1) Jaký je rozdíl mezi blokující a neblokující funkcí?

2) Mějme nějakou typickou funkci, která čeká na příjem zadaného množství dat z nějakého zdroje – funkce má parametr N, který určuje na kolik bytů čekáme, a všechny přijaté byty funkce vrací v návratové hodnotě jako seznam (představte si např. funkci `read` z modulu `serial`). Pokud takové funkci můžeme současně nastavit tzv. *timeout*, tak jaké byste nejspíše čekali její chování, pokud dojde k vypršení timeoutu, ale funkce zatím přijala pouze M bytů, kde M < N … ?

3) Jak bychom v programu zařídili, abychom mohli „současně“ čekat na splnění jedné ze dvou podmínek, pokud každou z podmínek kontroluje nějaká jiná funkce (tedy obě funkce jsou zcela nezávislé)? Musí být tyto funkce blokující nebo neblokující?

4) V komunikačním protokolu sériové myši z přednášky jsme viděli, že 1. byte každého paketu má 6. bit vždy nastavený na 1, kdežto 2. až 4. byte mají stejný 6. bit vždy nastavený na 0. Z jakého důvodu je nejspíše tento komunikační protokol tak navržený? Kdy bychom danou vlastnost využili?

5) Převeďte následující čísla z dvojkové do šestnáctkové soustavy:

```
0010
1001
1110
11111101
0000101000000000
01011010001110010000001011010100
11000000111111111111101110
```

6) Převeďte následující čísla z šestnáctkové do dvojkové soustavy:

```
01
23
C8
0200
8000
FEED
1DAD2BED
009A5401FF6EB007
```

7) Převeďte následující čísla z desítkové do šestnáctkové soustavy (a pro každé poznačte, kolik bytů by bylo třeba pro uložení hodnoty daného čísla):

```
10
15
16
256
250
358
1024
1023
8195
```

8) Převeďte následující čísla z šestnáctkové do desítkové soustavy (a pro každé poznačte, kolik bytů by bylo třeba pro uložení hodnoty daného čísla):

```
25
1C
FF
1423
0200
1000
FFFE
FFF0
00010000
FFFFFFFF (stačí přibližně)
```

9) Napište výsledek následujících bitových operací:

a) Zapsaných „matematicky“, kde jsou všechna čísla zapsaná ve dvojkové soustavě (***výsledky** zapište též **ve dvojkové soustavě***):

```
11101010 OR 10110001
11101010 AND 10110001
11101010 XOR 10110001
```

b) Zapsaných v jazyce Python (***výsledky** všech výrazů zapište **v šestnáctkové soustavě***):

```
0x51 | 3
0x7F02 | 0x8E18
256 | 0x00FF
0xF0005090 | 0x01F3080B
0xFFFF & 256
0x1234 & 0x0200
0xC9815093 & 0x00004000
0xC9815093 & 0xFFFEFFFF
0xC9815093 ^ 0xFF000000
```

10) Napište v Pythonu kód, který vezme proměnnou `data`, ve které je uložené celé číslo, a provede s ní následující operaci:
a) Nastaví 0., 4., a 5. bit na 1.
b) Vymaže 1., 4., 5., a 6. bit na 0.
c) Převrátí 8. až 10. bit.
d) Nastaví bity 1, 2, 3, vymaže bity 4, 7, a převrátí bity 0, 5.
e) Pokud jsou v `data` bity 0, 1, 2 a 7 současně nastavené na 1, tak vypíše `ANO`, jinak vypíše `NE`.
f) Pokud jsou v `data` bity 8, 9, 10 a 11 současně nastavené na 0, tak vypíše `ANO`, jinak vypíše `NE`.
g) Pokud je v `data` alespoň jeden z bitů 0, 1, 3 nastaven na 0, tak vypíše `ANO`, jinak vypíše `NE`.

Sada programovacích úloh – společné poznámky: Projekt `PySerialMouseComm-SimulatedMicrosoftWheelMouse-ASSIGNMENT` odpovídá přesně stavu čtení paketu sériové myši využívající komunikační protokol „***Microsoft wheel mode**“* tak, jak jsme si ho ukázali na 3. přednášce – tj. samotné zpracování paketu je zcela stejné, jako bylo v projektu `03-PySerialMouseComm` z přednášky. Rozdíl oproti tomuto přednáškovému příkladu je ten, že projekt `PySerialMouseComm-SimulatedMicrosoftWheelMouse-ASSIGNMENT` nevyužívá reálný Python modul `serial` a nekomunikuje s reálnou sériovou myší, ale obsahuje vlastní implementaci jak modulu `serial`, tak simulovaného řadiče RS-232 linky (řešeného stejným způsobem jako u self-assessment úloh ke 2. přednášce). Program je navíc rozšířený tak, že je v úvodní části programu v `PySerialMouseCommSimulatedMouse.py` zaznamenán seznam bytů, které by mohla vrátit reálná sériová myš. Hodnoty těchto bytů se pak v definovaných časech (vždy první položka typu `timedelta`) objevují v datovém registru simulovaného RS-232 řadiče, a tedy jsou pak vráceny funkcí `read` našeho modulu `serial`.

