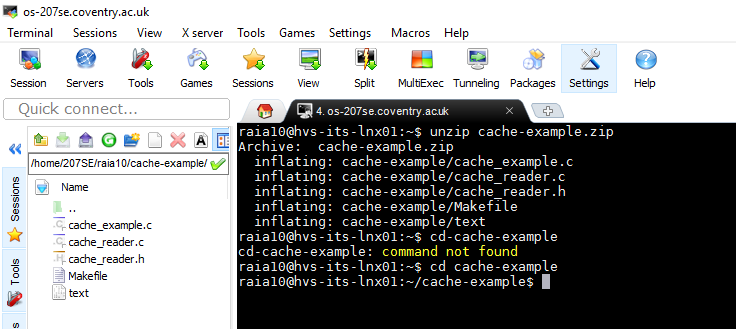
Operating Systems, Security and Networks (207SE) Lab 8: **Cache Library exercise**

This session we will explore an application that is a more elegant version of the last tutorial. It does this by using a structure that simulates a cache library.

# Setting up the activities

## Getting to required file

* Run MobaXterm and connect to the server
* Drag the zip file into MobaXterm or use filezilla to ftp the zip file to the server
* unzip cache-example.zip
* cd cache-example

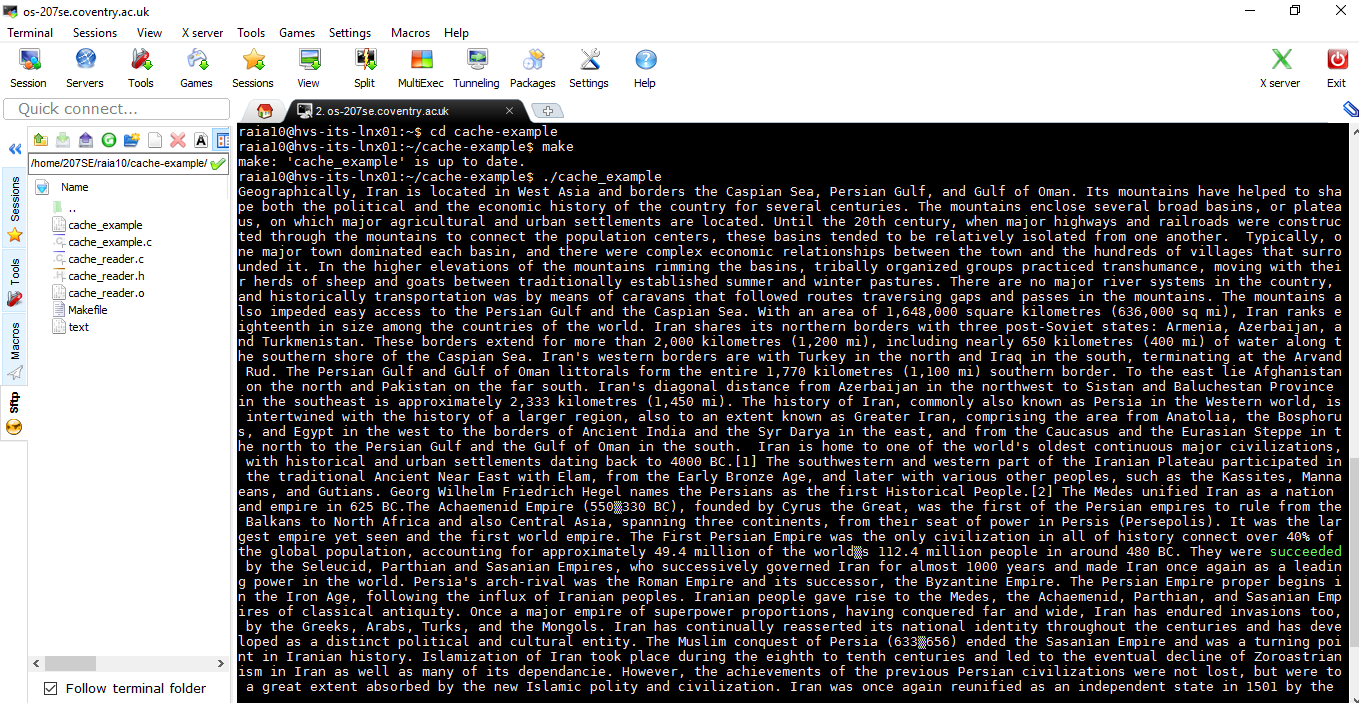


# Basic Portfolio Activity

**The actual code required to complete the cr\_read\_byte function will be simple.**

1. **char** cr\_read\_byte(cr\_file\* f)
3. {
4. **if** (f->usedbuffer == f->bufferlength) // If statement to check if the buffer is empty
5. {
6. refill(f); // if the buffer is empty then refill the buffer
7. };
9. **char** temp = f->buffer[f->usedbuffer]; // this will access char on index [usedbuffer]
10. f->usedbuffer++; // incrementing index usedbuffer to access next char in index
12. **return** (temp); // returns the characters from until End Of File is reached
13. **return** EOF; // this will make the compile work
15. }
16. //your code goes here
17. //remember that this needs a return char (a byte, put another way...)

