```c
//Andrew Ingle 04/07/2021 - Team I - Final Project
//Main Driver Programmer for Server, will fork and execute to create servers


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>

#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include "caleb_server.h"



int main() {

   printf("\nServer Driver is Alive and Creating 3 Servers!\n"); //for
    debugging

   int server_name = 1; //name of first server will be incremented as we go
    along

   unlink("myfifo1");
   if (mkfifo("myfifo1", 0666)==-1){    //Create Fifo to send server names to
    child servers
      printf("Could not create fifo file\n");
      perror("mkfifo() failed");
      return 1;
   }

   // Semaphore Syncronization start //CALEB ADDED SEMAPHORE CODE
   sem_unlink(SEM_READER_NAME);
   sem_unlink(SEM_WRITER_NAME);
   //We initialize the semaphore counter to (INITIAL_VALUE) in caleb_server.h
   sem_t *read_semaphore = sem_open(SEM_READER_NAME, O_CREAT | O_EXCL,
    SEM_PERMS, INITIAL_VALUE_READER);
   sem_t *write_semaphore = sem_open(SEM_WRITER_NAME, O_CREAT | O_EXCL,
    SEM_PERMS, INITIAL_VALUE_WRITER);



   if (read_semaphore == SEM_FAILED || write_semaphore == SEM_FAILED) {
        perror("sem_open(3) error");
        exit(EXIT_FAILURE);
```

```
 }

 //Close the semaphore as we won't be using it in the parent process
 if (sem_close(write_semaphore) < 0 && sem_close(read_semaphore) < 0) {
     perror("sem_close(3) failed");
     // We ignore possible sem_unlink(3) errors here
     sem_unlink(SEM_READER_NAME);
     sem_unlink(SEM_WRITER_NAME);
     exit(EXIT_FAILURE);
 }
 // Semaphore First part done


//creation of the socket to communicate with client
 int server_socket, c;
 server_socket = socket(AF_INET, SOCK_STREAM, 0);

if (server_socket == -1){
     printf("Could not create socket");
  }

struct sockaddr_in server_address, client_address;
server_address.sin_family = AF_INET;
server_address.sin_addr.s_addr = INADDR_ANY;

server_address.sin_port = htons(8001); //for local connections

//Bind
if( bind(server_socket,(struct sockaddr *)&server_address ,
 sizeof(server_address)) < 0){
    printf("bind failed");
}

listen(server_socket, 5); //will update second number to reflect max number
 of customers allowed at a time

//creating 3 servers
for (int i = 0;i<3;i++){

   pid_t pid = fork();

   if(pid < 0) {
      fprintf(stderr, "fork failed for next server.");
      return 1;
   }
   else if(pid == 0) { //next child server
      // send port to server and change port each time
      execlp("./server_main","server_main",NULL); //system call provided by
       Andrew
   }
   sleep(1); //to give time for servers to open
```

```
    }

    //opening fifo to send names to servers
    int fd = open("myfifo1", 01); //this waits until child server has opened
     fifo
       if (fd== -1){
       return 1;
    }

    //sending names to servers
    for (int i=0;i<3;i++){
       write(fd, &server_name, sizeof(int));
       server_name++; //incrementing name
    }

     //sending server_socket id to servers
    for (int i=0;i<3;i++){
       write(fd, &server_socket, sizeof(int));
    }


    sleep(3);
    if (  sem_unlink(SEM_READER_NAME) < 0 || sem_unlink(SEM_WRITER_NAME) < 0){
     perror("sem_unlink(3) failed");
    }
    printf("\nserver_driver complete");

    wait(NULL);

    return 0;
}
```