CS347: Operating Systems

Problem Set 1: Solutions

1. (a) The CPU switches to kernel mode, switches to the kernel stack of the process, and pushes some registers like the EIP onto the kernel stack.
   (b) The kernel pushes a few other registers, updates segment registers, and starts executing the system call code, which eventually causes P1 to block.
   (c) P2 saves user context, switches to kernel mode, services the disk interrupt that unblocks P1, and resumes its execution.
   (d) Ready / runnable.
2. (a) It contains a trap frame, followed by the context structure.
   (b) The parent's kernel stack has only a trap frame, since it is still running and has not been context switched out. Further, the value of the EAX register in the two trap frames is different.
   (c) The EIP value points to the same logical address, but to different physical addresses, as the parent and child have different memory images.
   (d) With copy-on-write, the physical addresses may be the same as well, as long as both parent and child have not modified anything.
   (e) Starts at forkret, followed by trapret. Pops the trapframe and starts executing at the instruction right after fork.
   (f) By changing the value of EAX in the trap frames.
3. (a) Yes, two processes can run the same program.
   (b) No in general. Only time this is possible is with copy-on-write during fork, and before any writes have been made.
4. (a) A blocking system call.
   (b) Timer interrupt.
5. (a) Yes, so that they can have separate execution state, and run independently.
   (b) No, threads share the program executable and data.
6. Yes, by time-sharing the CPU between threads on a single core.
7. No, not necessary, if the process is only lightly loading the system.
8. When $M < N$ and the workload to the server is CPU-bound.
9. (a) No, it will be adopted by init.

(b) Yes.

10. (a) 5. The value is only changed in the parent.
    (b) Yes, the file is only closed in the parent.

11. A blocking socket is better, because a non-blocking socket will waste time repeatedly polling.

12. First threads reads count into CPU register and increments. CPU switches to second thread. Second thread reads old value of count and increments. Both threads then write a value of 2 one after the other.

13. A,D