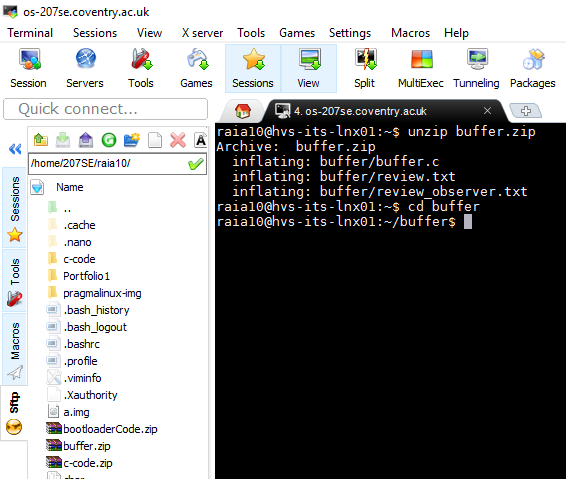
Operating Systems, Security and Networks (207SE) Lab 7: **Simple buffer exercise**

This week we will look at a simple buffer approach to copy a file from one location to another using a buffer. Like when you are using youtube to watch videos.

# Setting up the activities

**Getting to required file**

* Run MobaXterm and connect to the server
* Get buffer.zip from moodle
* Drag buffer.zip into MobaXterm or using filezilla
* unzip buffer.zip  cd buffer



**Background Knowledge**

* See lecture slides at end for a description on write/read and what the code does.

# Tasks - File copying using a buffer

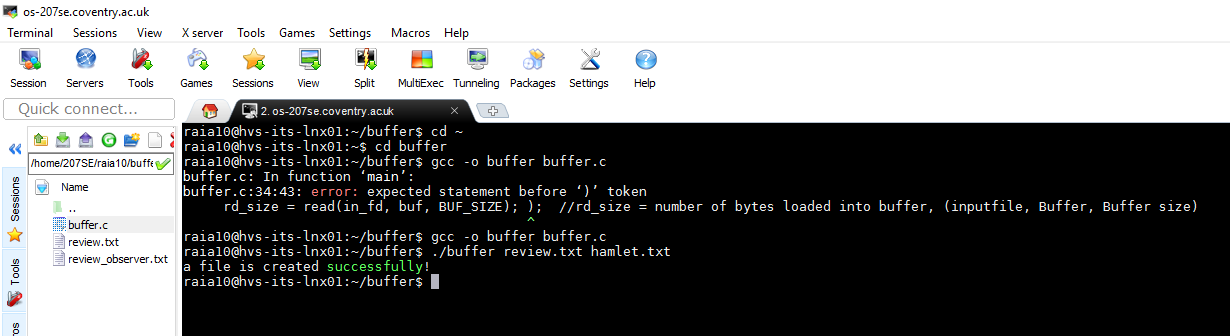
1. Comment the code below to indicate what different parts do.

1. #include <fcntl.h>          // defining the h files
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <stdio.h>
6. #define BUF\_SIZE 500           //this sets the biffer size to 500
7. #define OUTPUT\_MODE 0700       // this defines the output and sets file permissions
9. **int** main(**int** argc, **char** \*argv[]) //this line starts main function
10. {
11. **int** in\_fd, out\_fd;              // these are defining all the variables
12. **int** rd\_size = 1, wr\_size;
13. **char** buf[BUF\_SIZE];        //declaring Buffer
15. **if** (argc != 3)  //this will check if input entered is within parameters bound
16. exit(1);
18. in\_fd = open(argv[1], O\_RDONLY); // this will open review.txt
19. **if** (in\_fd < 0)
20. exit(2);
22. out\_fd = creat(argv[2], OUTPUT\_MODE); // creates argo.txt
23. **if** (out\_fd < 0)
24. exit(3);
26. **while** (rd\_size > 0) { //while loop to repeatedly read fom the original file into buffer
28. rd\_size = read(in\_fd, buf, BUF\_SIZE);  //rd\_size = number of bytes loaded into buffer, (inputfile, Buffer, Buffer size)
29. **if** (rd\_size <0)
30. exit(4);
32. wr\_size = write(out\_fd, buf, rd\_size); //Loads a section of the file into the buffer
33. **if** (wr\_size<=0){
34. close(in\_fd);   //close files
35. close(out\_fd);
36. exit(5);
37. }
38. }
39. }

