Operating Systems Lab (CS39002)

Assignment 3 Report

Suhas Jain (19CS30048)

Parth Jindal (19CS30033)

Task 1 (b)

Results

While testing on our system with 8 GB RAM and a i5 processor fork failed while multiplying matrices of size more than 139 * 139. So the the limit observed in number of forks which can be executed was around 19,000-19,500. There will also be some other processes running along with our code so the actual value of total processes running on the system will be a bit higher than this. Number of processes running in our system currently can be determined by the command mentioned below.

```
ps -A --no-header | wc -l
```

System Parameters as Limiting Factors

Linux has a setting for maximum threads per process, which specifies the maximum number of simultaneous executions that the process can handle. Changes to this can throttle the process and minimize latencies for the executions that happen. Reaching this limit means that the process needs that many threads at peak load. But, as long as it can serve requests in a timely manner, the process is adequately tuned. However, when the limit is reached, threads queue up, potentially overloading the process. At this point, the process defers creating new threads until the number of active threads drops below the limit.
Because of these system imposed limits the `fork()` system call stops working after a certain iterations. When we do some error handling and check the relevant man pages we find any the following system imposed limits could be the ones affecting this value.
`fork()` throws an error flag `EAGAIN` which indicates that there is a system imposed limit on the maximum number of forks that can be executed, that limit can be due to:

- The PID limit for a specific user group (`pids.max`) as specified in `/sys/fs/cgroup/pids/user.slice/user-1000.slice/pids.max`. In our system this limit was initially set to 20,605, which is very close to the number we obtained (19,000-19,500). Also, when we change this limit manually (as shown below) the maximum number of allowed forks increases.

  ```
  echo value > /sys/fs/cgroup/pids/user.slice/user-1000.slice/pids.max
  ```

- The soft resource limit which is indicated by `RLIMIT_NPROC` is also a limiting factor for the number of processes that can be run by a real user ID. In our system this was initially set to 31,221. When we increase the value of `pids.max` (mentioned in previous point) beyond

the value of `RLIMIT_NPROC` the number of allowed forks stop increasing so that clearly indicates that this is also a limiting factor. This is also a configurable value (can be set via `setrlimit(2)`).

- The kernel's system wide limit on the number of processes or threads which is stored in `/proc/sys/kernel/threads-max`. Initially this was set to 62,442. This is also a configurable value (can be set as mentioned below).

```
echo value > /proc/sys/kernel/threads-max
```

- Maximum number of PIDs as mentioned in (/proc/sys/kernal/pid_max). Initial value is set to 41,94,304. This value is already very high so will likely not act as any sort of bottleneck, still it is configurable and can be increased.

```
echo value > /proc/sys/kernal/pid_max
```

Any of the above 4 parameters can act as a bottleneck for maximum number of `fork()` that we can call so we can take a minimum of these parameters to get the maximum number of `fork()` that we can call. We are excluding the last parameter from the formula as its value is already very high and is unlikely to act as a limiting factor. In our matrix multiplication we need $r_1 * c_2$ number of processes so this value should not cross the theoretical maximum.

$$\mathbf{r_1 * c_2 \leq min(pids.max, RLIMIT\_NPROC, threads - max)}$$

Virtual Memory as Limiting Factor

Till now we discussed system parameters and we can increase them indefinitely, however the OS and the memory likely become the limiting factors well before that. The limit on the number of threads a process can have is calculated using the formula:

Number of Threads = Total Virtual Memory/Stack Size

Thus, the number of threads per process can be increased by increasing total virtual memory. The amount of stack size per thread is more likely to be the limit than anything else. Reducing the per-thread stack size is also a way to increase the total number of threads.
We can check the stack size per thread with ulimit:

```
ulimit -a | grep "stack size"
```

The value signifies that each of the threads will get this amount of memory assigned for its stack. On a 64-bit processor, we can adjust the stack size per thread with ulimit (here 8192 is new stack size in kbytes):

```
ulimit -s 8192
```

To reduce the effect of this bottleneck Linux follows an intelligent resource management technique called Copy on Write or simply COW is a resource management technique. Using this technique various child processes can refer to the same page entry which minimises the use of the virtual memory by a large extent. Because of these techniques the number of threads that we get from the above formula might be inaccurate however we do need to remember that such a limit exists.