Inter-Process Communication (IPC) using Pipes

Pre-requisite: Understanding of system calls like fork(), wait(), exec(), exit() and pipe()

Write C program for the problem specification given below. After executing the C program(s), you will need to capture the output in text file format. We learn inter-process communication between parent and child process using pipes in this problem.

Use the following file naming convention to complete your submission.

- C program file will be named as 201901nnn_pipes.c
- Text file with output captured will be names as 201901nnn_pipes.txt, where nnn is 3 digits of your student id

Consider you have two variables for questions and answers as below
char* questions[] = {"Quit", "In which university do you study?", "Which course are you studying?", "What is your area of interest?"};
char* answers[] = {"Quit", "DAIICT", "Systems Software", "Kernel Programming"};

• Parent process will accept input as question number from user. If the question number is between 0 to 3, the actual text of question will be sent from parent to child process using the following.

```
write(fd1[WRITE], questions[que], strlen(questions[que])+1);
```

• Once parent sends the question to child, it just waits for getting answer back from child using the following.

```
bytesRead = read(fd2[READ], message, MSGLEN);
```

• When child process receives the question using the following.

```
bytesRead = read(fd1[READ], message, MSGLEN );
```

• Child process then finds the index of the question that it receives from questions[] string array. It uses that index to get the correct answer from the answers[] string array.

• Child process then replies to the parent the answer using using the following.

```
write(fd2[WRITE], answers[que], strlen(answers[que])+1);
```

• If child finds that question number is equal to 0 (i.e. Quit) then it exits after closing all the open file handles. Parent process upon reading question number equal to 0 from and after receiving “Quit” as the answer from child closes all open file handles and waits for child to exit using wait() system call to ensure that we don’t create a zombie child process. Once parent comes out of wait() system call, it also exits.