ASSIGNMENT 3 REPORT

Nakul Aggarwal 19CS10044
Hritaban Ghosh 19CS30053

LIMIT ON NUMBER OF PROCESSES

/proc/sys/kernel/pid_max This file specifies the value at which PIDs wrap around (i.e., the value in this file is one greater than the maximum PID). PIDs greater than this value are not allocated; thus, the value in this file also acts as a system-wide limit on the total number of processes and threads.

The value of **PID_MAX_LIMIT** obtained through the command **cat /proc/sys/kernel/pid_max** is **4194304**.

This is the maximum number of processes that can run on the machine.

Since **init** is a process itself. It will account for one PID. Leaving us with (less than) 4194303 PIDs to work with.
Since we cannot make a sweeping generalization about the number of kernel processes already present in the memory, we will try to analyze the best case.
Even if ours (process corresponding to the program **Ass3_32_19CS10044_19CS30053_Task_1a.c**) was the only user process in the system, this number still poses a limit on the number of child processes the parent process can fork.

Reserving one PID for the parent process, we are left with 4194302 PIDs in the best case, and hence at max 4194302 child processes can be forked.

In the matrix multiplication program in the assignment, the parent process forks r1*c2 child processes.
Therefore $r1*c2 \leq 4194302$.

Therefore the number of elements in the resultant matrix is limited by 4194302.

BUT, this is not a good estimate of the number of processes that can be forked by the user because there are several other systems processes running such as the Graphical User Interface (GUI). Therefore, we need to take those into account as well and for that we can use the following ways.

SOFT LIMIT ON NUMBER OF USER PROCESSES
The following command shows the various soft limits (user-defined) on my account, obtained by running the command **ulimit -a -S**.

```
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
.
.
.
max user processes              (-u) 31204
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

Since this is the limit on the maximum number of user processes, we don't need to take the kernel or any other privileged program into account.

Since the parent process will have a PID, this leaves us with 31203 PIDs.

In the matrix multiplication program in the assignment, the parent process forks r1*c2 child processes.
Therefore $r1*c2 \leq 31203$.

Therefore the number of elements in the resultant matrix is limited by 31203.

HARD LIMIT ON NUMBER OF USER PROCESSES
The following command shows the various hard limits (admin-defined) on my system, obtained by running the command **ulimit -a -H**.

```
core file size          (blocks, -c) unlimited
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
.
.
.
max user processes              (-u) 31204
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

Since this is the limit on the maximum number of user processes, we don’t need to take the kernel or any other privileged program into account.

Since the parent process will have a PID, this leaves us with 31203 PIDs.

In the matrix multiplication program in the assignment, the parent process forks r1*c2 child processes.
Therefore r1*c2 ≤ 31203.

Therefore the number of elements in the resultant matrix is limited by 31203.

NOTE: On our Linux system the hard limit happens to be equal to the soft limit but this need not be the case in your system.

USING A C PROGRAM TO OBTAIN THE LIMIT ON NUMBER OF USER PROCESSES

```
#include <stdio.h>
#include <sys/resource.h>
int main()
{
     struct rlimit rl;
     getrlimit(RLIMIT_NPROC, &rl);
     printf("%d\n", rl.rlim_cur);
}
```

where struct rlimit is

```
struct rlimit
{
     rlim_t rlim_cur; /* Soft limit */
     rlim_t rlim_max; /* Hard limit (ceiling for rlim_cur) */
};
```

RLIMIT_NPROC The maximum number of processes (or, more precisely on Linux, threads) that can be created for the real user ID of the calling process. Upon encountering this limit, fork(2) fails with the error EAGAIN.

Since this is the limit on the maximum number of user processes, we don’t need to take the kernel or any other privileged program into account.

Since the parent process will have a PID, this leaves us with 31203 PIDs.

In the matrix multiplication program in the assignment, the parent process forks r1*c2 child processes.
Therefore $r1 \cdot c2 \leq 31203$.

Therefore the number of elements in the resultant matrix is limited by 31203.