Programming Assignment #2, Processes

Write a C program (`time_shm.c` and `time_pipe.c`) that determines the amount of time necessary to run a command from the command line. This program will be run as

```
./time <command [args...]>
```

and will report the amount of elapsed time to run the specified command. This will involve using `fork()` and `execvp()` functions, as well as the `gettimeofday()` function to determine the elapsed time. It will also require the use of two different IPC mechanisms.

The general strategy is to fork a child process that will execute the specified command. However, before the child executes the command, it will record a timestamp of the current time (which we term "starting time"). The parent process will wait for the child process to terminate. Once the child terminates, the parent will record the current timestamp for the ending time. The difference between the starting and ending times represents the elapsed time to execute the command. The example output below reports the amount of time to run the command `ls`:

```
./time ls -l
total 156
-rwxr-xr-x 1 thomas       25616
users                       Feb  4 15:59 a.out
-rw-r--r-- 1 thomas                 16:00
users                     252 Feb  4 output.txt
-rwxr-xr-x thomas                   20:58
1          users      86024 Feb  2 project.exe
-rwxr-xr-x thomas
1          users      25616 Feb  2 21:00 time
-rw-r--r-- thomas
1          users       5144 Feb  2 20:58 time.c
Elapsed time: 0.001448 seconds
```

As the parent and child are separate processes, they will need to arrange how the starting time will be shared between them.

You will write two versions of this program, each representing a different method of IPC.

2a) The first version, `time_shm.c`, will have the child process write the starting time to a region of shared memory before it calls `execvp()`. After the child process terminates, the parent will read the starting time from shared memory. The region of shared memory should be established before the child process is forked, allowing both the parent and child processes access to the region of shared memory.

2b) The second, `time_pipe.c`, version will use a pipe. The child will write the starting time to the pipe, and the parent will read from it following the termination of the child process.

You will use the `gettimeofday()` function to record the current timestamp.
This function is passed a pointer to a `struct timeval` object, which contains two members: `tv_sec` and `tv_usec`. These represent the number of elapsed seconds and microseconds since January 1, 1970 (known as the UNIX EPOCH).

The following code sample illustrates how this function can be used:

```
//   Get current time
timeval_t startTime; gettimeofday( &startTime, 0 );
//   get the end time
timeval_t end_time;
gettimeofday( &end_time, 0 );
//   calculate elapsed time
timeval_t elapsed_time;
timersub( &end_time, startTime, &elapsed_time );

// print elapsed time (microseconds right justified zero filled)

printf( "\nElapsed time: %d.%06d seconds\n",
       elapsed_time.tv_sec, elapsed_time.tv_usec );
```

***Hint*:**

1. For IPC between the child and parent processes, the contents of the shared memory pointer can be assigned the `struct timeval` representing the starting time. When pipes are used, a pointer to a `struct timeval` can be written to - and read from - the pipe.
2. `gcc time_shm.c -otime -lrt` is an example command to compile and link your program
3. `./time ls -l | tee time_shm_output.txt` is an example command to run your program while collecting your program's output to a text file and seeing the output on the console.

Submit (only) the following 4 files to CANVAS.

1. time_shm.c
2. time_shm_output.txt (both files must be present to earn credit)
3. time_pipe.c
4. time_pipe_output.txt (both files must be present to earn credit)