**Operating Systems, security and Networks (207SE)**

**Lab 10: The Kernel Again**

# Your task

**For up to 3/5 Change the cache\_reader library** from using the **fopen, fread, fclose** functions to the system call versions **open, read, close**

**Code from cache\_reader.c**

1. #define \_GNU\_SOURCE //allowed access to other functions
2. #include "cache\_reader.h"
3. #include <sys/stat.h>
4. #include <fcntl.h>
5. #include <unistd.h>
6. #include <sys/types.h>
8. //http://www.phim.unibe.ch/comp\_doc/c\_manual/C/SYNTAX/struct.html
9. //http://vergil.chemistry.gatech.edu/resources/programming/c-tutorial/structs.html
11. **int** refill(cr\_file\* buff){
12. //Refills a buffer
13. //Only works when completely used buffer
14. **if**(buff->usedbuffer!=buff->bufferlength)
15. **return** 0;
16. **else**{
17. buff->usedbuffer=0;
18. **int** len=read(buff->file, buff->buffer, buff->bufferlength); //read function format
19. //If we didn't fill the buffer, fill up with EOF
20. **if**(len<buff->bufferlength)
21. **for**(**int** i=len;i<buff->bufferlength;i++)
22. buff->buffer[i]=EOF;  //Accessing like an array!
23. **return** len;
24. }
26. }
28. **void** cr\_close(cr\_file\* f){
30. free(f->buffer);
31. close(f->file);
32. }
34. cr\_file\* cr\_open(**char** \* filename, **int** buffersize){
35. **char** buff[buffersize=512]; //needed to declare the buffer and its size
36. //Info on malloc
37. //http://www.space.unibe.ch/comp\_doc/c\_manual/C/FUNCTIONS/malloc.html
38. **FILE**\* f;
39. **if** ((f = open(filename, O\_RDONLY|O\_DIRECT)) ==-1 ) { //opened for reading only, format for open O\_DIRECT is used to reduce the caching from/for a file
40. fprintf(stderr, "Cannot open %s\n", filename);
41. **return** 0;
42. }
44. cr\_file\* a=(cr\_file\*)malloc(**sizeof**(cr\_file));
45. a->file=f;
46. a->bufferlength=buffersize;
47. a->usedbuffer=buffersize; //Start off with no characters, so refill will work as expected
48. a->buffer=(**char**\*)memalign(**sizeof**(**char**)\*buffersize, **sizeof**(**char**)\*buffersize); //needed to change mem type so used memalign instead of malloc.. this meant changing the struct a little bit due to syntax
50. refill(a);
51. **return** a;
52. }
54. //------------------------------------------------------------------
55. **char** cr\_read\_byte(cr\_file\* f){
56. printf("   "); //prints a space before each byte is read
57. **if** (f->usedbuffer==f->bufferlength){ //the character stored in the pointer of f is accessed by usedbuffer, its then checked to see whether the buffer is full
58. printf(" REFILL "); //prints refill when the buffer is full
59. refill(f); //if buffer isnt full then it will continue filling
60. }
61. **return** f->buffer[f->usedbuffer++]; //incrementing usedbuffer by 1
63. **return** EOF; // this is just so the compile works...
64. }

