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

```c
/* main.c */

#include "systemHelper.h"
#include "stack.h"

void handle_sigint(int signal);
void setup_sigint_handler();
void cleanup_resources();

int main(int argc, char const * argv[])
{
    int dummyControl; /* for checking syscall errors */

    struct timeval startTime;
    struct timeval endTime;

    /***************************** argument control *****************************/

    if (argc != 5)
    { ...
    }

    /****************************************************************************/

    setup_sigint_handler();

    bufferSize = toInteger(argv[1]);
    uint32_t workersNumber = toInteger(argv[2]);

    buffer = (RequestBody *)malloc(bufferSize * sizeof(RequestBody));

    if (buffer == NULL)
    { ...
    }
    bufferCount = 0;

    DirPaths directories;
    strncpy(directories.sourceDirPath, argv[3], NAME_SIZE);
    strncpy(directories.destinDirPath, argv[4], NAME_SIZE);

    pthread_t workers[workersNumber];
    pthread_t manager;

```

```
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      { ...
73      }
74
75      dummyControl = pthread_join(manager, NULL);
76      errExitSyscall("Error on joining manager thread", dummyControl);
77
78      for (unsigned int i = 0; i < workersNumber; ++i)
79      { ...
82      }
83      gettimeofday(&endTime, NULL); // End timing
84      /***** TIME END *****/
85
86      long seconds = endTime.tv_sec - startTime.tv_sec;
87      long microseconds = endTime.tv_usec - startTime.tv_usec;
88      double elapsed = seconds + microseconds * 1e-6;
89
90      printf("------------------ STATISTICS ------------------\n");
91      printf("Total files copied: %u\n", filesCopied);
92      printf("Total bytes copied: %u\n", totalBytesCopied);
93      printf("Number of regular files: %u\n", numRegularFiles);
94      printf("Number of FIFOs: %u\n", numFIFOFiles);
95      printf("Number of directories: %u\n", numDirectories);
96      printf("Number of symbolic links: %u\n", numSymbolicFiles);
97      printf("Total time elapsed: %.2f seconds\n", elapsed);
98      printf("------------------------------------------------\n");
99
100     cleanup_resources();
101
102     return EXIT_SUCCESS;
103 }
104
105 void setup_sigint_handler()
106 { ...
117 }
118
119 void handle_sigint(int signal)
120 { ...
131 }
132
133 void cleanup_resources()
134 { ...
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          { ...
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:

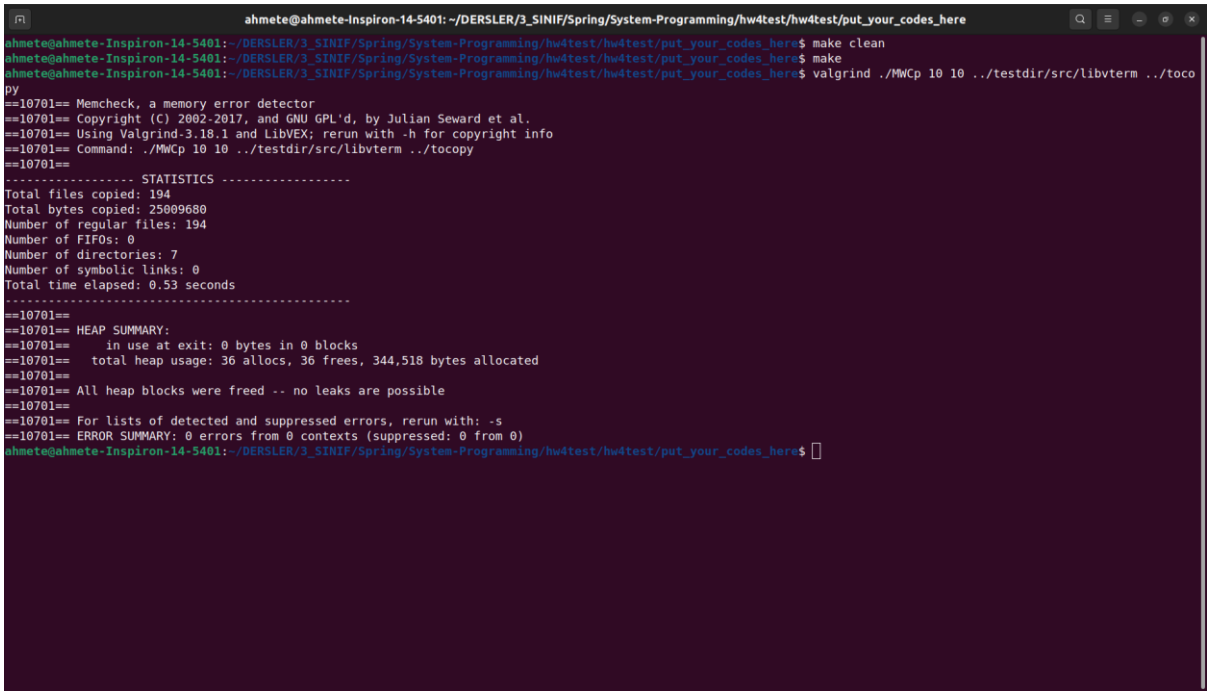
```c

typedef struct StackNode
{
    DirPaths dirPaths;
    struct StackNode * next;
}
StackNode;

StackNode * createStackNode(DirPaths dirPaths);
DirPaths pop(StackNode ** stack);
void push(StackNode ** stack, DirPaths dirPaths);
void clearStack();
int isStackEmpty(StackNode * stack);

```

General flow of the Worker thread:

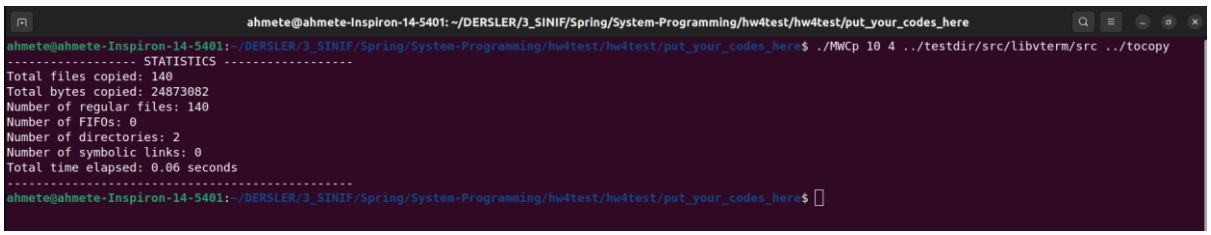
```c
void * workerTask(void * argument)
{
    while (TRUE)
    {
        pthread_mutex_lock(&bufferMutex);

        while (bufferCount == 0 && !done && !killSignal)
        { ...
        }

        if ((bufferCount == 0 && done) || killSignal)
        { ...
        }

        RequestBody request = buffer[--bufferCount];

        pthread_cond_signal(&bufferNotFull);
        pthread_mutex_unlock(&bufferMutex);

        if (request.FDSource == -1 || request.FDDestin == -1)
        { ...
        }

        char bufferData[CHUNK_SIZE];
        ssize_t bytesRead;

        while ((bytesRead = read(request.FDSource, bufferData, CHUNK_SIZE)) > 0)
        { ...
        }
        close(request.FDSource);
        close(request.FDDestin);

        pthread_mutex_lock(&bufferMutex);
        filesCopied++;
        pthread_mutex_unlock(&bufferMutex);

#ifdef SYS_DEBUG ...
#endif

    }
    pthread_exit(0);
}

```

Outputs of tests:

Test1:



Test2:



Test3: