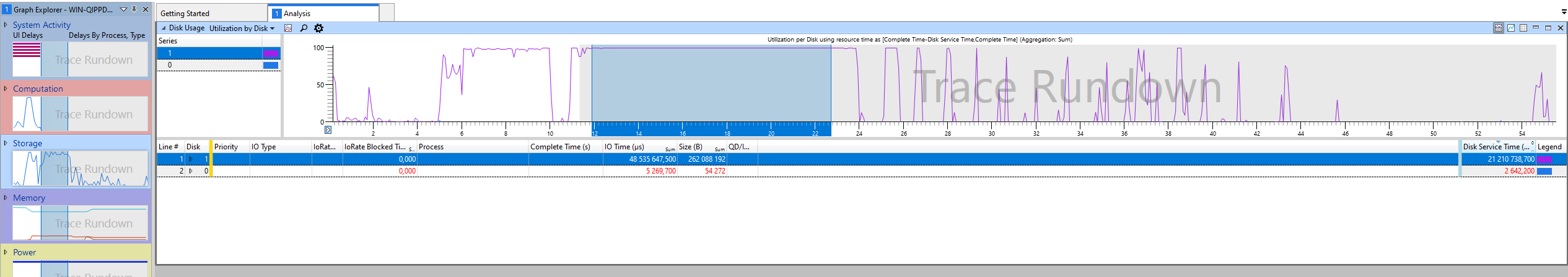
(benchmark1 7, 221.615153)

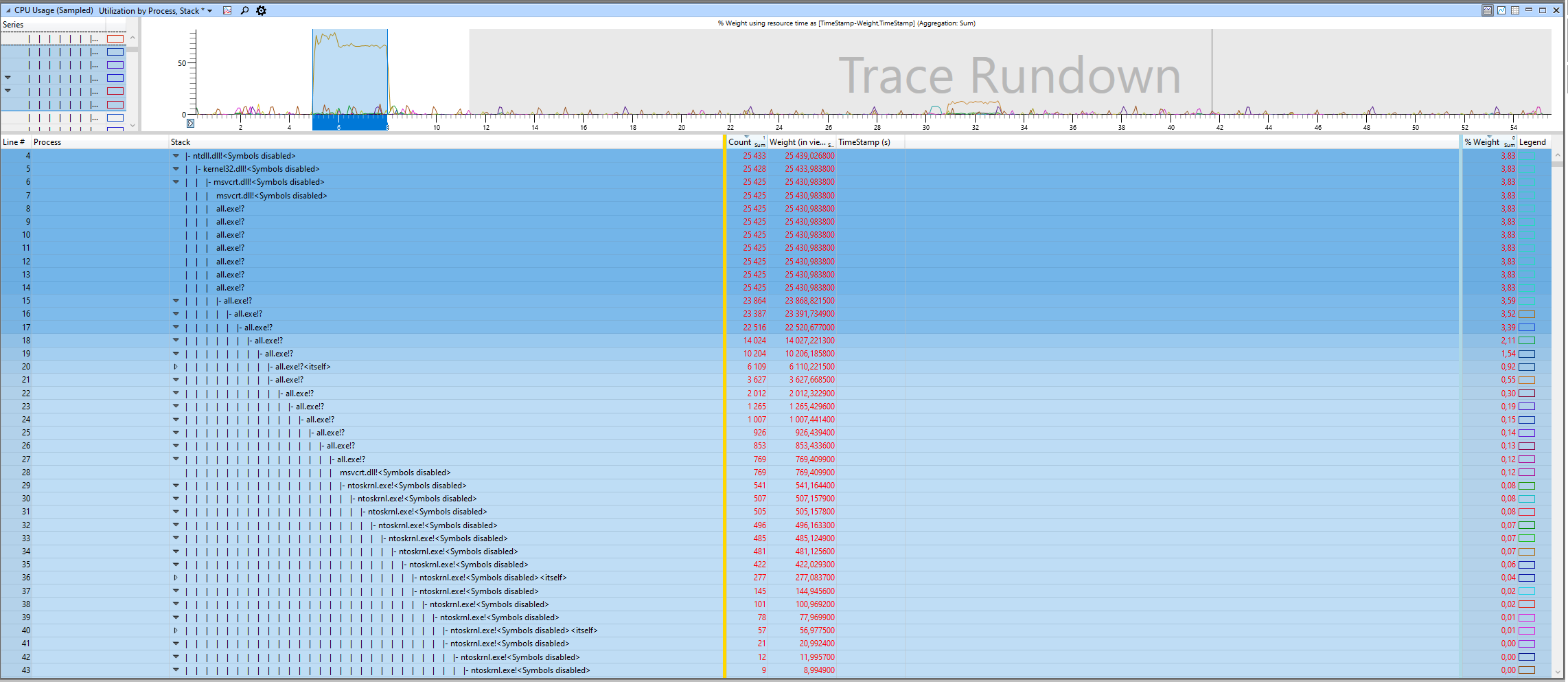
(benchmark2 7, 35.046037)

(benchmark1 8, 221.712748)

(benchmark2 8, 35.218575)

Трассировка







## **Сравнительный анализ ожидаемых и фактических значений**

В общем, наблюдаемые метрики соответствуют ожиданиям для программы, дедупликация элементов в массиве, и показывают характерные признаки работы с IO.

В целом, наблюдаемые метрики соответствуют ожиданиям для программы, выполняющей измерение пропускной способности на последовательную запись.

## **Вывод**

Во время выполнения лабораторной работы были изучены утилиты для мониторинга работы программы - Task Manager и Sysinternals, доступные в операционной системе Windows.