Answers to the questions

1.

P1	P2	P3	P4	P4	P4	P4	P4	P3	P3	P3	P3	P3	P3	P3	P3	P2	P2	P2	P1	P1	P1	P1	P1	P1	P1
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

Average waiting time = (18 + 14 + 5 + 0) / 4 = 37/4 = 9.25

2.

In a real system the scheduler and dispatcher have complete and immediate control over processes and threads. If an interrupt or exception occurs and a process should no longer be running it is stopped immediately. If it should never run again it is removed from any scheduling queues and has its resources returned to the system, then it effectively has disappeared.

With Python threads there is no thread kill method and you really should ask a thread nicely to kill itself as in the assignment code. A thread kill method is viewed as dangerous because it might be partway through modifying a shared data structure which could be left in an inconsistent state.

Alternatively some people might have listed one of the many low-level ways of killing a Python thread as given on Stack Overflow for example <http://stackoverflow.com/questions/323972/is-there-any-way-to-kill-a-thread-in-python>. Using the underlying pthread at the level of C will accomplish this but as comments point out, this is generally not a good idea.

3.

No. As the number of processors (CPUs) increases the contention to lock and unlock the dispatcher data structures grows. Normally in this situation it would be best to have one stack per CPU and employ some form of load balancing.