

CSE 344 – System Programming Homework 4 Report

Ahmet Özdemir

In the assignment, I will first describe my path of progress. The game consists of three main files, main.c systemHelper.c and stack.c, and two header files. Now I will explain this programme starting from the main function.

The general flow of the main function is as follows:

- 1 - argument checks
- 2 - buffer memory allocation
- 3 - time initialisation
- 4 - thread creation and thread end waiting blocks
- 5 - statistics information

```

1  /* main.c */
2
3  #include "systemHelper.h"
4  #include "stack.h"
5
6  void handle_sigint(int signal);
7  void setup_sigint_handler();
8  void cleanup_resources();
9
10 int main(int argc, char const * argv[])
11 {
12     int dummyControl; /* for checking syscall errors */
13
14     struct timeval startTime;
15     struct timeval endTime;
16
17     /***** argument control *****/
18
19     if (argc != 5)
20     {
21     }
22
23     /*****/
24
25     setup_sigint_handler();
26
27     bufferSize = toInteger(argv[1]);
28     uint32_t workersNumber = toInteger(argv[2]);
29
30     buffer = (RequestBody *)malloc(bufferSize * sizeof(RequestBody));
31
32     if (buffer == NULL)
33     {
34     }
35     bufferCount = 0;
36
37     DirPaths directories;
38     strncpy(directories.sourceDirPath, argv[3], NAME_SIZE);
39     strncpy(directories.destinDirPath, argv[4], NAME_SIZE);
40
41     pthread_t workers[workersNumber];
42     pthread_t manager;
43
44
45

```

```

63     /**** TIME START ****/
64     gettimeofday(&startTime, NULL);
65
66     dummyControl = pthread_create(&manager, NULL, managerTask, (void *)&directories);
67     errExitSyscall("Error on main(creating thread manager) function", dummyControl);
68
69     for (unsigned int i = 0; i < workersNumber; ++i)
70     {
71     }
72
73     dummyControl = pthread_join(manager, NULL);
74     errExitSyscall("Error on joining manager thread", dummyControl);
75
76     for (unsigned int i = 0; i < workersNumber; ++i)
77     {
78     }
79
80     gettimeofday(&endTime, NULL); // End timing
81     /**** TIME END ****/
82
83     long seconds = endTime.tv_sec - startTime.tv_sec;
84     long microseconds = endTime.tv_usec - startTime.tv_usec;
85     double elapsed = seconds + microseconds * 1e-6;
86
87     printf("----- STATISTICS -----\\n");
88     printf("Total files copied: %u\\n", filesCopied);
89     printf("Total bytes copied: %u\\n", totalBytesCopied);
90     printf("Number of regular files: %u\\n", numRegularFiles);
91     printf("Number of FIFOs: %u\\n", numFIFOFiles);
92     printf("Number of directories: %u\\n", numDirectories);
93     printf("Number of symbolic links: %u\\n", numSymbolicFiles);
94     printf("Total time elapsed: %.2f seconds\\n", elapsed);
95     printf("-----\\n");
96
97     cleanup_resources();
98
99     return EXIT_SUCCESS;
100 }
101
102 void setup_sigint_handler()
103 {
104 }
105
106 void handle_sigint(int signal)
107 {
108 }
109
110 void cleanup_resources()
111 {
112 }
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141

```

The manager thread is run first. Here paths are set and source path is opened, then copy files are created in destination path. Only necessary files are created, these files are opened with open on both sides, file descriptors and file paths are written to a struct object named fileBody. This object is also written to buffer. In these intervals, mutex locking operations are also performed where necessary.

```

49 void * managerTask(void * argument)
50 {
51     DirPaths * initialDirs = (DirPaths *)argument;
52     StackNode * stack = createStackNode(*initialDirs);
53
54     while (!isStackEmpty(stack) && !killSignal)
55     {
56         DirPaths currentDirs = pop(&stack);
57
58         const char * sourcePath = currentDirs.sourceDirPath;
59         const char * destinPath = currentDirs.destinDirPath;
60
61         DIR * sourceDir = opendir(sourcePath);
62
63         if (!sourceDir)
64         {
65             //
66         }
67         struct dirent * dEntry;
68
69         while ((dEntry = readdir(sourceDir)) != NULL)
70         {
71             if (strcmp(dEntry->d_name, ".") == 0 || strcmp(dEntry->d_name, "..") == 0)
72             {
73                 //
74             }
75             char sourceFileName[NAME_SIZE];
76             char destinFileName[NAME_SIZE];
77
78             snprintf(sourceFileName, NAME_SIZE, "%s/%s", sourcePath, dEntry->d_name);
79             snprintf(destinFileName, NAME_SIZE, "%s/%s", destinPath, dEntry->d_name);
80
81             if (strncmp(destinPath, sourceFileName, strlen(destinPath)) == 0)
82             {
83                 //
84             }
85
86             if (dEntry->d_type == DT_REG || dEntry->d_type == DT_FIFO || dEntry->d_type == DT_LNK)
87             {
88                 //
89             }
90
91             else if (dEntry->d_type == DT_DIR)
92             {
93                 //
94             }
95         }
96         closedir(sourceDir);
97     }
98     pthread_mutex_lock(&bufferMutex);
99     done = 1;
100     pthread_cond_broadcast(&bufferNotEmpty);
101     pthread_mutex_unlock(&bufferMutex);
102     clearStack(&stack);
103     pthread_exit(0);
104 }
105
106
107

```

If `dEntry->d_type == DT_DIR`, i.e. the found element is a directory, they are stored in a stack data structure. Sub-directories are then called sequentially with a recursive call logic.