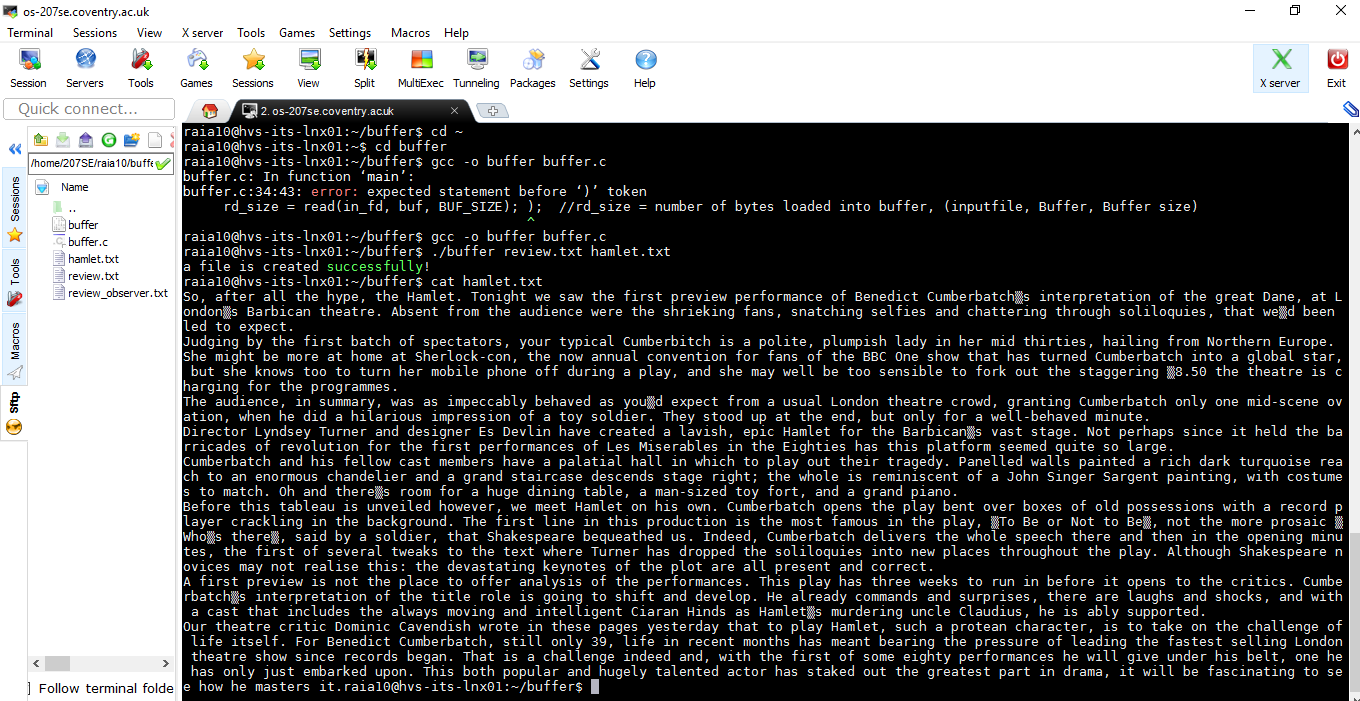
2 . Update the code so it prints the error that has occurred or if a file has been successfully created and a copy of the review included in it.

