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 – adding pthread_barrier (* new)

6 - statistics information

```
62
63      pthread_mutex_init(&bufferMutex, NULL);
64      pthread_cond_init(&bufferNotEmpty, NULL);
65      pthread_cond_init(&bufferNotFull, NULL);
66      pthread_barrier_init(&barrier, NULL, workersNumbers);
67
68      /**** TIME START ****/
69      gettimeofday(&startTime, NULL);
70
71      dummyControl = pthread_create(&manager, NULL, managerTask, (void *)&directories);
72      errExitSyscall("Error on main(creating thread manager) function", dummyControl);
73
74      for (unsigned int i = 0; i < workersNumber; ++i)
75      { ...
78      }
79
80      dummyControl = pthread_join(manager, NULL);
81      errExitSyscall("Error on joining manager thread", dummyControl);
82
83      for (unsigned int i = 0; i < workersNumber; ++i)
84      { ...
87      }
88      gettimeofday(&endTime, NULL);
89
90      /***** TIME END *****/
91
92      long seconds = endTime.tv_sec - startTime.tv_sec;
93      long microseconds = endTime.tv_usec - startTime.tv_usec;
94      double elapsed = seconds + microseconds * 1e-6;
95
96      printf("------------------ STATISTICS ------------------\n");
97      printf("Total files copied: %u\n", filesCopied);
98      printf("Total bytes copied: %u\n", totalBytesCopied);
99      printf("Number of regular files: %u\n", numRegularFiles);
100     printf("Number of FIFOs: %u\n", numFIFOFiles);
101     printf("Number of directories: %u\n", numDirectories);
102     printf("Number of symbolic links: %u\n", numSymbolicFiles);
103     printf("Total time elapsed: %.2f seconds\n", elapsed);
104     printf("------------------------------------------------\n");
105
106     cleanup_resources();
107
108     pthread_mutex_destroy(&bufferMutex);
109     pthread_cond_destroy(&bufferNotEmpty);
110     pthread_cond_destroy(&bufferNotFull);
111     pthread_barrier_destroy(&barrier);
112
```

