Q1: With the modified KBD driver, P0 can NOT call kgetc().
EXPLAIN: WHY?
P0 cannot call kgetc() because the process goes to sleep, and when p0 is the only process, there is no other process to give it away to, so p0 cannot go to sleep.

---------- ANSWER QUESTIONS Q2, Q3, Q4 below ----------
Given (1) to (6) in the ARM MTX code:

(1). Vector tabble at memory address 0
```
0x18: LDR PC, irq_handler_addr
irq_handler_addr: .word irq_handler
```

(2). irq_handler:
```
sub lr, lr, #4
stmfd sp!, {r0-r12, lr}
bl  IRQ_handler
ldmfd sp!, {r0-r12, pc}^
```

(3). IRQ_handler{
```
if (VIC.statusRegBit32 && SIC.statusRefBit3)
   kbd_handler();
}
```

```
int hasData = 0; // global flag set to 0 in kbd_init()
char c;         // global char shared by KBD driver and processes
```

(4). kbd_handler()
```
{
 get scancode;
 c = ASCII char mapped by scancode;
 hasData = 1;
 wakeup(&hasData);
}
```

(5). char kgetc()
```
{
 if (hasData==0)
   sleep(&hasData);
 hasData = 0;
 return c;
}
```

(6). Process_Code()
```
{
  unlock();    // allow CPU to accept IRQ interrupts
  kgetc();     // Process P1 executes this line
}
```
==========
Q2: Draw a diagram to show the control flow of the PROCESS P1

(7). Before any key is pressed:
Because there is no data, P1 is sleeping. P0 loops infinitely because ready queue is empty

(8). When a key is pressed.

As soon as key is pressed, the interrupt handler is triggered at the vector table and executed. This will wake up P! and send it to readyQueue. P0 now switches to readyQueue as it is no longer empty and return to the loop.

==========

Example:

Let's Suppose P0 forked P1 and P2 which puts P2 and P0 in readyQueue. When P1 tries to get a command line by kgets(char line[ ]); P1 would go to sleep on kgetc(), which switches to run P2, which also tries to get a command line.

(9). Verify that each process will get a complete input LINE rather than a different char of the same LINE.

Q3: EXPLAIN: HOW and WHY does (9) work?

When you press a key, interrupt wakes up both process 1 and 2, and p1 will then return c, once kgetc puts p1 back to sleep, it resumes p2. P2 then returns the same c since you have not processed another interrupt.

Q4: Interrupt handlers can only call wakeup() but should NEVER call sleep().
EXPALIN: WHY?

The processor goes into an interrupt exception state when the interrupt occurs. During this state the scheduler is disabled, If we try to sleep while in the interrupt handler, sleep will tell the scheduler to dequeue the next process, but since scheduler is disabled, the program will hang.

==========