**The correct output from the program should be the contents of the ‘text’ file!**



# Additional Portfolio Activity

1. How can you prove the file output is being buffered, given the output to the screen is essentially the contents of the file. Devise some way to show that **each byte is being read,** and **when the buffer is being refilled.** This may require (nominal) changes to the main program and the library code.

**How I proved**

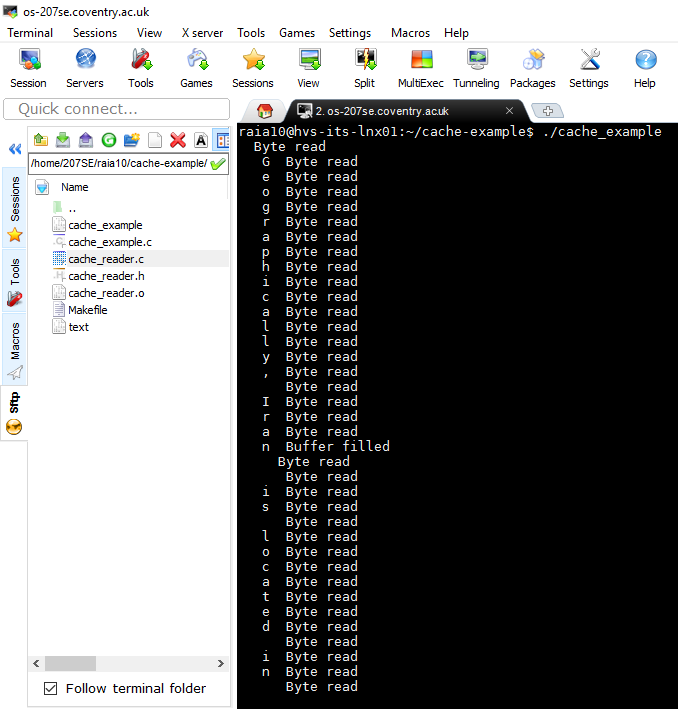
To prove that the file output is being buffered, I added a new code, ‘printf(“Buffer filled\n”)’;, this will be printed every time a new buffer is being filled.

To prove that each byte is being read, I added a new code, ‘printf(“ byte read/n)’, this will be printed every time a character is read.

**Updated code**

1. **char** cr\_read\_byte(cr\_file\* f)
3. {
4. **if** (f->usedbuffer == f->bufferlength) // If statement to check if the buffer is empty
5. {
6. printf("  Buffer filled\n   "); //prints the message "buffer is being filled" in a new line whenever a buffer is filled //advance task
7. refill(f); // if the buffer is empty then refill the buffer
8. };
10. printf("  Byte read\n   "); //this will make sure that message is printed everytime a buffer stores a character //advanced task
12. **char** temp = f->buffer[f->usedbuffer]; // this will access char on index [usedbuffer]
13. f->usedbuffer++; // incrementing index usedbuffer to access next char in index
15. **return** (temp); // returns the characters from until End Of File is reached
16. **return** EOF; // this will make the compile work
18. }
19. //your code goes here
20. //remember that this needs a return char (a byte, put another way...)

**Proof that the code works**



1. Provide some statistics in the cache\_example.c code to show how many bytes were read in total, how many words were read in and how many times the buffer was refilled.

**Updated Code**

1. #import "cache\_reader.h"
3. //updated code to show how many 'bytes were read in total', 'how many words were read in' and 'how many times the buffer was refilled'
5. **int** main(){
6. **char** c;
7. **int** bytecount = 0;
8. **int** buffercount=0;
9. **int** wordscount =0;
11. //Open a file
12. cr\_file\* f = cr\_open("text",20);
14. //While there are useful bytes coming from it
15. **while**((c=cr\_read\_byte(f)) !=EOF) {
16. //print them
17. bytecount = bytecount + 1; //count each character in the file
18. printf("%c", c);
20. //How many times buffer was filled = count with IF statment
21. **if** (f->usedbuffer==f->bufferlength)//If statement to check if the buffer is empty
22. buffercount=buffercount+1; //increments count which displays number of times buffer filled
24. //How many words were read = Word count with IF statement
25. **if** ((f->buffer[f->usedbuffer])==' ') //this will transverse within spaces of characters
26. wordscount = wordscount +1; //increments words if space is found between words
27. }
29. printf("\n"); //sets a new line before printing the results
30. printf("Total characters/bytes in the file: %d\n", bytecount);  //prints how many 'bytes were read in total', 'how many words were read in' and 'how many times the buffer was refilled'
31. printf("Total time buffer was refilled: %d\n", buffercount);    //prints how many times the buffer was refilled
32. printf("Total words read in the file: %d\n", wordscount);    //prints how many words were read in the file
33. cr\_close(f);
35. **return** 0;
37. }

**Proof that code works**