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          { ...
67          }
68          struct dirent * dEntry;
69
70          while ((dEntry = readdir(sourceDir)) != NULL)
71          {
72              if (strcmp(dEntry->d_name, ".") == 0 || strcmp(dEntry->d_name, "..") == 0)
73              { ...
75              }
76              char sourceFileName[NAME_SIZE];
77              char destinFileName[NAME_SIZE];
78
79              snprintf(sourceFileName, NAME_SIZE, "%s/%s", sourcePath, dEntry->d_name);
80              snprintf(destinFileName, NAME_SIZE, "%s/%s", destinPath, dEntry->d_name);
81
82              if (strncmp(destinPath, sourceFileName, strlen(destinPath)) == 0)
83              { ...
85              }
86
87              if (dEntry->d_type == DT_REG || dEntry->d_type == DT_FIFO || dEntry->d_type == DT_LNK)
88              { ...
153
154             }
155
156             else if (dEntry->d_type == DT_DIR)
157             { ...
176             }
177         }
178         closedir(sourceDir);
179     }
180     pthread_mutex_lock(&bufferMutex);
181     done = 1;
182     pthread_cond_broadcast(&bufferNotEmpty);
183     pthread_mutex_unlock(&bufferMutex);
184     clearStack(&stack);
185     pthread_exit(0);
186 }
187
```

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:

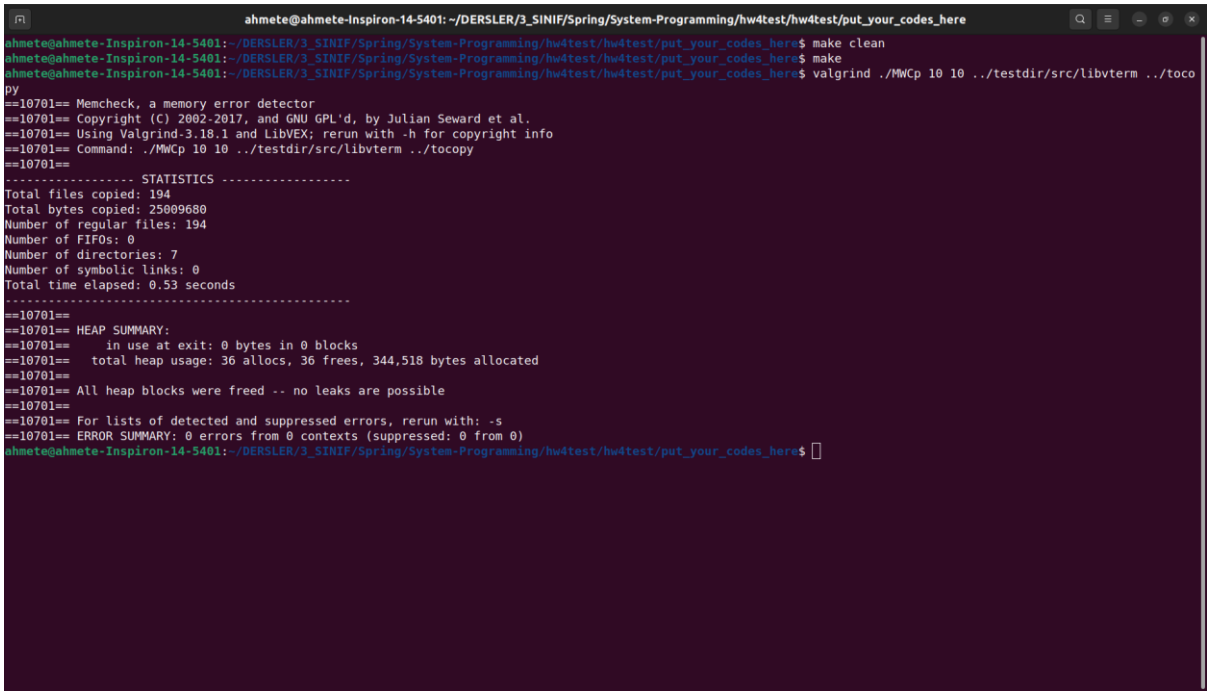
```
 9
10    typedef struct StackNode
11    {
12        DirPaths dirPaths;
13        struct StackNode * next;
14    }
15    StackNode;
16
17    StackNode * createStackNode(DirPaths dirPaths);
18    DirPaths pop(StackNode ** stack);
19    void push(StackNode ** stack, DirPaths dirPaths);
20    void clearStack();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

General flow of the Worker thread:

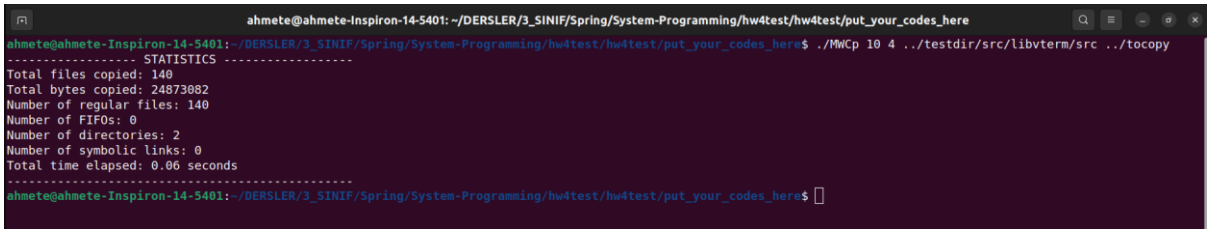
```
202
203   void * workerTask(void * argument)
204   {
205       pthread_barrier_wait(&barrier);
206
207       while (TRUE)
208       { ...
278
279       }
280       pthread_exit(0);
281   }
282
283
```

Outputs of tests:

Test1:



Test2:



Test3: