## Programming Exercise D2 - Multiple Processes

### Assignment Objectives

* Practice using the fork() function to create multiple processes
* Practice implementing a program in C that consists of several source code files

### Assignment Summary

In this assignment, you will finish the coding of a partially-completed C program that uses the fork() and wait() functions to create child processes and then wait for each of them to terminate. The program finds the largest integer value in an array of integers. Provided on the command line are the number of children processes to be used to search the array, and the total number of entries in the array. In the Software Requirements section below is an explanation of the two functions that you will implement for this program.

The **sample-runs.txt** file contains a series of sample runs of the program. The output from your part of the program should match the format and style of the (Child process#) lines shown in these sample runs. The rest of the lines are printed by the parent process using code in the main module. Note that the actual maximum values found by the children processes in your program will vary depending on the contents of the integer array.

### Software Requirements

You shall implement the following two functions:

* **void searchIntegerList(int list[], int nbrOfListEntries, int childrenCount)**

This function shall first compute how many numbers to give to each child based on the total number of entries in the list and the number of children. Each child shall receive the same amount of numbers to search except for possibly the last child, who may get less. The function shall then use a for loop (as shown in the Robbins slides) and the fan method to distribute the list of numbers out to each child. Inside the for loop, the parent shall set two local variables. One variable is the starting index for the next child that is created. The other variable is the number of entries to be searched by the next child that is created. See the section below named **An Algorithm for Partitioning the List** for more details.

The source code for the **child** process shall call the findMaxInteger() function to search its part of the integer array. It shall then call the exit() function with a parameter containing the value of the maximum integer found.

The source code for the **parent** process shall calculate the starting index for the next segment of the array to be searched by the next child. It shall also make adjustments, when necessary, to the number of entries value that is used by the last child.

The screen output (i.e., standard out) given in the sample-runs.txt file demonstrates how the parts of a list are divided up among each of the children.

* **findMaxInteger(int list[], int startingIndex, int numbrOfEntries)**

Based on the parameters passed, this function shall do the following:

* 1. Initialize a local variable named maxInteger to contain the first entry in the array
  2. Calculate the stopping index for searching the array
  3. Print the "(Child process ..." message in the format shown in the sampleruns.txt file
  4. Use a for loop that searches for the highest integer value in the array within the range assigned to that child, and changes the value of the maxInteger variable as appropriate
  5. Return the maximum value that was found

### An Algorithm for Partitioning the List

This is an algorithm for partitioning the list of entries out to each of the children. Use this algorithm in the searchIntegerList()function.

* 1. Declare a variable of type integer to hold the starting index
  2. Declare a variable of type integer to hold the size of each subgroup of indexes
  3. Set the starting index to zero
  4. Set the size of the subgroup of indexes to the truncated integer quotient of the total number of list entries and the total number of children
  5. Loop from the first child through to the last child
     1. Perform the fork operation
     2. Check for a failed fork operation
     3. For the child, call the findMaxInteger() function. When that function is complete, call the exit() function
     4. For the parent
        1. Increase the starting index by the amount of the size of the subgroup of indexes
        2. If the parent has just performed the fork operation to produce the second to the last child (i.e., childrenCount - 1), then do the following
           1. If the sum of the starting index and the size of the subgroup is less than total number of list entries then set the size of the subgroup to the difference between the total number of list entries and the starting index

### Assignment Directions

1. The program consists of three files described below. Download each of these files from the assignment page on Blackboard:
   * **program-driver.c** - This file contains a completed main() function and other functions that parse the command line, fill the integer array with randomly-generated numbers, and gather the return values from the child processes. Make no changes to this file
   * **listSearcher.h** - This header file contains the prototype for the **searchIntegerList()** function that the driver will call to search the integer array. Make no changes to this file
   * **search-module-student.c** - This file contains the stubbed-out definitions of the **searchIntegerList()** function and the **findMaxInteger()** function. This is the file where you will put your source code
2. Change the name of the searc-module-student.c file to search-module.c,
3. Implement the functions described in the Software Requirements section
4. After you complete the implementation of your source code and the testing of the program, only submit your search-module.c file to Blackboard. Do **not** submit the other two source code files because the instructor already has them

### Design and Implementation Constraints

* Follow the same coding standards given in the previous assignments.