11) V projektu `PySerialMouseComm-SimulatedMicrosoftWheelMouse-ASSIGNMENT` za komentáři `### TODO:` v souboru `PySerialMouseCommSimulatedMouse.py` nahraďte přiřazení do proměnných `x` a `y` tak, aby následující výpisy pomocí `print` vypsaly správné hodnoty změny v X-ové a Y-ové souřadnici tak, jak odpovídají obsahu analyzovaného paketu dle komunikačního protokolu „***Serial Microsoft wheel mode***“ – viz původní kód z přednášky a přiložený dokument „`02-protocol (serial mice).txt`“. Očekává se, že pro řešení použijete základní bitové operace ze 3. přednášky, a nebudete využívat tzv. bitové posuny, které si představíme až na přednášce 4.
[Vzorové řešení viz `PySerialMouseComm-SimulatedMicrosoftWheelMouse-COMPLETE-REFERENCE-SOLUTION.`]

12) V projektu `PySerialMouseComm-SimulatedPS2StandardMouse-ASSIGNMENT` nahraďte komentář `### >>> YOUR CODE COMES HERE <<< ###` kompletním kódem, který bude z objektu `serialPort` číst data paketů myši ve formátu „***PS/2 standard mode***“, a provede kompletní analýzu paketu a zobrazení informace o stisknutých tlačítkách myši, a hodnoty změny v osách X a Y. Chování vašeho kódu by mělo odpovídat vzorovému výstupu uvedenému v komentáři v kódu. Specifikaci komunikačního protokolu „***PS/2 standard mode***“ najdete opět v přiloženém dokumentu „`02-protocol (serial mice).txt`“. Očekává se, že pro řešení použijete základní bitové operace ze 3. přednášky, a nebudete využívat tzv. bitové posuny, které si představíme až na přednášce 4.
[Vzorové řešení viz `PySerialMouseComm-SimulatedPS2StandardMouse-COMPLETE-REFERENCE-SOLUTION.`]

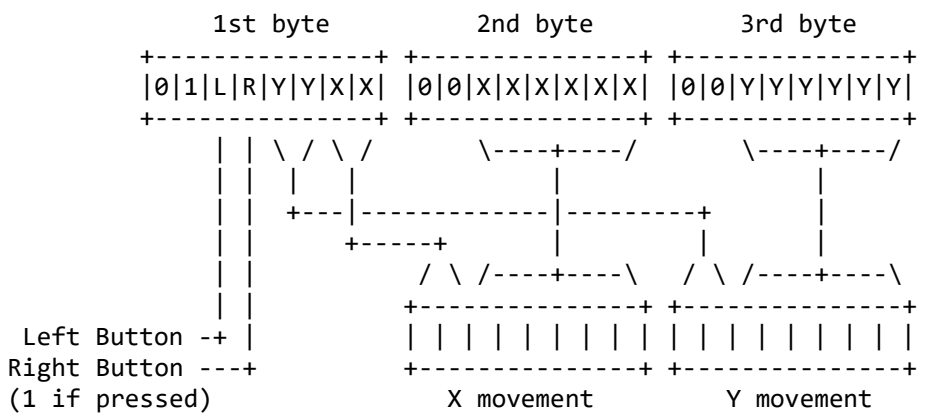
Poznámka: Cílem těchto úloh je, abyste si doma v rámci opakování látky z přednášky mohli sami ověřit, jak jste látce porozuměli. Úlohy jsou koncipovány víceméně přímočaře, a po projití poznámek a případně video záznamů z přednášek by jejich řešení mělo být zřejmé. Pro řešení úloh tedy není potřeba studium látky nad rámec probraný na přednáškách (nicméně je třeba mít i znalosti a pochopení látky z paralelně probíhajících přednášek a cvičení z předmětu Programování I). Pokud i po detailním a opakovaném projití látky z přednášek máte s řešením těchto úloh problém, tak se na nejasnosti co nejdříve ptejte na on-line konzultaci k přednášce, případně zvažte se mnou domluvit na konzultaci (zvlášť pokud tento stav u vás přetrvává i po dalších přednáškách).

