CS3813 Systems Programming

Test 2 – Programming

____ 12:45 – 2:00 Sec 1

____ 3:55 – 5:10 Sec 2

Instructions: What follows is the skeleton for one or more programs written using system calls. Each problem represents a section of missing code. Using the provided comments, complete each section of code.

You will be expected to submit both a written and digital copy of your final work. Your final answers must be legible. There isn't enough space on the test to record your answers, so you will be expected to use additional paper to do so.

Do not seek outside assistance for this part of the test. You can use lectures, notes, man pages, and sources like cpp reference and google to investigate the syntax for any system call or feature of C.

Check the Canvas page for an execution video to see how the expected behavior for this program.

```
/* Write a multiple concurrent process program that does the following
   1. Prompts the user for the number of integers to enter
   2. Prompts the user for each integer and writes them into
      a file named data.dat
   3. Determines how many integers are > 100
   4. Determines how many integers are < 100
   5. Outputs the total count for each group.

   The program should perform this task in the following way:
   Create a producer child that:
   1. Prompts the user for the number of integers to enter
   2. Prompts the user for each integer and writes them into a file
      named data.dat
   Create a consumer child that:
   1. Determines how many integers are > 100
   2. Outputs that value
   3. Sets that value to its exit value
   Create a consumer child that:
   1. Determines how many integers are < 100
   2. Outputs that value
   3. Sets that value to its exit value
   Create a parent that:
   1. Creates children 1 - 3
   2. Pauses.
 */

/* Include files */
#include <stdio.h>
#include <stdlib.h>

/* Problem 1. Add the necessary include files for this program */

/*
   Global variables:
   For this project I am storing the pid's of the three children
   and two integers that serve the role of flags and counters

 */

/* Problem 2. Declare the global variables for pid's of the three
   children and the two integers that serve the role of flags and
   counters. The first flag deals with the large count; the second
   flag deals with the small count.
 */

#define BUF_SIZE 1024

/* myPrint is a convenience function to use when we are in a signal
```

```
  handler. This is because printf uses buffered I/O.
 */
void myPrint(const char *str)
{
  if (write(STDOUT_FILENO, str, strlen(str)) == -1)
  {
    perror("write");
    exit(EXIT_FAILURE);
  }
}

/* Signal handlers
  We will need to implement at least two signal handlers.
  One for the parent and a minimum of one for the children
 */

/* Problem 3. Define and write the signal handler for the parent */

/* This is the signal handler for the parent.
  This function handles SIGCHLD, SIGUSR1, SIGUSR2.

  On SIGUSR1, increment the counter for large by 1
  On SIGUSR2, increment the counter for small by 1
  On SIGCHLD,
   1. Loop while children remain that need to be cleaned up
     a. if the child that is cleaned up is child 1,
       send child 2 SIGUSR1
       send child 3 SIGUSR2
   2. If no children remaining, output: larger: large
                                        smaller: small
     and exit the parent.
 */

/* Problem 4. Define and write the signal handler for the children */

/* This is the signal handler for the children.
  This function handles SIGUSR1 and SIGUSR2.

  On SIGUSR1, set the first flag to 1.
  On SIGUSR2, set the second flag to 1.
 */

/* Functions for each of the children.
  We will be writing functions for each of the three children.
  This should make it easier to answer the questions on threads.
 */

/* Problem 5. Define and write the function for child 1. */
/* This is the function for child 1.
   It doesn't require any data nor does it return data.

   This function does the following:

   1. Prompts the user for the number of integers to enter
   2. Opens the file data.dat for write. This should result in
      a new file each time.
   3. Prompt the user to enter each desired integer and write them into
      the file
   4. Close the file.
 */

/* Problem 6. Define and write the function for child 2. */

/* This is the function for child 2.
   It doesn't require any data nor does it return data.

   This function does the following:

   1. Assign a signal handler for SIGUSR1
   2. Loop while the first flag is 0
      a. pause
   3. Open the file data.dat for read
   4. Reads each value from the file
      a. if the value > 100,
           increments the counter
           sends SIGUSR1 to the parent
   5. Close the file.
   6. Outputs: total larger: counter
 */

/* Problem 7. Define and write the function for child 3. */

/* This is the function for child 3.
   It doesn't require any data nor does it return data.

   This function does the following:

   1. Assign a signal handler for SIGUSR2
   2. Loop while the second flag is 0
      a. pause
   3. Open the file data.dat for read
   4. Reads each value from the file
      a. if the value < 100,
           increments the counter
           sends SIGUSR2 to the parent
```

```
  5. Close the file.
  6. Outputs: total larger: counter