1. #include <fcntl.h>      // defining the h files
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <stdio.h>
6. #define BUF\_SIZE 500     //this sets the biffer size to 500
7. #define OUTPUT\_MODE 0700      // this defines the output and sets file permissions
9. **int** main(**int** argc, **char** \*argv[])   //this line starts main function
10. {
11. **int** in\_fd, out\_fd;           // these are defining all the variables
12. **int** rd\_size = 1, wr\_size;
13. **char** buf[BUF\_SIZE];         //declaring Buffer
15. **if** (argc != 3) {       //this will check if input entered is within parameters bound
16. printf("Input must be within parameter = 3.\n");  // this will print an Error message
17. exit(1);
18. }
20. in\_fd = open(argv[1], O\_RDONLY); // this will open review.txt
21. **if** (in\_fd < 0) {
22. printf("Given file is empty!\n");  // this will print an Error message
23. exit(2);
24. }
26. out\_fd = creat(argv[2], OUTPUT\_MODE); // creates argo.txt
27. **if** (out\_fd < 0) {
28. printf("Unable to write in the given file\n");  // this will print an Error message
29. exit(3);
30. }
32. **while** (rd\_size > 0) {
34. rd\_size = read(in\_fd, buf, BUF\_SIZE);   //rd\_size = number of bytes loaded into buffer, (inputfile, Buffer, Buffer size)
35. **if** (rd\_size <0) {
36. printf("There is nothing in the file to be read.\n");  // this will print an Error message
37. exit(4);
38. }
39. wr\_size = write(out\_fd, buf, rd\_size); //Loads a section of the file into the buffer
40. **if** (wr\_size<=0){
41. printf("a file is created successfully!\n");  //prints that the review.txt has been created successfully
42. close(in\_fd); //close files
43. close(out\_fd); //close files
44. exit(5);
45. }
46. }
47. }

Compile the code using **gcc –o buffer buffer.c** and run it by typing **./buffer review.txt hamlet.txt.**

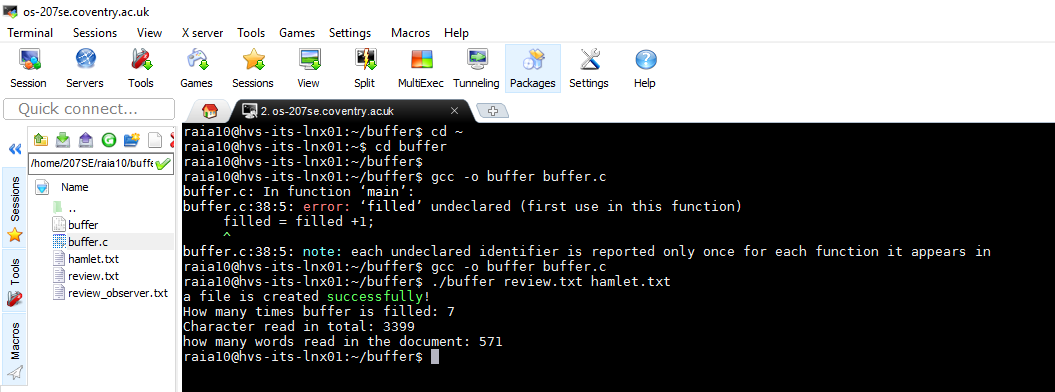


What is in hamlet.txt?



**Hint review.txt is a review of the latest production of Hamlet**

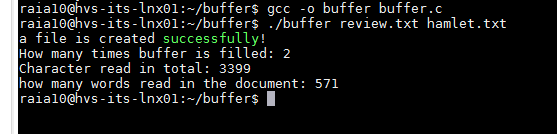
1. Adapt the code to show how many characters were read in total, how many characters are read from the buffer at a time, how many words are in the document, and how many times the buffer is filled.



