CS4328: Project #2
Student: Jose B. Gutierrez
Professor: Mina Guirguis
Date: 12/01/2016
Class: Operating Systems

File Attachments:

- *main.c:* source file for the program.
- *busy_wait.c:* source file for an alternative way to run the game. This version implements a busy wait mechanism.
- *makefile*: compiles and cleans main.c. if a make clean is issued it will remove all *.txt *.o files.
- results.txt: results of the game for 5 different seed variables.

Running the Simulator:

Unzip the folder osPr2.zip

```
j_g572@hercules:~/osPr2$ make
gcc -c main.c
gcc main.o -o play -lpthread
j_g572@hercules:~/osPr2$ ./play 25
```

The following will run the simulator with a seed value of 25 and will store the results in a file called results.txt

```
j_g572@hercules:~/osPr2$ make clean
rm *.o *.txt play
j_g572@hercules:~/osPr2$
```

Clean directory for removal of all files (including the results).

Project Description:

- Simulation of a card game called pair war.

Project Scope:

- Synchronization of threads using the POSIX library.

Data Structures Implemented:

- Card Stack Array: Mimics a stack of 52 cards.
- Player Array: Holds each of the player's cards at any given moment (1 index for each player and 3 indexes total).

Design Approach #1: Busy wait - *busy_wait.c:*
The very first of the designs implemented for this project was one which used a global variable in order to signal the corresponding thread to 'do work'. Although the implementation of this method was fairly easy, the data produced by a test run demonstrated the inefficiency of this design approach. During a busy wait any given thread on any order will attempt to run by checking whether a certain variable or condition has became true or false (depending on the logic of the algorithm).

More specifically for this project, by implementing this method players do not know when it is their turn and they keep asking for their turn randomly at a very high speed.

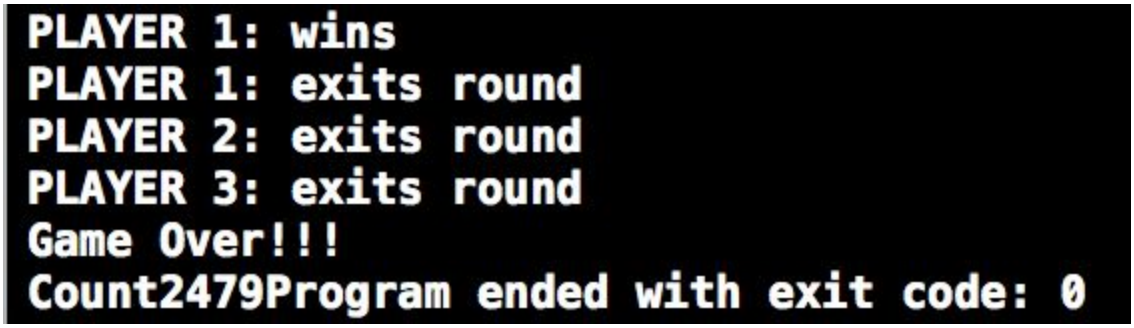
By putting a global variable counter to show the number of times a player or the dealer attempted to ask for its turn in each of the threads we can get an estimate of the efficiency of the design. For his example I used seed variable 2 and captured the results for the test run:

```
1
2  static void * dealer_RUN(void * _)
3  {
4      while(gameOver == false)
5      {
6          pthread_mutex_lock(&newTurn);
7          count++;
8          if(turn == 0)
```

count is the counter variable used to observe how busy the threads were using for their turn.

```
static void * player_RUN(void * arg)
{
    int ptr = (intptr_t) arg;
    pthread_mutex_lock(&newTurn);
    while(gameOver == false)
    {
        count++;
        pthread_cond_wait(&turnCond,&newTurn);
```

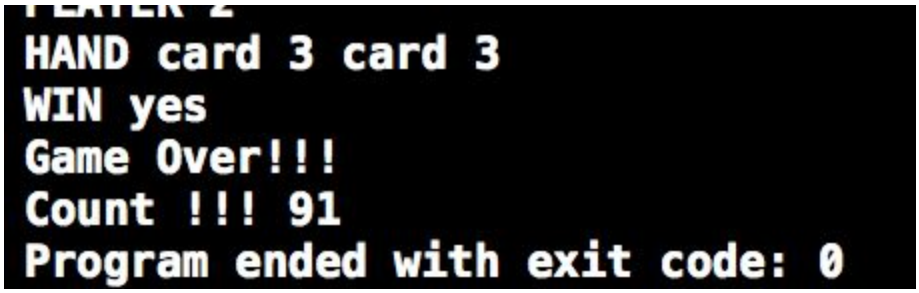
Here three different players share this piece of code in this thread.



When the game ended the count variable was set to 2479. Later the results of a different approach are compared.

Design Approach #2: Thread Watcher - *main.c* Implemented

The other design approach was to use a thread (the dealer) to let each one of the players know that it was their turn. While the players were waiting they simply did nothing. Once a player was signaled, it would signal the dealer for the dealer's turn or to assign another player for their turn. The same count variable was used in each of the threads in order to keep track of the number of times the threads were busy. The same seed of 25 was used



Here the number of times the thread was busy was significantly lower which proves that this design was more efficient.

Unlike the busy wait design if we use the same variable for the seed argument the count variable is guaranteed to be the same each time the game is ran since we know exactly that the threads will not be asking for their 'turn' and will be asleep while they wait.

Note: that this design could be further better implemented to improve efficiency. This would be by instead of having a thread watcher, simply signal the other player that it is their turn. This will avoid calling the dealer each time to assign a turn to a player. (I could not implement this. Will still be trying on the next few days to see if I can get it to work, however).