Write a Unix C program that:

1. **Creates a child process** using fork().
2. The **child process** must execute the wc -l <filename> command using execlp().
3. The **parent process** must create a **POSIX thread** to count the total number of vowels in the same file.
4. **File I/O must use** open() **and** read() **system calls** (no fopen() / fread() / getc()).

   Please use man command to add the required libraries.
5. Use pthread_join() and wait() to synchronize.
6. You can not use Visual studio.
7. The code and the output from child and parents process should be pasted in the website.

**Input File Generating Process:**

Ubuntu installations already have **OpenSSL** preinstalled because many system components depend on it. OpenSSL is basically the **Swiss army knife of cryptography** in the software world.
It's an open-source toolkit that implements the **SSL (Secure Sockets Layer)** and **TLS (Transport Layer Security)** protocols, plus a big library of **cryptographic functions**.

Here the input file generation process follows below steps:

*Goal :*

- Generate **RSA public and private keys**.
- Use your **private key** to sign a file (create a digital signature).
- View the signature in a **Base64-encoded** format.

---

**Step 1: Create a Text File with Your Registration Number**

- Open a terminal and type:

  *echo "YourRegistrationNumber" > file.txt*

Replace YourRegistrationNumber with your actual registration number. Do **not** include spaces or tabs. This will be the **message** we will sign.

**Step 2: Generate RSA Private Key (2048 bits)**

- Run:

*openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048*

**RSA** is a public-key cryptography algorithm. **Private key** is a secret key that only you should have.
It will be used to create digital signatures. **2048 bits** means the key length — larger key sizes are more secure but slower.

**Step 3: Extract the Public Key**

- Run:

*openssl rsa -in private_key.pem -pubout -out public_key.pem*

The **public key** is derived from the private key. Anyone with your public key can verify your signatures, but **cannot create** a valid signature. This is the core of public-key cryptography.

**Step 4: Create a Digital Signature**

- Run:

*openssl dgst -sha256 -sign private_key.pem -out file.sig file.txt*

*dgst -sha256 → Computes the SHA-256 hash of file.txt.* -sign private_key.pem → Uses your private key to encrypt the hash, producing the **digital signature**. file.sig → The output binary signature file. This ensures that:

1. The file came from **you** (authenticity).
2. The file hasn't been changed (integrity).

**Step 5: View Signature in Base64**

**Run:**

*base64 file.sig > file.sig.b64*

*Digital signatures are binary files. It is hard to send or view in plain text.* Base64 encoding turns binary data into readable ASCII characters so you can easily email or display it. The file file.sig.b64 contains your signature in text form.