**Assignment 7**

**Due, Sunday, December 5, 2021 for maximum 100%**

**Monday, December 6, 2021 for maximum 90%**

**Tuesday, December 7, 2021 for maximum 80%**

**Wednesday, December 8, 2021 for maximum 70%**

**Deliverables**

To complete this assignment you must submit your **OSManagement.c** to Webcourses.

**Project description**

This project will require students to simulate the behaviors of an operating system with a series of assignments.

1. Simulate process allocation to memory blocks based on memory management algorithms First Fit, Best Fit, Next Fit, and Worst Fit.
2. Simulate file management of directories and files stored in a directory.
3. Simulate multi-threaded programming with the POSIX (Portable Operating System Interface) threads (a.k.a. pthreads).

**C programming language integrated development environment (IDE)**

1. Code::Blocks ~ NOT Mac compatible
2. Visual Studio Code
3. Atom
4. <https://replit.com/>
5. XCode

**Assignment Scope: Multithreading with mutual exclusion**

1. Update source code file OSManagement.c to include preprocessor directives, function multiThreads, and function threadFunction.
2. Create multiple threads that call the same function providing mutual exclusion.
3. Familiarization with multi-threaded programming with the POSIX (Portable Operating System Interface) threads, or pthreads.

**References**

1. Thread library
   1. <pthread.h>
      1. <https://pubs.opengroup.org/onlinepubs/7908799/xsh/pthread.h.html>
   2. <unistd.h>
      1. <https://pubs.opengroup.org/onlinepubs/009604499/basedefs/unistd.h.html>
2. Threading in C
   1. https://www.includehelp.com/articles/threading-in-c-programming-language-with-gcc-linux.aspx
3. Multithreading in C
   1. <https://www.geeksforgeeks.org/multithreading-c-2/>
4. Mutual exclusion for multithreading in C
   1. <https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/>

Diagram

Description automatically generated

**Tasks**

|  |  |
| --- | --- |
| Activity | |
| OSManagement.c | 1. Make a copy of **OSManagement.c** then update the C source code |
| preprocessor | 1. Include the following C libraries    1. unistd.h    2. pthread.h 2. Function prototype for function **multiThreads ()** 3. Function prototype for function **threadFunction ()** |
| main() | 1. Update the main function so when the user enters menu selection value 3, function **multiThreads** is called |
| multiThreads() | 1. Write the function **multiThreads** to do the following    1. Return type **void**    2. Empty parameter list    3. Declare a constant of data type intto store the number of threads created (i.e. **SIZE**) set equal to **7** (seven)    4. Declare a looping variable of data type **int** (i.e. **i)**    5. Declare a variable of data type int to store the return value from function **pthread\_create** (i.e. **error**)    6. Declare a variable of data type **pthread\_t** as an array, size 7 (i.e. **tid**)    7. Loop **SIZE** times to do the following       1. Set variable **error** equal to function call **pthread\_create ()** passing as arguments          1. The address of the element in thread array **tid[i]**          2. NULL          3. The address of function **threadFunction**          4. Explicit type cast (void \*) of the address of the element in thread array **tid[i]**       2. Evaluate if variable **error** is not equal to 0 (i.e. indicating the thread could not be created)          1. Output to the console explicit text          2. "\nThread can't be created : [%s]" “Press `Enter' to continue . . .”, strerror(error)    8. Loop **SIZE** times to do the following       1. Call function **pthread\_join ()** passing as arguments          1. The element in thread array **tid[i]**          2. NULL |
| threadFunction () | 1. Write function **threadFunction ()** to do the following    1. Return type **void \***    2. Parameter list includes **void\* vargp**    3. Declare a constant int for the number of **LOOP** iterations (i.e. 10)    4. Declare a looping variable, data type int (i.e. **i**)    5. Declare a variable, pointer of data type int, to store the thread id (i.e. **myid**) set equal to explicit type cast to integer pointer of parameter **vargp**    6. Output the that thread # has started, similar to Figure 2    7. Loop 10 iterations       1. Output that thread # is printing, similar to Figure 2    8. Output that thread # has finished, similar to Figure 2    9. Return NULL |
| OSManagement executable |  |
| Test Case 1 | Test Case 1 passes |
| Test Case 2 | Test Case 2 passes |
|  | Source compiles with no errors |
|  | Source compiles with no warnings |
|  | Source runs with no errors |
|  | Source includes comments |

**Test Cases**

|  |  |  |
| --- | --- | --- |
| Test Case 1 | Run executable | The executable runs  The output in the command prompt should be similar to *Figure 1 displayMenu function output* |
| Test Case 2 | **User enters 3 to select Multithreading** | Console output should be similar to *Figure 2 Multithreads* |

Text

Description automatically generated

A screenshot of a computer screen

Description automatically generated with medium confidenceFigure 1 displayMenu function output

Figure 2 Multithreads