Xander May - 101038396
Joshua Fryer - 101032335

================= PSEUDO-CODE =====================


```
Initialize the shared memory array {
    create 5 shared memory sections, each containing a number.
    get pointers to each of the shared memory sections and put
them in an array.
    assign semaphores to each of the individual sections.
    claim all semaphore locks, and populate the shared memory
segments with
        provided values.
    release the locks so that child processes can claim them as
soon as they
        start running.
}
```

This code section allocates the shared memory that will later be used by the various child processes during their execution. It does this in such a way that it ensures that later the child processes will not begin running before they have all had a chance to start

```
Spawn four worker processes {
    fork() four child processes
    add 1 to the number of active children, stored in a separate
semaphore.
    assign two numbers (from shared memory) to each process,
these are what
        they are responsible for sorting
}
```

As the child processes are created a control semaphore is incremented, allowing the parent process the ability to see how many of the child processes have come online and how many are currently processing

```
Worker processes continually check their numbers {
    if the numbers are out of order {
        claim the locks on those numbers.
        swap the numbers.
        release the locks on those numbers.
    }
}
```

This section of the code is responsible for the actual sorting of the integers. Each number in the array has a semaphore associated with it that must be held for a process to perform operations on it. This ensures exclusive operations for the child processes.

```
Parent process waits until  all children finish working and the
array is sorted {
        tell the child processes to stop
        print out the sorted array, as well as its min, max, and
mean.
    }
}
```

The parent waits for two conditions to ensure that the task is complete. First it ensures that there are currently no child processes performing a switch, and second it checks if the array is sorted. If either of these conditions are false then it will wait and check again.

This program sorts the array of 5 given numbers in non-increasing order.
It is in essence a variation on the dining philosophers problem; implementing
a bubble sort using concurrent processes. In this case the processes

cannot deadlock because the dependencies are not circular, and the ends are only accessed
by a single member, but our solution is extensible for longer number arrays
and would work if the arrays had any number of dependencies. If you did not want to
write an extensible solution you only need locks/semaphores for the elements that are
accessed by more than one member; in this case the three middle elements.

Our solution ensures mutual exclusion with all elements using one semaphore per array
element, and uses one final semaphore for use as a control to ensure the main process
does not prematurely terminate the children doing the sorting. Furthermore, our solutions
use of semaphores allows all child processes to sort concurrency without issue, meaning a
faster execution time.

--------------- DISCUSSION -----------------------

For all of the test cases in sampleArrays.h, the program executes correctly in
sorting the supplied arrays in non-ascending order. It has also worked for every
instance of user-supplied numbers. The following screenshots illustrate the correctness of
our solution, and that it satisfied the requirements of the assignment.