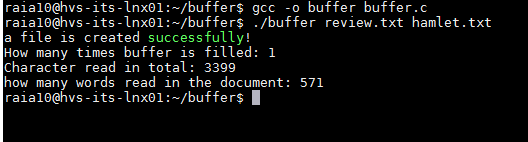
**Updated code**

1. #include <fcntl.h>      // defining the h files
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <stdio.h>
6. #define BUF\_SIZE 500     //this sets the biffer size to 500
7. #define OUTPUT\_MODE 0700      // this defines the output and sets file permissions
9. **int** main(**int** argc, **char** \*argv[])   //this line starts main function
10. {
11. **int** in\_fd, out\_fd;           // these are defining all the variables
12. **int** rd\_size = 1, wr\_size;
13. **char** buf[BUF\_SIZE];    //declaring Buffer
14. **int** characters = 0;
15. **int** filled = -1;
16. **int** word = 0;
17. **int** counter = 0;
19. **if** (argc != 3) {       //this will check if input entered is within parameters bound
20. printf("Input must be within parameter = 3.\n");  // this will print an Error message
21. exit(1);
22. }
24. in\_fd = open(argv[1], O\_RDONLY); // this will open review.txt
25. **if** (in\_fd < 0) {
26. printf("Given file is empty!\n");  // this will print an Error message
27. exit(2);
28. }
30. out\_fd = creat(argv[2], OUTPUT\_MODE); // creates argo.txt
31. **if** (out\_fd < 0) {
32. printf("Unable to write in the given file\n");  // this will print an Error message
33. exit(3);
34. }
36. **while** (rd\_size > 0) {
38. filled = filled +1;
39. rd\_size = read(in\_fd, buf, BUF\_SIZE);   //rd\_size = number of bytes loaded into buffer, (inputfile, Buffer, Buffer size)
40. characters = rd\_size +characters;
41. **for** (counter = 0; counter < rd\_size; counter ++) {
42. **if** (buf[counter] == ' '){
43. word = word+1;
44. }
45. }
46. **if** (rd\_size <0) {
47. exit(4);
48. }
49. wr\_size = write(out\_fd, buf, rd\_size); //Loads a section of the file into the buffer
50. **if** (wr\_size<=0){
51. printf("a file is created successfully!\n");  //prints that the review.txt has been created successfully
52. printf("How many times buffer is filled: %d\n", filled);
53. printf("Character read in total: %d\n", characters);
54. printf("how many words read in the document: %d\n", word);
55. close(in\_fd); //close files
56. close(out\_fd); //close files
57. exit(5);
58. }
59. }
60. }
61. Alter BUF\_SIZE to 2000. How does this influence the number of times the buffer is filled. Try different values for BUF\_SIZE.

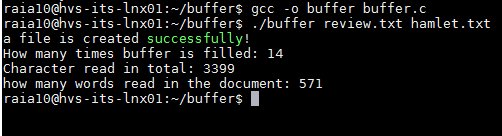
BUF\_SIZE to 2000.



BUF\_SIZE = 4000



BUF\_SIZE = 250



Conclusion – High number of buffer size influences the number of times buffer is filled to be lower and lower, however lower buffer size causes increase in number of the times buffer is filled.

1. Adapt the code in buffer.c so it is possible to compare if two files are the same. If they are different say how they differ. Explain the approach for comparing the files and say why you feel it is an appropriate for comparing the files.

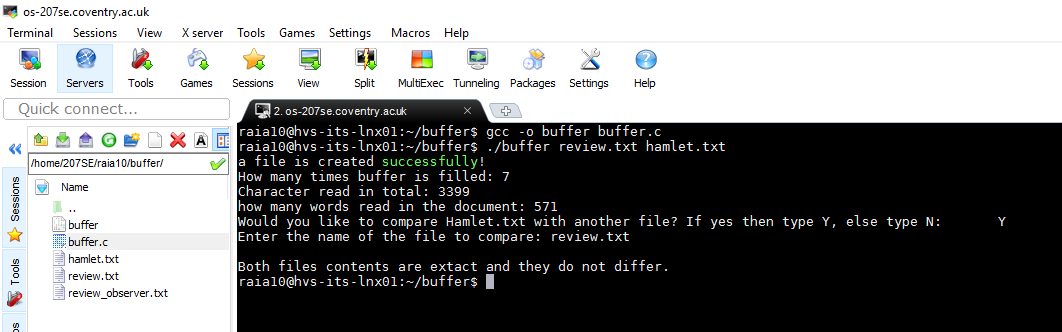
The approach here is simple. The main function will create a new .txt file called hamlet.txt containing a review of latest production of Hamlet and compare function will compare hamlet.txt file against a file name given by the user through input. I feel like it is an appropriate approach for comparing files because it can be coded to compare files and display differences through all sorts such as colour coded, lines, sentences and paragraphs etc. because command such as diff is limited to displaying everything at once and it can be messy to see.

