**Program 2 Report- Multiprocessor manager/worker program for**

**card game Twenty-One**

Name: Adwaith Moothezhath Rajesh

Date: March 14, 2021

NetID: axm180043

CE 4348.001

**Problem Statement**

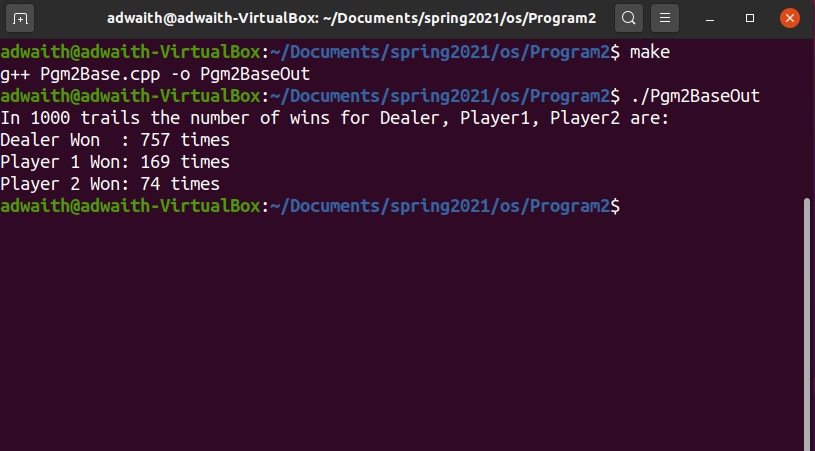
The assigned task was to make a multiprocessor manager/worker application to implement the game twenty-one. Two players and a dealer should be implemented with each of them having a different strategy than others for playing the game 21. One thousand trials should be run each time with a shuffled card deck. The number of times player 1, player 2, and the dealer won should be recorded each trial to see who won the most after one thousand trials. The importance of this project is to use fork and pipe efficiently.

**Approach**

The primary approach was first creating a deck of cards and a shuffle algorithm to shuffle the cards. Next, create three pipes using pipe() which can be used to communicate between the process. The three pipes are the first one is to the dealer; the second pipe is to player 1 and the third pipe is to player 2. Next, I created the players using fork() which creates the child process. Next, I ran a thousand trials each time shuffling the deck and recording the wins of the two players and the dealer each time. Finally, two different algorithms were made for the two players. The first player was implemented smarter than the second player as the first player can choose the card ace to be either 1 or 11. The second player always chooses ace to be 11. The dealer had the standard dealer algorithm. The language used for this program was C++ and the environment I ran the code was on Ubuntu. I made a make file to run the C++ program on Ubuntu 20.04 LTS environment.

**Solution**

I wrote the program in C++ in an Ubuntu 20.04 LTS environment. First, I made a deck of cards for the 4 suits clubs, diamonds, hearts, and spades each with 13 cards and only the values of the cards. I assumed Ace to have a value of 11 and Jack, Queen, King to have the value of 10. Then I put all the different suits into a card deck of 52 cards. Secondly, I created three pipes and error-checked each one to make sure it opens properly. The first pipe is to the dealer, the second pipe is to player 1 and the third pipe is to player 2. The pipes are for process communication to communicate data such as the current card on top of the deck and also players total to the dealer using read and write functions. Thirdly, I used fork() to create children processes and set them up as player 1 and player 2. To use fork and pipes I had to include the unistd.h library. Fourthly, I ran one thousand trials and, on each trail, I shuffled the card deck. I implemented the dealer’s strategy which is if the total value of cards is less or equal to 17 then the dealer will take a card from the deck. I also kept track of if any of the players or dealer goes over 21 then they bust or else they are safe. I then incremented the win by one for whoever has the highest total. Finally, I implemented both the player's strategies. Player 1’s strategy I came up with was after the first two cards if the total is still less than or equal to 14 then player 1 takes a new card. If the card that player 1 picks up is Ace and their total is 12 or more use the value 1 for ace. For player 2’s strategy after the first two cards, if the total is less than or equal to 16 then the player will take one more card. If the card is Ace, then the player will always use the value 11 so the probability of player 2 winning is less than player 1. This C++ program was compiled and ran on Ubuntu 20.04 LTS environment using a make file like shown in Figure 1 below.



**Figure1: Output Screenshot**

The program output displays the sentence in 1000 trials the number of wins for Dealer, Player1, Player2 are Dealer Won: 757 times, Player 1 Won: 169 times, Player 2 Won: 74 times. To compile the program, I used g++ and put it in a make file. To run the code, I typed the command **./Pgm2BaseOut**.

**Appendix**

To run the program first make sure to have Ubuntu 20.04 LTS environment. Next, have the build-essential package downloaded. Next, download the Pgm2Base.cpp and Makefile and put them in the same directory, and run the command **make** in the terminal. Next, run the command **./Pgm2BaseOut** and now the output will be displayed just like the output screenshot above.