13. Óra

Miről lesz szó

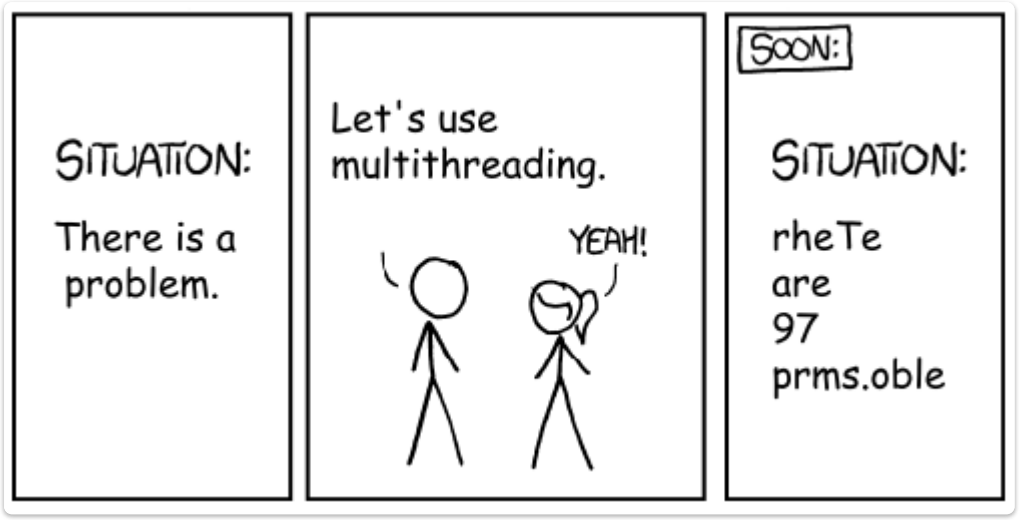
Multithreading

Cserép Máté: Többszálú programozás C#-ban, Windows Forms alkalmazások párhuzamosítása
Számítógépek párhuzamosan több feladatot is elvégezhetnek

```
static void Main(string[] args) {
    Console.WriteLine("Main thread starts");

    Thread childThread = new Thread(DoWork);
    childThread.Start(); // child thread starts
    Console.WriteLine("Main thread waiting");
    childThread.Join(); // waiting for child thread to finish

    Console.WriteLine("Main thread finishes");
}

public static void DoWork() {
    Console.WriteLine("Child thread starts");
    Console.WriteLine("Child thread goes to sleep");
    Thread.Sleep(5000); // the thread is paused for 5000 ms
    Console.WriteLine("Child thread resumes and finishes");
}
```

Milyen sorrendben fognak megjelenni a konzol outputon a két szál
által kiírt üzenetek?

Main thread starts
Child thread starts
Child thread goes to sleep
Main thread waiting
Child thread resumes and finishes
Main thread finishes
vagy akár
Main thread starts
Main thread waiting
Child thread starts
Child thread goes to sleep
Child thread resumes and finishes
Main thread finishes