1. #include <fcntl.h>      // defining the h files
2. #include <stdlib.h>
3. #include <unistd.h>
4. #include <stdio.h>
6. #define BUF\_SIZE 500     //this sets the biffer size to 500
7. #define OUTPUT\_MODE 0700      // this defines the output and sets file permissions
9. **int** compare(**char** \*comFile)  //Compare function for comparing 2 files
10. {
11. **int** infd1, infd2;     //Defining variables for compare function
12. **char** compareFile[50];
13. **int** rdSize1 = 1, rdSize2 = 1;
14. **char** buffer1[BUF\_SIZE];
15. **char** buffer2[BUF\_SIZE];
16. **int** i, x=0;
17. **char** p;
19. printf("Enter the name of the file to compare: ");        //ask for input(filename) to be compared
20. scanf("%s", compareFile);                 //read input
21. infd1 = open(compareFile, O\_RDONLY);              //opens the file inputted
22. infd2 = open(comFile, O\_RDONLY);              //opens the file created from Main function(hamlet.txt)
24. **if** (infd1 < 0)
25. exit(printf("The given file is empty!\n"));     //Exits if there is nothing in given file
27. **while** (rdSize1 > 0) {              //checks if there are characters in the text file
29. rdSize1 = read(infd1, buffer1, BUF\_SIZE);       //changes varible to bytes and gives buffer1 variable the buffsize definined at the start
30. rdSize2 = read(infd2, buffer2, BUF\_SIZE);       //changes varible to bytes and gives buffer2 variable the buffsize definined at the start
32. **if** (rdSize1 <0)
33. exit(printf("There is nothing in the file to be read\n")); //if statments to check if there is something to be read in the file, if not then give error message
34. **if** (rdSize2 <0)
35. exit(printf("There is nothing in the file to be read\n"));      //if statments to check if there is something to be read in the file, if not then give error message
37. **for** (i = 0;i<rdSize1;i++){       //loop in two buffers
38. p = buffer2[i];
39. **if** (buffer1[i] != p)    {       //compares characters in two buffers
40. **if** (x == 0)
41. printf("The two files different: \n");  //prints message
42. printf("%c", p);    //prints characters that are different in two files
43. x=1;            // sets boolean data to True
44. }
45. }

48. }
49. printf("\n");         //just starts a new line
50. **if** (x == 0)           // checks if both files are exactly the same
51. printf("Both files contents are extact and they do not differ.\n");     //prints message
52. }
54. **int** main(**int** argc, **char** \*argv[])   //this line starts main function
55. {
56. **int** in\_fd, out\_fd;           // these are defining all the variables
57. **int** rd\_size = 1, wr\_size;
58. **char** buf[BUF\_SIZE];    //declaring Buffer
59. **int** characters = 0;
60. **int** filled = -1;
61. **int** word = 0;
62. **int** counter = 0;
64. **if** (argc != 3) {       //this will check if input entered is within parameters bound
65. printf("Input must be within parameter = 3.\n");  // this will print an Error message
66. exit(1);
67. }
69. in\_fd = open(argv[1], O\_RDONLY); // this will open review.txt
70. **if** (in\_fd < 0) {
71. printf("Given file is empty!\n");  // this will print an Error message
72. exit(2);
73. }
75. out\_fd = creat(argv[2], OUTPUT\_MODE); // creates argo.txt
76. **if** (out\_fd < 0) {
77. printf("Unable to write in the given file\n");  // this will print an Error message
78. exit(3);
79. }
81. **while** (rd\_size > 0) {
83. filled = filled +1;
84. rd\_size = read(in\_fd, buf, BUF\_SIZE);   //rd\_size = number of bytes loaded into buffer, (inputfile, Buffer, Buffer size)
85. characters = rd\_size +characters;
86. **for** (counter = 0; counter < rd\_size; counter ++) {
87. **if** (buf[counter] == ' '){
88. word = word+1;
89. }
90. }
91. **if** (rd\_size <0) {
92. exit(4);
93. }
94. wr\_size = write(out\_fd, buf, rd\_size); //Loads a section of the file into the buffer
95. **if** (wr\_size<=0){
96. printf("a file is created successfully!\n");  //prints that the review.txt has been created successfully
97. printf("How many times buffer is filled: %d\n", filled);
98. printf("Character read in total: %d\n", characters);
99. printf("how many words read in the document: %d\n", word);
100. printf("Would you like to compare Hamlet.txt with another file? If yes then type Y, else type N:   ");
101. scanf("%s", buf);
102. **if** (buf[0] == 'Y')
103. compare(argv[2]);
104. close(in\_fd); //close files
105. close(out\_fd); //close files

108. exit(5);
109. }
110. }
111. }

Compare **review.txt** and **hamlet.txt.**



Compare **hamlet.txt** and **review\_observer.txt.**