Upozornění: Úlohy jsou vybrány a postaveny tak, abyste si po přednášce a před přednáškou následující mohli rychle připomenout hlavní části probrané látky a ověřit si její pochopení. Nicméně úlohy nejsou vyčerpávajícím přehledem látky z přednášek a tady rozhodně nepokrývají kompletní látku přednášek, která bude vyžadována u zkoušky. Pokud tedy u každé úlohy víte, jak by se měla řešit, tak to ještě neznamená, že jste dostatečně připraveni na zkoušku – nicméně jste jistě na velmi dobré cestě. Každopádně nezapomeňte, že na zkoušce se vyžaduje pochopení a porozumění právě všem konceptům ze všech přednášek, a navíc jsou zkouškové příklady postaveny komplexněji tak, aby ověřily také vaši schopnost přemýšlet nad látkou napříč jednotlivými přednáškami.

Příloha – obsah souboru `02-protocol (serial mice).txt` z přednášky:

```
Serial mouse reset
------------------

1: Set UART to 'break line' state (set bit 6 in the LCR).
2: Clear the RTS and DTR (bits 0-1) in the MCR, wait a while.
3: Set the RTS and DTR bits again.


Serial mouse detection (identification bytes before optional PnP data)
----------------------------------------------------------------------

In Mouse Systems mode, mouse sends nothing.
In Microsoft mode, mouse sends 'M' after dropping and raising RTS.
In Logitech mode, mouse sends 'M3' after dropping and raising RTS.
In wheel mode, mouse sends 'MZ@',0,0,0 after dropping and raising RTS.

PS/2 pointing device ID (reported after 0F2h command)
-----------------------------------------------------

In standard mode, the device reports 0.
In wheel mode, the device reports 3. This mode is enabled by sending a
  Select Report Rate 200, a Rate 100 and finally a Rate 80 command sequence.
In extended mode, the device reports 4. This mode is enabled by sending a
  Select Report Rate 200, a Rate 200 and finally a Rate 80 command sequence.


========================================================================
Serial Mouse Systems mode: 1200 bps, 8 data bits, 1 stop bit, no parity

              1st byte          2nd byte          3rd byte
         +---------------+ +---------------+ +---------------+
         |1|0|0|0|0|L|M|R| |X|X|X|X|X|X|X|X| |Y|Y|Y|Y|Y|Y|Y|Y|
         +---------------+ +---------------+ +---------------+
                   | | |     Xa movement        Ya movement
                   | | |
                   | | |       4th byte          5th byte
  Left Button ------+ | |  +---------------+ +---------------+
Middle Button --------+ |  |X|X|X|X|X|X|X|X| |Y|Y|Y|Y|Y|Y|Y|Y|
 Right Button ----------+  +---------------+ +---------------+
(0 if pressed)               Xb movement        Yb movement


Xa/Ya - movement of the mouse since last packet.
Xb/Yb - movement of the mouse since Xa/Ya.
Movement values are 8-bit signed twos complement integers.
Positive movement value indicates motion to the right/upward.


========================================================================
```

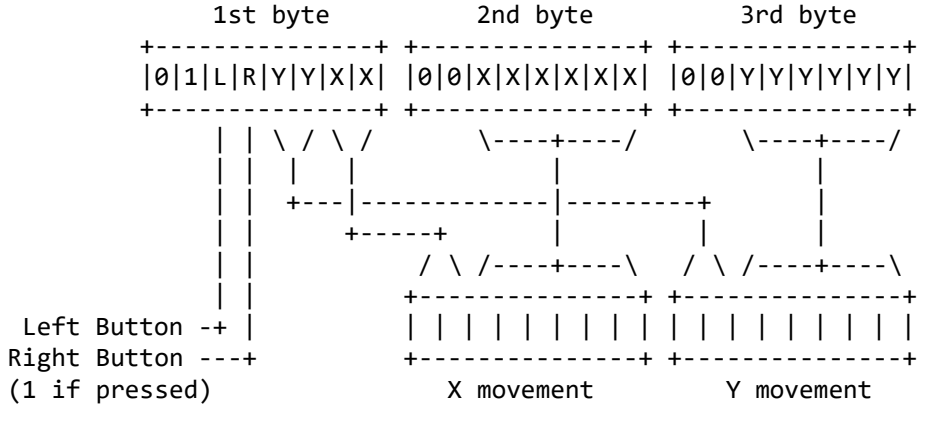
Serial Microsoft mode: 1200 bps, 7 data bits, 1 stop bit, no parity



