**Abstract**: (This is the paper in brief and should state the basic contents and conclusions of the paper. It should be complete enough to understand what will be covered in the project.)

In this paper you will find the programming solution to a process synchronization problem, "baboons crossing a canyon on a rope from both sides". It contains a breakdown of the program's functions and a brief understanding of its purpose. This paper will present results showing the solution to the synchronization problem. It also contains the contemplation of results to quantify a basis of anything new we've learned for the duration of the project.

**Introduction**: (This is a short overview of what you did, and what you learnt. This should contain more motivation than the abstract.)

For this project we wrote a program that solved a synchronization problem using semaphores and the Pthreads library. By solving this problem we continue to enhance our understanding of threads, and semaphore use. Overall the motivation behind this project would include the addition of linked list queue buffers and a better understanding of synchronization of a shared resource.

**Methodology**: (Description of algorithms you used and a description of the platform you used to the level of detail such that someone else could reproduce the same experiments elsewhere.)

The concept of the program is to have a canyon with baboons on both sides and a rope that spans the canyon. The baboons on the left side want to cross over to the right, and vise versa for the baboons on the right. There can only be three baboons on the rope at a time, and the three baboons must all be going in the same direction when they cross; or they will collide and fight. In addition, the order of the baboons crossing the rope should be preserved; such as FIFO order. A stream of baboons going one direction should not impede baboons from the other direction from crossing; a method to prevent starvation of either side.

The baboons behavior should be implemented using Pthreads library. An input parameter of "time required to cross" needs to be implemented, using sleep function; This will signify the time it takes for a baboon to cross the rope, it also signifies how long the shared resource will be occupied. The baboons need to be read from a file, were the format of baboons should be "L,R,L,L,R,R,R,L,R", without the quotation marks. Every L indicated a baboon from the left is trying to go right, and R indicates the opposite of L.

In our code we use three buffers, a rope of size 3 and two linked list buffers, one for the left baboon queue and one for the right baboon queue. As the file is read, baboons will be queue to their respective buffers, ready and waiting to cross the rope. Next, we have four semaphores that we use to protect our shared resources; direction\_mutex, rope\_space, left\_mutex, right\_mutex. Aside from the main function, we also have makeBaboonCross, baboonCrossing, leftQueueFunction, rightQueueFunction, and a produce function. Below is a brief description of each one of these functions.

* void makeBaboonCross(char dir): Initiates semaphore wait on rope\_space, increments baboons crossing on rope, creates a baboon thread with direction, then detaches thread.
* void baboonCrossing(void \*dir\_ptr): sleeps for 2 milliseconds, displays direction of baboon crossing, releases semaphore rope\_space, decrements baboons crossing on rope, closes thread.
* void leftQueueFunction(): This is populated by the produce() function. When called this function set the direction\_mutex and left\_mutex semaphore to wait, then de-queues a character from the left buffer queue. Next it releases the left\_mutex semaphore and calls the makeBaboonCross(c) function. Lastly, while the baboon is crossing we want to release the direction\_mutex semaphore and close any threads open by the left queue.
* void rightQueueFunction(): The rightQueueFunction() works exactly like the leftQueueFunction(), except we modify the right\_mutex semaphore, instead of the left.
* void produce(): This function reads the characters from a file, where the string meets the requirements above, "L,R,R,L,L,R", and assigns the character to its respective queue. When inserting an item into either the left or right queue, we activated semaphore wait and post on the queues. Lastly, we inserted a '\*' character at the end of the queues to let the threads know when to stop.

Finally, in the main function, we initialize our semaphores and queues. We also create our pthread identifiers and attributes. Next, we request the length of time it takes a baboon to cross the rope from the user. We then create our producer, left queue and right queue threads, and join them. Lastly, when the threads have completed the semaphore will be destroyed.

**Results**: (Include your program, input data, outputs of your program)

**Conclusion**: (Summarize the conclusions here, and discuss things you have learnt during this experimentation)

**REFERENCES**