Adatmegosztás a szálak közt

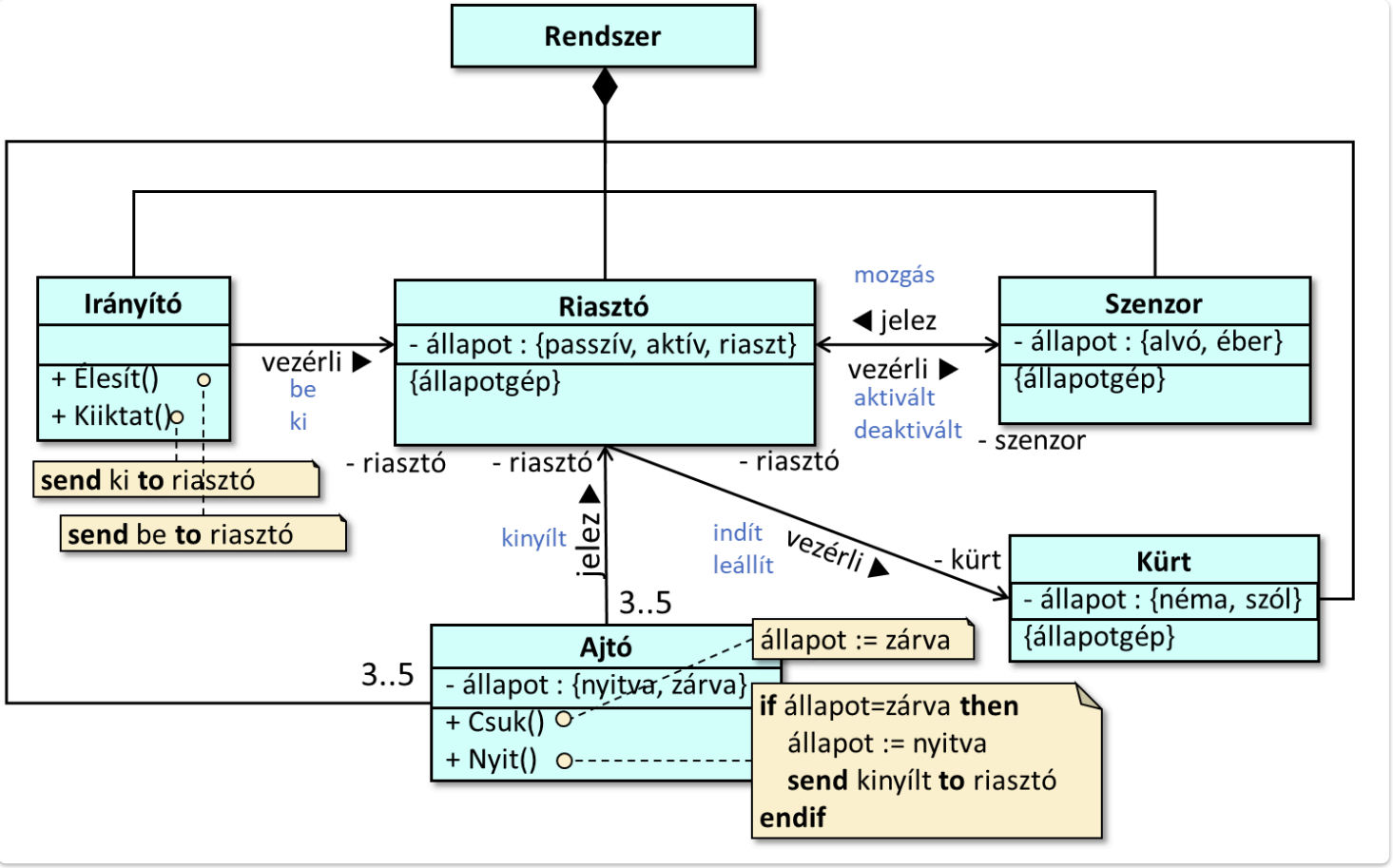
- A gyerek szál nem tud egy végeredményt visszaadni a szülő szálnak.
- A szálak közötti adatkommunikáció megosztott erőforrások (memória, közös változók) segítségével (mindkét irányban)
- A közös erőforrások kezelése *a program kritikus szakasza* (**critical section**). Azonos erőforrásra vonatkozó kritikus szakaszok párhuzamos végrehajtása (kivéve, ha minden művelet csak olvasni próbálja) hibát, nem várt futásidejű viselkedést okozhat.
- A szálakat a közös erőforrások használatakor szinkronizálni kell, kölcsönös kizárás (**mutual exclusion**) segítségével garantálva, hogy egyszerre csak egy kritikus szakasz kerül végrehajtásra.

