Mattia Danese
Homework #1
CS118 - Cloud Computing
Professor Sambasivan

Question 1

a. Four processes can run without any of them incurring context switching overhead because each process can use one unique set of general purpose registers, stack-pointer registers, PC registers, and TLB caches. Since each set is unique, it is implied that each set is independent of the other sets. Therefore, no set will issue a lock to protect access to shared data or issue a block when writing or reading files as actions of one set will not affect the state of the other three sets. Thus, the criteria for context switching will not be met if four processes run concurrently.

b. There are 517 processes currently running on my machine. Given this, duplicating registers and TLB caches is not a valid scalable way to mitigate context-switch overhead because there simply may be too many processes concurrently running for this approach to have a substantial effect. For this approach to mitigate context-switch overhead by say 20%, there would need to be a unique set of registers, TLB caches, etc. for one in every five processes. Given the amount of processes currently on my machine, that amounts to roughly 100 unique sets which is a 5000% increase of the amount of unique sets my machine currently has. Clearly this is not scalable for substantially bigger distributed systems.

Question 2

a. It takes 11 clock ticks for both processes to run.

```
~/Desktop/CS 118 🧠 python process-run.py -l 4:100,1:0 -c -p
Time        PID: 0        PID: 1           CPU           IOs
  1        RUN:cpu         READY             1
  2        RUN:cpu         READY             1
  3        RUN:cpu         READY             1
  4        RUN:cpu         READY             1
  5           DONE        RUN:io             1
  6           DONE       BLOCKED                           1
  7           DONE       BLOCKED                           1
  8           DONE       BLOCKED                           1
  9           DONE       BLOCKED                           1
 10           DONE       BLOCKED                           1
 11*          DONE   RUN:io_done             1

Stats: Total Time 11
Stats: CPU Busy 6 (54.55%)
Stats: IO Busy  5 (45.45%)

~/Desktop/CS 118 🧠
```

b. Yes, swapping the two processes allows for the I/O process to start and, while it is running, yield the CPU to the CPU process. Thus, the two processes can run concurrently, resulting in the overall runtime to decrease by 4 time ticks.

```
~/Desktop/CS 118 🧠 python process-run.py -l 1:0,4:100 -c -p
Time        PID: 0        PID: 1           CPU           IOs
  1         RUN:io         READY             1
  2        BLOCKED       RUN:cpu             1             1
  3        BLOCKED       RUN:cpu             1             1
  4        BLOCKED       RUN:cpu             1             1
  5        BLOCKED       RUN:cpu             1             1
  6        BLOCKED          DONE                           1
  7*   RUN:io_done          DONE             1

Stats: Total Time 7
Stats: CPU Busy 6 (85.71%)
Stats: IO Busy  5 (71.43%)

~/Desktop/CS 118 🧠
```

c. Kernels context switch to another thread or process when the currently running thread or process issues a blocking I/O in order to make concurrent progress on multiple threads or processes. Since a CPU must wait for the I/O of the current process or thread to finish before continuing further computation, it would be beneficial to switch to another process or thread and start computation on that instead of waiting for the I/O to finish. In other words, kernels context switching would also mitigate CPU idle time when the CPU is waiting for an external process, like network activity or disk access, to terminate.

Question 3

a. The worst-case amount of time the latency-sensitive process will wait before being scheduled on the CPU after the OS receives a request destined to it is 5ms. The latency-sensitive process receives requests, which in other words means it receives I/O, therefore it will be deemed a boosted process and be given priority to the CPU. However, upon the latency-sensitive process receiving a request, there may already be another process running on the CPU. In the worst case, the latency-sensitive process will receive a request as another process begins running on the CPU and this other process will continue to run on the CPU for at most 5ms.

b. The longest amount of time the latency-sensitive process must wait before being scheduled on the CPU is 20ms. Since all CPU-bound processes receive a small number of I/O's, they, including the latency-sensitive process, are all deemed boosted processes and will have priority to the CPU. It is also specified that boosted processes are executed in FIFO. Therefore, the worst case scenario is that all 4 CPU-bound processes are executed on the CPU before the latency-sensitive process and that all 4 CPU-bound processes do not receive an I/O, thus executing on the CPU for the full available time. This would result in the 4 CPU-bound processes each executing for 5ms, which amounts to 20ms before the latency-sensitive process can be executed on the CPU.

Question 4

The aspects of a physical computer's resources that are virtualized by traditional OSes are CPU and memory. The CPU is virtualized on a per thread basis, where each thread stores its stack in the thread's virtual heap. The stack of each thread emulates a CPU by storing the thread's registers, PC, stack pointer, and general. This now enables multiple threads of a process to run concurrently as it provides the illusion of there being multiple CPUs. Memory, that is RAM, is also virtualized and this is done by mappings between virtual memory addresses and physical hard disk space. Every process' memory is kept separate and secret from that of the other processes concurrently running, making it seem as if only one process is running and using the resources of the machine, relative to the perspective of a running process. Additionally, this allows for multiple processes to run when it would otherwise not be possible due to limited RAM space. The process of virtualizing memory is composed of three layers: the hardware layer (physical hard disk and CPU of the host machine), the kernel layer (responsible for mapping virtual memory addresses to physical memory addresses), and the virtual layer (unique virtual memory used and seen by a particular process).