ANSWER 1.
(The algorithm implied that j and i are neighbors – this was not clarified in the question, but explained to those who asked)

Below is a scenario in which the program does not terminate.
Assume the initial state is (1,1,0). The guards of P0 and P1 are now true, so assume that P0 was chosen to execute, and the new configuration is (0,1,0). Now the guards of P0 and P2 are enabled, so the scheduler, which is weakly fair, can choose P0 again for execution, leading to (1,1,0). The guards of P0 and P1 are again true, so P0 can be chosen again – the guards of P1 and P2 change between true and false, so because they do not remain true after they become enabled there is no guarantee that they will be chosen for execution (weakly fair scheduler’s definition). Therefore, the program will not terminate – P0 will keep making moves.

ANSWER 2.

In the previous problem, the algorithm doesn’t terminate as a process (node) switches its state/color from 0 to 1 and vice-versa. But in this problem, a process will flip its color only if its parent has the same color. Here is an inductive proof that the algorithm will terminate.

**Base case.** The color of the root is stable since it has no parent.

**Inductive step.** Assume that a node attains a stable color. Then each of its children will attain a stable color in at most one more step

This proves that each node will attain a stable color and the algorithm will terminate.  

This will happen with both a weakly fair (and obviously a strongly fair) scheduler. In fact it will terminate even if the scheduler is unfair.

ANSWER 3.

Assumption 1. Message numbers are unbounded
Assumption 2. Message source and destination are two different processes.
We present the algorithm where we show the activities of a process $i$ when it receives a message $M[k]$ (i.e. message with a sequence number $k$). Initially $k=0$. Each process has a local variable $seq$ that is initialized to 0. Let the sender send out a message $M[k]$, after which it increments $k$ and sends the next message $M[k+1]$.

\[
\{\text{For process } i:\}
\]

\[
\text{do } \quad i = M[k].\text{destination_id} \quad v[i] := M[k].\text{value}
\]

\[
\text{\quad } \text{\quad } i \neq M[k].\text{destination_id} \& \text{ Seq } = k \quad \text{forward } M[k] \text{ along every outgoing edge;}
\]

\[
\text{\quad } \text{\quad } \text{seq} := \text{seq} + 1
\]

\[
\text{\quad } \text{\quad } \text{Seq} \neq k \quad \text{kill } M[k]
\]

\text{od}

Since the graph is strongly connected, eventually a copy of every message will reach the destination. The last action ensures that every message will be forwarded at most once, and all duplicates of $M[k]$ will be killed.

\section*{Answer 4}

\textbf{Assumption 1} Shared Memory Model: There is a global integer variable $X$.

\textbf{Assumption 2}. Every process has knowledge of $n$.

Note that the statement “between two consecutive actions by the same process every process with enabled guard executes their action at least once” is equivalent to the statement “once a process $i$ is scheduled, every other process need to be scheduled before process $i$ is scheduled again”, which in turn is equivalent to the statement “the processes are scheduled in a strict cyclic manner” (it will take action or not depends on the guards). Even if they’re scheduled ahead of their own turn with enabled guard, we need to make sure they don’t execute their action.

Following is the algorithm for process $i$:

\[
\text{do } \quad X \neq i \quad \text{skip}
\]

\[
\text{\quad } \text{\quad } X = i \& \text{ guard is enabled } \quad \text{Execute action; } X := i + 1 \text{ mod } n
\]

\[
\text{\quad } \text{\quad } X = i \& \text{ guard is not enabled } \quad X := i + 1 \text{ mod } n
\]

\text{od}

For a message based solution, one can use Lamport’s distributed mutual algorithm.
ANSWER 5

Part (a). The following solution will work with a strongly fair scheduler.

\[
\textbf{do} \quad \text{leftbook & rightbook \read; write; return books} \quad \textbf{od}
\]

The argument is that since every philosopher will find both books on the table infinitely often. The solution will not work with a weakly fair scheduler.

Part (b). Each process has a flag $f$ that is set to true when it wants to grab the books. Also, for each edge $(i, j)$ of the triangle, there is a two-valued variable $\text{turn}(i, j)$, whose value can be set to either $i$ or $j$. A solution with a weakly fair scheduler is as follows. Initially $f[i] = 0$ for every process $i$.

\[
\textbf{do} \quad \text{true} \quad \textbf{do}
\]

\[
\begin{align*}
&f[i] := \text{true}; \\
&\text{leftbook & rightbook & } \big[ j \neq i : f[j] = \text{false OR } \text{turn}(i,j) = i \big] \\
&\quad \text{read; write; return books; } \\
&\quad \big[ j \neq i : \text{turn}(i,j) := j; \\
&\quad f[i] := \text{false}
\end{align*}
\]

\[
\textbf{od}
\]

The main idea is that if a neighbor is not interested in grabbing the books, than it is no problem, but if there is a willing neighbor $j$, then the tie will be broken using the value of $\text{turn}$. 