This is an example of compiling and running the sorting algorithm for the two sets of required
data (we have them as hardcoded examples, 5 of which are available for testing).

```
xmay@merlin: ConcurrentSort $ make
make: 'STATS' is up to date.
xmay@merlin: ConcurrentSort $ ./STATS 0
Would you like to use debug mode? [y/n] n
Allocated shared memory.
Using test case number 0.
Parent aquiring locks and filling values.
Parent holds all locks and values are initialized.
Parent released locks. The array is: 5 6 8 2 7
Created child process responsible for numbers 5 and 6
I am worker 0 and I handle numbers 5 and 6
Created child process responsible for numbers 6 and 8
I am worker 1 and I handle numbers 6 and 8
Children are ready
Created child process responsible for numbers 8 and 2
Created child process responsible for numbers 2 and 7
I am worker 2 and I handle numbers 8 and 2
I am worker 3 and I handle numbers 2 and 7
Parent notified child with PID 3554 that execution is complete
Parent notified child with PID 3555 that execution is complete
Parent notified child with PID 3556 that execution is complete
Parent notified child with PID 3557 that execution is complete
The final sorted array is: 8 7 6 5 2
The mean value of the array is: 5.
The minimum value of the array is: 2.
The maximum value of the array is: 8.
```

```
xmay@merlin: ConcurrentSort $ make
gcc -o STATS main.c semaphoreOps.c -I -g.
main.c: In function 'main':
main.c:437:29: warning: passing argument 2 of 'init_array' discards 'const'
         init_array(num_ids, test_arrays[testCase]);
                             ^
main.c:181:6: note: expected 'int *' but argument is of type 'const int *'
 void init_array(int* mem_id, int* value_array) {
      ^
xmay@merlin: ConcurrentSort $ ./STATS 1
Would you like to use debug mode? [y/n] n
Allocated shared memory.
Using test case number 1.
Parent aquiring locks and filling values.
Parent holds all locks and values are initialized.
Parent released locks. The array is: 10 9 11 5 7
Created child process responsible for numbers 10 and 9
I am worker 0 and I handle numbers 10 and 9
Created child process responsible for numbers 9 and 11
I am worker 1 and I handle numbers 9 and 11
Created child process responsible for numbers 11 and 5
Children are ready
Created child process responsible for numbers 5 and 7
I am worker 2 and I handle numbers 11 and 5
I am worker 3 and I handle numbers 5 and 7
Parent notified child with PID 3172 that execution is complete
Parent notified child with PID 3170 that execution is complete
Parent notified child with PID 3169 that execution is complete
Parent notified child with PID 3171 that execution is complete
The final sorted array is: 11 10 9 7 5
The mean value of the array is: 8.
The minimum value of the array is: 5.
The maximum value of the array is: 11.
```

And this is an example of allowing the user to input custom data for use in sorting.

```
Would you like to use debug mode? [y/n] n
Allocated shared memory.
Please input 5 numbers:
>88
>43
>2357
>32508
>523
Parent aquiring locks and filling values.
Parent holds all locks and values are initialized.
Parent released locks. The array is: 88 43 2357 32508 523
Created child process responsible for numbers 88 and 43
I am worker 0 and I handle numbers 88 and 43
Created child process responsible for numbers 43 and 2357
I am worker 1 and I handle numbers 43 and 2357
Created child process responsible for numbers 2357 and 32508
Children are ready
I am worker 2 and I handle numbers 2357 and 32508
Created child process responsible for numbers 32508 and 523
I am worker 3 and I handle numbers 32508 and 523
Parent notified child with PID 4941 that execution is complete
Parent notified child with PID 4943 that execution is complete
Parent notified child with PID 4940 that execution is complete
Parent notified child with PID 4942 that execution is complete
The final sorted array is: 32508 2357 523 88 43
The mean value of the array is: 7103.
The minimum value of the array is: 43.
The maximum value of the array is: 32508.
```

This is an example of the program running with debug mode enabled, though it did not all fit on screen.

```
Child 3 switching 2 with 7
Child 0 not switching 8 with 6
Child 2 not switching 5 with 2
Child 1 not switching 6 with 5
Child 0 not switching 8 with 6
Child 3 not switching 7 with 2
Child 1 not switching 6 with 5
Child 2 switching 5 with 7
Child 0 not switching 8 with 6
Child 3 not switching 7 with 2
Child 1 not switching 6 with 5
Child 0 not switching 8 with 6
Child 2 not switching 7 with 5
Child 3 not switching 5 with 2
Child 0 not switching 8 with 6
Child 2 not switching 7 with 5
Child 1 switching 6 with 7
Child 3 not switching 5 with 2
Child 0 not switching 8 with 6
Child 2 not switching 7 with 5
Child 3 not switching 5 with 2
Child 1 not switching 7 with 6
Child 0 not switching 8 with 7
Child 2 not switching 6 with 5
Child 2 not switching 6 with 5
Parent notified child with PID 4075 that execution is complete
Child 0 not switching 8 with 7
Parent notified child with PID 4073 that execution is complete
Child 3 not switching 5 with 2
Parent notified child with PID 4076 that execution is complete
Child 1 not switching 7 with 6
Parent notified child with PID 4074 that execution is complete
The final sorted array is: 8 7 6 5 2
The mean value of the array is: 5.
The minimum value of the array is: 2.
The maximum value of the array is: 8.
```