 */

/* This function forks a child and runs the function passed
  in after the child has successfully forked. I have provided
  it to make the code easier to read.
 */
pid_t hndlFork(void (*child)(void))
{
  pid_t p;
  p = fork();
  if (p == -1)
   {
    perror("fork");
    exit(EXIT_FAILURE);
   }
  if (p == 0)
   {
    child();
   }
  return p;
}

/* Problem 8: Define and write the function main */

/* This is the function for function main prior to calling fork
  and the parent after calling fork.
  The function does the following:
  1. Assign a signal handler for SIGCHLD
  2. Fork child 2 and child 3
    You can do this by calling hdnlFork as follows:
    child = hndlFork(childFcn);
    where child stores a pid and childFcn is the
    function you have written to handle that child
  3. Assign a signal handler for SIGUSR1, SIGUSR2
  4. Fork child 1
  5. Loop forever
    a. pause
 */

int main(int argc, char *argv[])
{

  exit(EXIT_SUCCESS);
}
```

```
/* Write a threaded program that does the following
   1. Prompts the user for the number of integers to enter
   2. Prompts the user for each integer and writes them into
      a file named data.dat
   3. Determines how many integers are > 100
   4. Determines how many integers are < 100
   5. Outputs the total count for each group.

   The program should perform this task in the following way:
   Create a producer thread that:
   1. Prompts the user for the number of integers to enter
   2. Prompts the user for each integer and writes them into a file
      named data.dat
   Create a consumer thread that:
   1. Determines how many integers are > 100
   2. Outputs that value
   3. Sets that value to its exit value
   Create a consumer thread that:
   1. Determines how many integers are < 100
   2. Outputs that value
   3. Sets that value to its exit value
   Create a main thread that:
   1. Creates threads 1 - 3
   2. Waits on the values of threads 2 and 3.
   3. Outputs the values from threads 2 and 3.
 */

/* Include files */
#include <stdio.h>
#include <stdlib.h>
/* Problem 1. Add the necessary include files for this program */

/*
  Global variables:
  We will need a mutex, a condition variable, and a predicate variable.
  Recall that the predicate variable is the variable we use to determine
  whether data was available prior to our first call to pthread_cond_wait
 */

/* Problem 2. Declare the global variables for the predicate variable,
  the mutex, and the condition variable. Do no forget to initialize
  the mutex and the condition variable
 */
```

```
/* This is a convenience function for dealing with errors
   and threads
 */

void hndlError(int error, const char *str)
{
  if (error == 0) return;
  errno = error;
  perror(str);
  exit(EXIT_FAILURE);
}


/* Define the three thread start functions.
   You can name them whatever you wish
 */

/* Problem 3. Define and write the start function for thread 1 */

/* This is the start function for thread 1.
   This function does the following:

   1. Detaches it self
   2. Prompts the user for the number of integers to enter
   3. Locks the mutex
   4. Opens the file data.dat for write. This should result in
      a new file each time.
   5. Prompt the user to enter each desired integer and write them into
      the file
   6. Close the file
   7. Set the predicate variable to 1
   8. Unlock the mutex
   9. Signal the condition variable
  10. exit the thread with a value of NULL
*/

/* Problem 4. Define and write the start function for thread 2 */

/* This is the start function for thread 2.
   This function does the following:

   1. Declare and allocate space for the counter
   2. Locks the mutex
   3. Loops while the predicate variable is 0
      a. waits on the condition variable and the mutex
   4. Opens the file data.dat for read
   5. Unlocks the mutex
   6. Reads each value from the file
   a. if the value is > 100, increments the counter
  7. Closes the file
  8. Outputs: total larger: counter
  9. exit the thread with a value of the counter
*/

/* Problem 5. Define and write the start function for thread 3 */

/* This is the start function for thread 3.
  This function does the following:

  1. Declare and allocate space for the counter
  2. Locks the mutex
  3. Loops while the predicate variable is 0
    a. waits on the condition variable and the mutex
  4. Opens the file data.dat for read
  5. Unlocks the mutex
  6. Reads each value from the file
    a. if the value is < 100, increments the counter
  7. Closes the file
  8. Outputs: total smaller: counter
  9. exit the thread with a value of the counter
*/

/* Problem 6: Define and write the function for the main thread */

/* This is the function for the main thread (i.e. main)
  The function does the following:
  1. Create threads 1 - 3.
    We don't know what order they will execute in, but create them
    in the order 2, 3, 1 for this problem,
  2. Join thread 2 and store its return value.
    This should be the total number of values >  100
  3. Join thread 3 and store its return value.
    This should be the total number of values < 100
  4. Output: larger: return value from thread 2
         smaller: return value from thread 3
 */

int main(int argc, char *argv[])
{
 exit(EXIT_SUCCESS);
}
```