Tuan Nguen

Question 1:
In Phos, if a function that forms the body of a process returns, then that process is not going to be executed again. Only the other processes are going to interleave. If all the processes return, then none of them are going to interleave. All of them are going to run independently, one after another.

Question 2:
Suppose that in Phos, a process is trying to send a message to itself. If the process is ready to accept the message, then nothing will happen. The process accepts the message directly, continues with the saved registers and stack pointer. No delay will happen. But if the process is not ready to accept a message, the process will not send its message, and another process might start running. If no other process is present, then the idle process will run.

Question 3:
Suppose there is a directed graph, whose nodes are the processes and there is an arrow to each process from all the other processes that are waiting to send to it. If at any point there is a cycle, the program would grind to a halt. The program would go through the cycle to find a process that is ready to receive a message. But because it is a cycle, there would be no end to that search, so the program would search forever (before it runs out of memory). A directed graph has a cycle iff when doing a DFS traversal a back edge shows up. So the avoid getting into a cycle, we can avoid forming back edges.

Question 4:
```
#include “phos.h”

static volatile itn r = 0;

void proc1(int n){
     for(int i = 0; i < 10; i++){
          serial_printf(“r = %d\n”, r);
     }
}

void proc2(int n){
     while(r < 1000000) r++;
}

void init(void){
     serial_init();
     start(USER+0, “Proc1”, proc1, 0, STACK)
     start(USER+1, “Proc2”, proc2, 0, STACK)
}
```

This is the original version of the problem. If we run this program, we will not see 10 times each integer from 0 to 1,000,000. The output is going to be random number in increasing order. This is

because every time `serial_printf` is called, we have to wait for `serial_putc` to become ready to transmit. So the program jumps to `proc2`, and increments the number until `proc1` is ready to transmit again. In the other case, when the `start` calls are re-ordered, then the output is going to be 10 times 1,000,000. The reasoning behind it, is that there is nothing that `proc2` has to wait, so it will keep incrementing `r` to 1,000,000, and then `proc1` is called.

Question 5:

a) When a character is being sent, two context switches occur. One when we need to wait for `UART_TXDRDY` to become true, and one when it the register indicates, that it is ready to transmit another character.

Question 6:

The output is garbled, because of the switching between processes. When `serial_putc` is called, we have to wait for `UART` to be ready to transmit a character, so the second `start` is called. When the second `start` is called, then `serial_putc` is again going to be called, so we will have to wait for that so the first `start` is executed. That is the reason why we see interchanging letters from both slogans. We can fix that by writing a helper function that calls `speaker`.

```
#include “phos.h”

void put_string(char *s){
            for(char *p = s; *p != ‘\0’; p++)
                serial_putc(*p);
}

static const char *slogan[] = {
            “no deal is better than a bad deal\n”,
            “BREXIT MEANS BREXIT!\n”
};

void speaker(int n){
            while(1) put_string(slogan[n]);
}

void helper(int n){
            for(int i = 0; i < 2; i++)
                speaker(i)
}

void init(void){
            serial_init();
            start(USER+0, “May”, helper, 0, STACK);
}
```