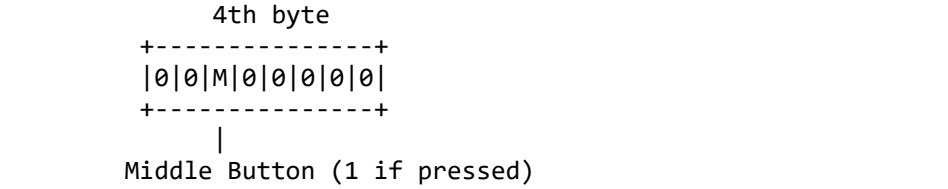
Movement values are 8-bit signed twos complement integers.
Positive movement value indicates motion to the right/downward.

==========

Serial Logitech mode: 1200 bps, 7 data bits, 1 stop bit, no parity



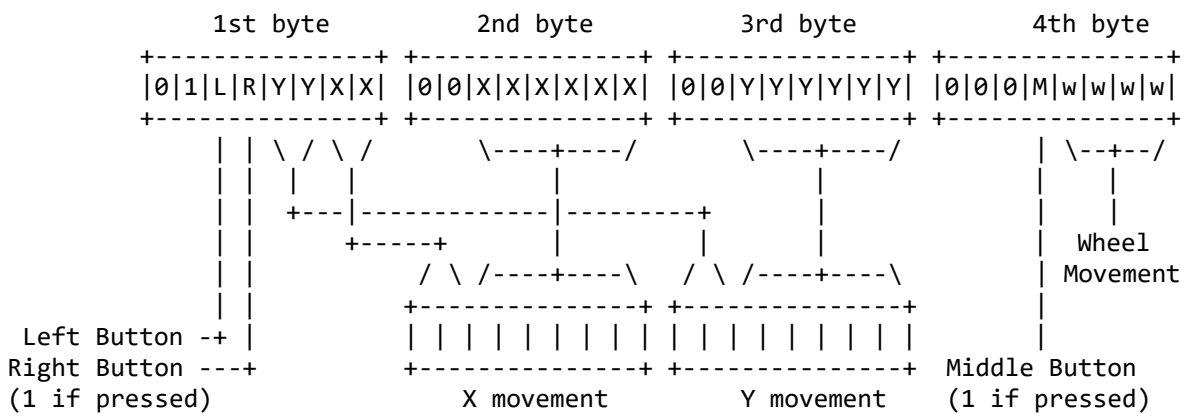
The extra byte (only when middle button is pressed)



First three bytes are equal to Mouse mode packet.
Movement values are 8-bit signed twos complement integers.
Positive movement value indicates motion to the right/downward.

==========

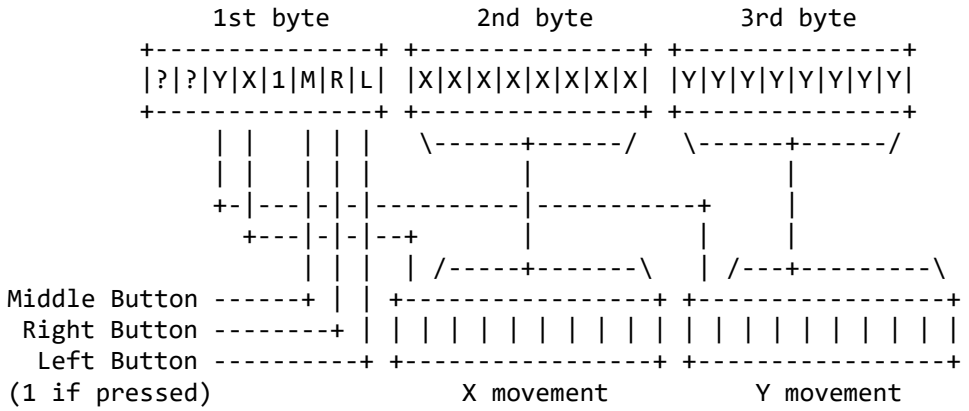
Serial Microsoft wheel mode: 1200 bps, 7 data bits, 1 stop bit, no parity



First three bytes are equal to Mouse mode packet.
Movement values are 8-bit signed twos complement integers.
Positive movement value indicates motion to the right/downward.
Wheel movement is a 4-bit signed twos complement integer.
Positive wheel movement value indicates rotation downward.