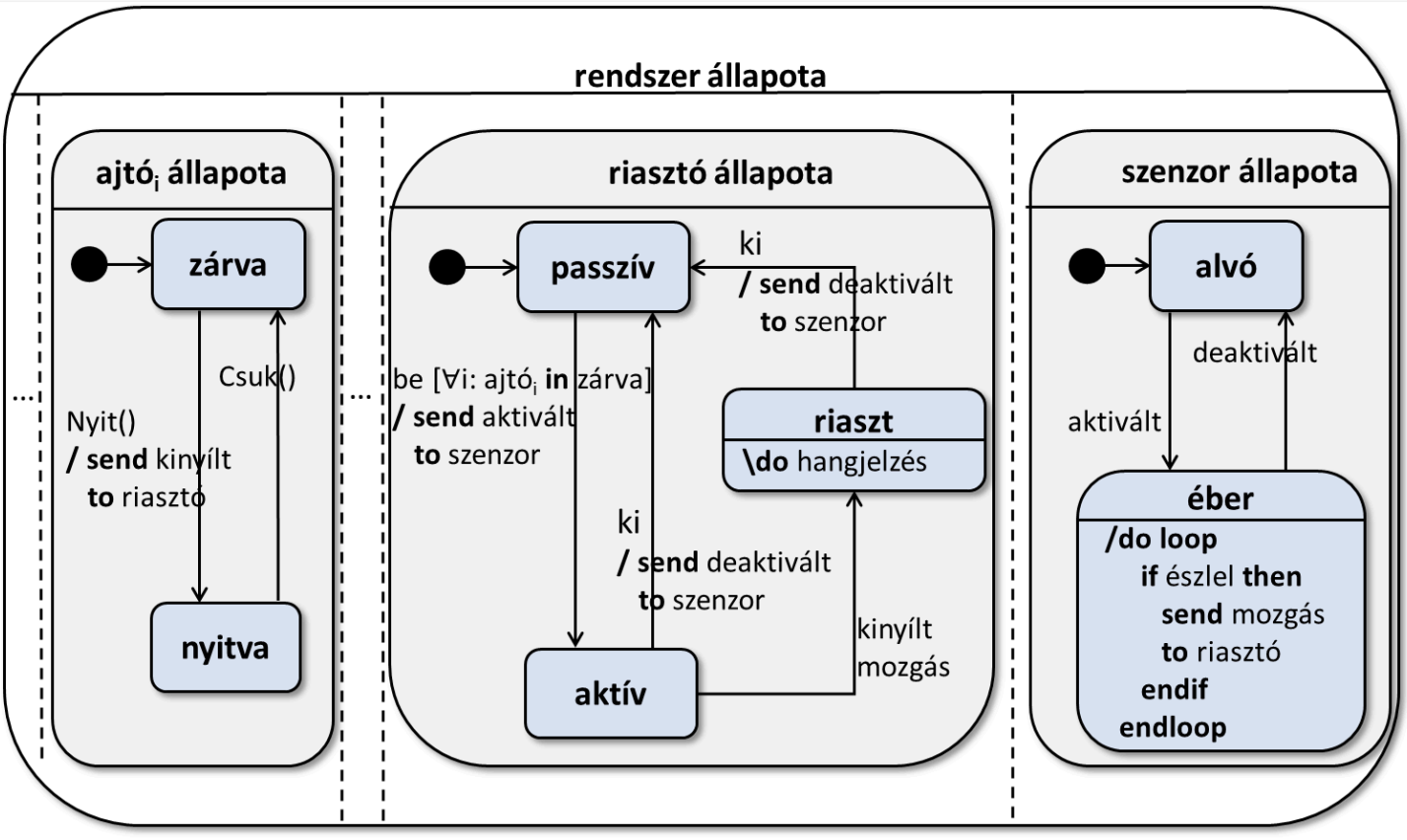
Lock utasítás használata:

```
public class SafeQueue<T> {
    private readonly Queue<T> _queue;

    public SafeQueue() {
        _queue = new Queue<T>();
    }

    public bool Empty() {
        lock (_queue) {
            return _queue.Count == 0;
        }
    }

    public void Enqueue(T e) {
        lock (_queue) {
            _queue.Enqueue(e);
        }
    }
    public T Dequeue() {
        lock (_queue) {
            try {
                return _queue.Dequeue();
            } catch {
                throw;
            }
        }
    }
}
```

Feladatok

Car alarm


Rendszer
Irányító
+ Élesít()
+ Kiiktat()
send ki to riasztó
send be to riasztó
vezérli
be
ki
Riasztó
- állapot : {passzív, aktív, riaszt}
{állapotgép}
- riasztó
- riasztó
- riasztó
mozgás
jelez
vezérli
aktivált
deaktivált
Szenzor
- állapot : {alvó, éber}
{állapotgép}
- szenzor
kinyílt
jelez
indít
leállít
vezérli
- kürt
Kürt
- állapot : {néma, szól}
{állapotgép}
3..5
3..5
Ajtó
- állapot : {nyitva, zárva}
+ Csuk()
+ Nyit()
állapot := zárva
if állapot=zárva then
állapot := nyitva
send kinyílt to riasztó
endif


rendszer állapota
ajtó$_i$ állapota
zárva
Csuk()
Nyit()
/ send kinyílt
to riasztó
nyitva
riasztó állapota
passzív
ki
/ send deaktivált
to szenzor
be [∀i: ajtó$_i$ in zárva]
/ send aktivált
to szenzor
riaszt
\do hangjelzés
ki
/ send deaktivált
to szenzor
kinyílt
mozgás
aktív
szenzor állapota
alvó
deaktivált
aktivált
éber
/do loop
if észlel then
send mozgás
to riasztó
endif
endloop