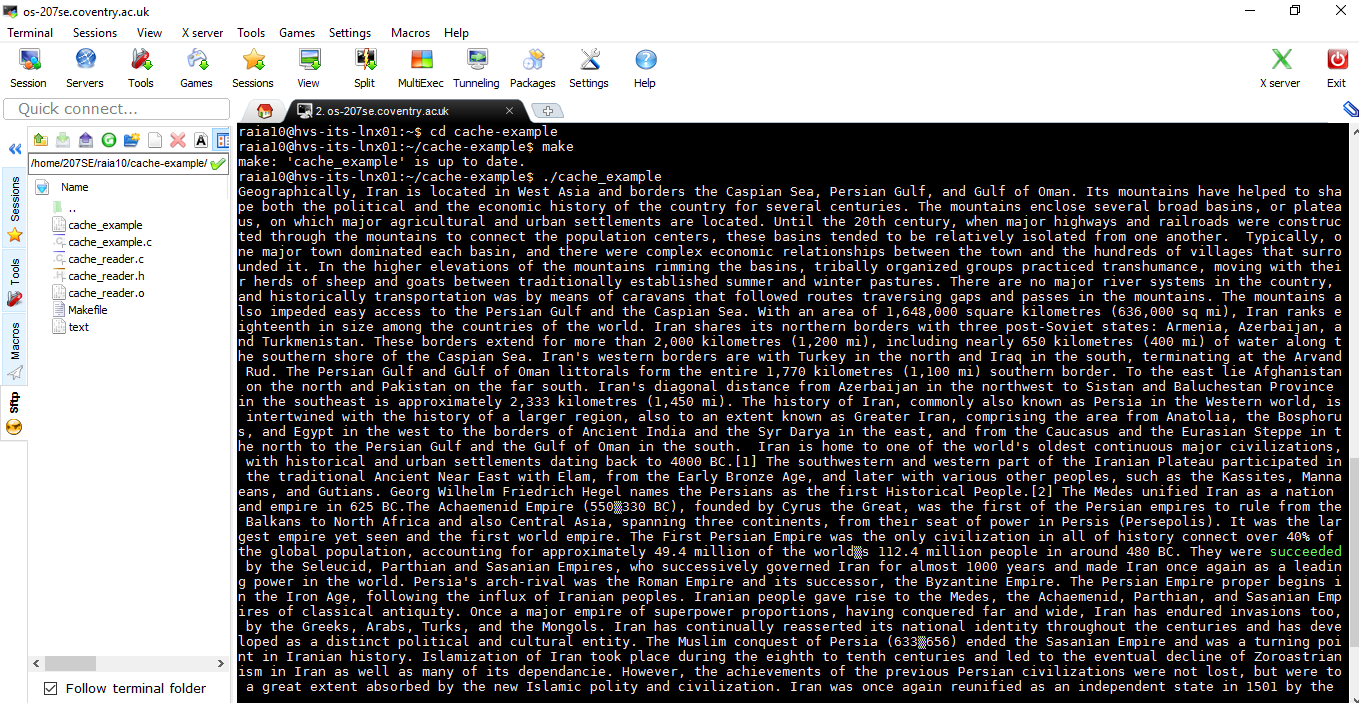
**Code from cache\_reader.h**

1. #include <stdio.h>
2. #include <stdlib.h>

5. //The internals of this struct aren't important
6. //from the user's point of view
7. **typedef** **struct**{
8. **int** file;        //File being read, reads as ASCII
9. **int** bufferlength;  //Fixed buffer length
10. **int** usedbuffer;    //Current point in the buffer
11. **char**\* buffer;      //A pointer to a piece of memory
12. //  same length as "bufferlength"
13. } cr\_file;
15. //Open a file with a given size of buffer to cache with
16. cr\_file\* cr\_open(**char**\* filename, **int** buffersize);

19. //Close an open file
20. **void** cr\_close(cr\_file\* f);
22. //Read a byte.  Will return EOF if empty.
23. **char** cr\_read\_byte(cr\_file\* f);
25. //---------------------------------------------------------
27. //Refill an empty buffer.  Not intended for users
28. **int** refill(cr\_file\* buff);

**Functions to the system call versions open, read, close gives following output**



**For up to 5/5** Remove (as far as possible) the effects of caching on the library. The slides have a big hint, but you **will** need the man pages and google for this.

1. The code commented appropriately

**Code from cache\_reader.c**

1. #define \_GNU\_SOURCE //allowed access to other functions
2. #include "cache\_reader.h"
3. #include <sys/stat.h>
4. #include <fcntl.h>
5. #include <unistd.h>
6. #include <sys/types.h>
8. //http://www.phim.unibe.ch/comp\_doc/c\_manual/C/SYNTAX/struct.html
9. //http://vergil.chemistry.gatech.edu/resources/programming/c-tutorial/structs.html
11. **int** refill(cr\_file\* buff){
12. //Refills a buffer
13. //Only works when completely used buffer
14. **if**(buff->usedbuffer!=buff->bufferlength)
15. **return** 0;
16. **else**{
17. buff->usedbuffer=0;
18. **int** len=read(buff->file, buff->buffer, buff->bufferlength); //read function format
19. //If we didn't fill the buffer, fill up with EOF
20. **if**(len<buff->bufferlength)
21. **for**(**int** i=len;i<buff->bufferlength;i++)
22. buff->buffer[i]=EOF;  //Accessing like an array!
23. **return** len;
24. }
26. }
28. **void** cr\_close(cr\_file\* f){
30. free(f->buffer);
31. close(f->file);
32. }
34. cr\_file\* cr\_open(**char** \* filename, **int** buffersize){
35. **char** buff[buffersize=512]; //needed to declare the buffer and its size
36. //Info on malloc
37. //http://www.space.unibe.ch/comp\_doc/c\_manual/C/FUNCTIONS/malloc.html
38. **FILE**\* f;
39. **if** ((f = open(filename, O\_RDONLY|O\_DIRECT)) ==-1 ) { //opened for reading only, format for open O\_DIRECT is used to reduce the caching from/for a file
40. fprintf(stderr, "Cannot open %s\n", filename);
41. **return** 0;
42. }
44. cr\_file\* a=(cr\_file\*)malloc(**sizeof**(cr\_file));
45. a->file=f;
46. a->bufferlength=buffersize;
47. a->usedbuffer=buffersize; //Start off with no characters, so refill will work as expected
48. a->buffer=(**char**\*)memalign(**sizeof**(**char**)\*buffersize, **sizeof**(**char**)\*buffersize); //needed to change mem type so used memalign instead of malloc.. this meant changing the struct a little bit due to syntax
50. refill(a);
51. **return** a;
52. }
54. //------------------------------------------------------------------
55. **char** cr\_read\_byte(cr\_file\* f){
56. printf("   "); //prints a space before each byte is read
57. **if** (f->usedbuffer==f->bufferlength){ //the character stored in the pointer of f is accessed by usedbuffer, its then checked to see whether the buffer is full
58. printf(" REFILL "); //prints refill when the buffer is full
59. refill(f); //if buffer isnt full then it will continue filling
60. }
61. **return** f->buffer[f->usedbuffer++]; //incrementing usedbuffer by 1
63. **return** EOF; // this is just so the compile works...
64. }

**Code from cache\_reader.h**

1. #include <stdio.h>
2. #include <stdlib.h>

5. //The internals of this struct aren't important
6. //from the user's point of view
7. **typedef** **struct**{
8. **int** file;        //File being read, reads as ASCII
9. **int** bufferlength;  //Fixed buffer length
10. **int** usedbuffer;    //Current point in the buffer
11. **char**\* buffer;      //A pointer to a piece of memory
12. //  same length as "bufferlength"
13. } cr\_file;
15. //Open a file with a given size of buffer to cache with
16. cr\_file\* cr\_open(**char**\* filename, **int** buffersize);

19. //Close an open file
20. **void** cr\_close(cr\_file\* f);
22. //Read a byte.  Will return EOF if empty.
23. **char** cr\_read\_byte(cr\_file\* f);
25. //---------------------------------------------------------
27. //Refill an empty buffer.  Not intended for users
28. **int** refill(cr\_file\* buff);
29. Screenshot of execution of the code

