CS3510 – OS1 – Asgn1

Sharath Chandra Sheripally

ES18BTECH11016

What is this?

This program creates a child process and calculates the time it took for the child process to complete its execution in a Linux environment.

How to use?

1. Compile using `` `gcc Asgn1-ES18BTECH11016.c -lrt` `` [-*lrt* explained later]
2. Execute the program using `` `./a.out <command> <args>` `` e.g., `` `./a.out ls -l -a` ``
3. It returns as following

```
./a.out ls -l -a
total 28
drwxrwxrwx 1 tnfssc tnfssc  4096 Dec 20 18:48  .
drwxrwxrwx 1 tnfssc tnfssc  4096 Dec 19 22:10  ..
-rwxrwxrwx 1 tnfssc tnfssc   162 Dec 20 18:35 '~$gn1-Report.docx'
-rwxrwxrwx 1 tnfssc tnfssc 17232 Dec 20 18:48  a.out
-rwxrwxrwx 1 tnfssc tnfssc  1622 Dec 20 18:46  Asgn1-ES18BTECH11016.c
-rwxrwxrwx 1 tnfssc tnfssc   426 Dec 20 18:32  Asgn1-README.txt
-rwxrwxrwx 1 tnfssc tnfssc     0 Dec 20 18:35  Asgn1-Report.docx
drwxrwxrwx 1 tnfssc tnfssc  4096 Dec 20 18:37  .git
-rwxrwxrwx 1 tnfssc tnfssc    12 Dec 20 14:28  .gitignore
-rwxrwxrwx 1 tnfssc tnfssc   426 Dec 20 18:32  README.md
Elapsed time: 15989
```

4. The elapsed time is in microseconds.

How it works?

1. The given command and arguments are passed on to the command as a child process.
2. The child process is created using `` `fork()` ``. This creates a clone of the calling process. It returns PID of the new process to the old/parent process.
3. A shared memory segment between the child and the parent processes is required because the child needs to tell its parent about its start time.
4. This shared memory segment is created using `` `shm_open()` ``. A name and permissions for this memory segment can be passed as parameters.

5. This memory segment is mapped to a pointer using `mmap()`.
6. In the child process, after obtaining current time (start_time) using `gettimeofday()`, it is stored in the shared memory segment using the pointer which maps to the segment.
7. In this case, the current time in epoch is written to the shared memory as a string.
8. Right after storing the time of start, the command given is executed using `execvp()` along with the arguments passed. If `execvp()` fails, the following code below it is run. If it succeeds, that code is not executed.
9. While this is happening in a different thread, the old/parent process is told to `wait()` for the child processes to `exit()`.
10. Once the wait is over, the parent process obtains the current time (end_time).
11. The string stored in the shared memory by the child process is now read using the previously created pointer. The string is converted back to epoch.
12. The difference between the times (end_time – start_time) is the time taken by the child process to execute.
13. Finally, since the shared memory is no longer required, it is removed using `shm_unlink()`.

What have we got?

1. The output is a program which can calculate the time taken for a command to execute.
2. Compilation time for this code took about 180ms. This is calculated using the same program.
3. This simple utility can be used to benchmark program speed and evaluate the fastest implementation using real world numbers.

What have I learnt?

I learnt how child processes are created and how they communicate with each other. Using this knowledge, I can create programs which can take full advantages of parallelization. Sorting, Averaging, Searching etc., can all be done much quicker this way.

Note: `-lrt` is an option for `gcc` which tells it to link the libraries `librt.a` and `librt.so` to the output file. These libraries are called `Real-time extension libraries`.