Stack data-structures and necessary functions:

```

9
10  typedef struct StackNode
11  {
12      DirPaths dirPaths;
13      struct StackNode * next;
14  }
15  StackNode;
16
17  StackNode * createStackNode(DirPaths dirPaths);
18  DirPaths pop(StackNode ** stack);
19  void push(StackNode ** stack, DirPaths dirPaths);
20  void clearStack();
21  int isStackEmpty(StackNode * stack);
22

```

General flow of the Worker thread:

```

187
188  void * workerTask(void * argument)
189  {
190      while (TRUE)
191      {
192          pthread_mutex_lock(&bufferMutex);
193
194          while (bufferCount == 0 && !done && !killSignal)
195          {
196          }
197
198          if ((bufferCount == 0 && done) || killSignal)
199          {
200          }
201
202          RequestBody request = buffer[--bufferCount];
203
204          pthread_cond_signal(&bufferNotFull);
205          pthread_mutex_unlock(&bufferMutex);
206
207          if (request.FDSource == -1 || request.FDDestin == -1)
208          {
209          }
210
211          char bufferData[CHUNK_SIZE];
212          ssize_t bytesRead;
213
214          while ((bytesRead = read(request.FDSource, bufferData, CHUNK_SIZE)) > 0)
215          {
216          }
217          close(request.FDSource);
218          close(request.FDDestin);
219
220          pthread_mutex_lock(&bufferMutex);
221          filesCopied++;
222          pthread_mutex_unlock(&bufferMutex);
223
224          #ifdef SYS_DEBUG
225          #endif
226      }
227      pthread_exit(0);
228  }
229

```

Outputs of tests:

Test1:

```
ahmete@ahmete-Inspiron-14-5401: ~/DERSLER/3_SINIF/Spring/System-Programming/hw4test/hw4test/put_your_codes_here
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw4test/hw4test/put_your_codes_here$ make clean
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw4test/hw4test/put_your_codes_here$ make
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw4test/hw4test/put_your_codes_here$ valgrind ./MMcP 10 10 ../testdir/src/libvterm ../tcopy
py
==10701== Memcheck, a memory error detector
==10701== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==10701== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==10701== Command: ./MMcP 10 10 ../testdir/src/libvterm ../tcopy
==10701==
----- STATISTICS -----
Total files copied: 194
Total bytes copied: 25009680
Number of regular files: 194
Number of FIFOs: 0
Number of directories: 7
Number of symbolic links: 0
Total time elapsed: 0.53 seconds
-----
==10701==
==10701== HEAP SUMMARY:
==10701==   in use at exit: 0 bytes in 0 blocks
==10701==   total heap usage: 36 allocs, 36 frees, 344,518 bytes allocated
==10701==
==10701== All heap blocks were freed -- no leaks are possible
==10701==
==10701== For lists of detected and suppressed errors, rerun with: -s
==10701== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw4test/hw4test/put_your_codes_here$
```

Test2:

```
ahmete@ahmete-Inspiron-14-5401: ~/DERSLER/3_SINIF/Spring/System-Programming/hw4test/hw4test/put_your_codes_here
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw4test/hw4test/put_your_codes_here$ ./MMcP 10 4 ../testdir/src/libvterm/src ../tcopy
----- STATISTICS -----
Total files copied: 140
Total bytes copied: 24873682
Number of regular files: 140
Number of FIFOs: 0
Number of directories: 2
Number of symbolic links: 0
Total time elapsed: 0.06 seconds
-----
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw4test/hw4test/put_your_codes_here$
```

Test3:

```
ahmete@ahmete-Inspiron-14-5401: ~/DERSLER/3_SINIF/Spring/System-Programming/hw4test/hw4test/put_your_codes_here
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw4test/hw4test/put_your_codes_here$ ./MMcP 10 10 ../testdir ../tcopy
----- STATISTICS -----
Total files copied: 3116
Total bytes copied: 73520554
Number of regular files: 3116
Number of FIFOs: 0
Number of directories: 151
Number of symbolic links: 0
Total time elapsed: 0.07 seconds
-----
ahmete@ahmete-Inspiron-14-5401:~/DERSLER/3_SINIF/Spring/System-Programming/hw4test/hw4test/put_your_codes_here$
```