==========

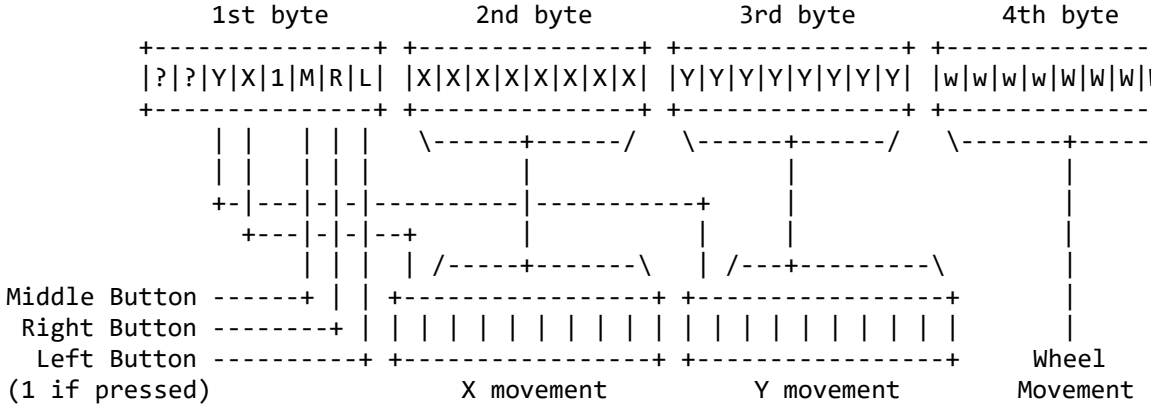
PS/2 standard mode protocol:



Two most significant bits in first byte indicate overflow (more than 9 bits
of movement) in each direction. Usually ignored.
Movement values are 9-bit signed twos complement integers.
Positive movement value indicates motion to the right/upward.

==========

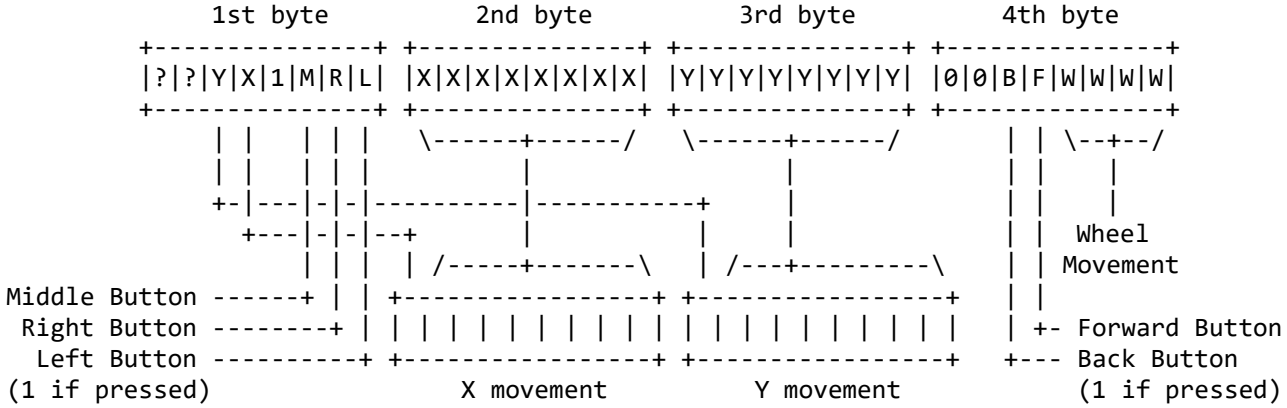
PS/2 wheel mode protocol:



First three bytes are equal to PS/2 standard mode packet.
Two most significant bits in first byte indicate overflow (more than 9 bits
of movement) in each direction. Usually ignored.
Movement values are 9-bit signed twos complement integers.
Positive movement value indicates motion to the right/upward.
Wheel movement is a 8-bit signed twos complement integer and usually
limited by -8..+7 range (4-bit value).
Positive wheel movement value indicates rotation downward.

==========

PS/2 extended mode protocol:



First three bytes are equal to PS/2 standard mode packet.
Two most significant bits in first byte indicate overflow (more than 9 bits
of movement) in each direction. Usually ignored.
Movement values are 9-bit signed twos complement integers.
Positive movement value indicates motion to the right/upward.
Wheel movement is a 4-bit signed twos complement integer.
Positive wheel movement value indicates rotation downward.