

Auta s pískem

Velmi typická úloha na přednáškách programování

- Chceme převést hromadu písku na stavenišťě,
- máme k dispozici jistý počet dělníků a jistý počet aut,
- na cestě mohou být kritické sekce (u hromady a na stavbě také),
- dělníků je omezený počet, na vybraných úsecích může být zákaz předjíždění nebo dokonce jeden jízdní pruh pro oba směry.
- Jak rozdělit dělníky a naplánovat auta tak, aby bylo převezeno co nejdříve?
- Diskrétní simulace bude simulovat konkrétní rozvrh (dělníků a aut) a tedy určí, za jak dlouho bude hromada odvezena.
- Obvykle uvažujeme jedno místo, kde je provoz sveden do jednoho jízdního pruhu pro oba směry.

Kalendář událostí

- Obsahuje přehled (popis) událostí s časem, kdy mají nastat.

Kalendář událostí

- Obsahuje přehled (popis) událostí s časem, kdy mají nastat.
- Musíme být schopni z kalendáře zjistit nejbližší událost,

Kalendář událostí

- Obsahuje přehled (popis) událostí s časem, kdy mají nastat.
- Musíme být schopni z kalendáře zjistit nejbližší událost,
- musíme být schopni události přidávat a ubírat (případně je přeplánovat).

Kalendář událostí

- Obsahuje přehled (popis) událostí s časem, kdy mají nastat.
- Musíme být schopni z kalendáře zjistit nejbližší událost,
- musíme být schopni události přidávat a ubírat (případně je přeplánovat).
- Simulační jádro z kalendáře vytáhne první událost (tedy tu, která nastane nejdříve ze všech),

Kalendář událostí

- Obsahuje přehled (popis) událostí s časem, kdy mají nastat.
- Musíme být schopni z kalendáře zjistit nejbližší událost,
- musíme být schopni události přidávat a ubírat (případně je přepřítánovat).
- Simulační jádro z kalendáře vytáhne první událost (tedy tu, která nastane nejdříve ze všech),
- v rámci obsluhy události můžeme přidávat další události nebo rušit naplánované události.

Kalendář událostí

- Obsahuje přehled (popis) událostí s časem, kdy mají nastat.
- Musíme být schopni z kalendáře zjistit nejbližší událost,
- musíme být schopni události přidávat a ubírat (případně je přeplánovat).
- Simulační jádro z kalendáře vytáhne první událost (tedy tu, která nastane nejdříve ze všech),
- v rámci obsluhy události můžeme přidávat další události nebo rušit naplánované události.
- Kalendář je též typicky schopen měřit čas (určit čas simulace).

Kalendář událostí

- Obsahuje přehled (popis) událostí s časem, kdy mají nastat.
- Musíme být schopni z kalendáře zjistit nejbližší událost,
- musíme být schopni události přidávat a ubírat (případně je přeplánovat).
- Simulační jádro z kalendáře vytáhne první událost (tedy tu, která nastane nejdříve ze všech),
- v rámci obsluhy události můžeme přidávat další události nebo rušit naplánované události.
- Kalendář je též typicky schopen měřit čas (určit čas simulace).
- Simulace končí, pokud po obsluze události zůstane kalendář prázdný.

Stavy procesů

- Každý proces (v simulaci) je vždy v nějakém stavu.

Stavy procesů

- Každý proces (v simulaci) je vždy v nějakém stavu.
- Typické stavy jsou:

Stavy procesů

- Každý proces (v simulaci) je vždy v nějakém stavu.
- Typické stavy jsou:
- Běží (aktivní) – právě se obsluhuje jeho událost,

Stavy procesů

- Každý proces (v simulaci) je vždy v nějakém stavu.
- Typické stavy jsou:
- Běží (aktivní) – právě se obsluhuje jeho událost,
- naplánovaný – čeká v kalendáři událostí,

Stavy procesů

- Každý proces (v simulaci) je vždy v nějakém stavu.
- Typické stavy jsou:
 - Běží (aktivní) – právě se obsluhuje jeho událost,
 - naplánovaný – čeká v kalendáři událostí,
 - čeká (pasivní) – čeká, až ho někdo vzbudí,

Stavy procesů

- Každý proces (v simulaci) je vždy v nějakém stavu.
- Typické stavy jsou:
 - Běží (aktivní) – právě se obsluhuje jeho událost,
 - naplánovaný – čeká v kalendáři událostí,
 - čeká (pasivní) – čeká, až ho někdo vzbudí,
 - ukončený – doběhl a nebude mít další události.

Způsoby řešení situací

- Sjedou-li se auta u kritické sekce, je třeba, aby jedno počkalo.
Jak to udělat?

Způsoby řešení situací

- Sjedou-li se auta u kritické sekce, je třeba, aby jedno počkalo. Jak to udělat?
- Buďto čekající auto přejde do stavu čeká, nebo si spočítá čas, kdy projíždějící auto odjede a na tu dobu se naplánuje.

Způsoby řešení situací

- Sjedou-li se auta u kritické sekce, je třeba, aby jedno počkalo. Jak to udělat?
- Buďto čekající auto přejde do stavu čeká, nebo si spočítá čas, kdy projíždějící auto odjede a na tu dobu se naplánuje.
- V prvním případě musí auto někdo probudit,

Způsoby řešení situací

- Sjedou-li se auta u kritické sekce, je třeba, aby jedno počkalo. Jak to udělat?
- Buďto čekající auto přejde do stavu čeká, nebo si spočítá čas, kdy projíždějící auto odjede a na tu dobu se naplánuje.
- V prvním případě musí auto někdo probudit,
- ve druhém případě musí znovu zjistit, zda je kritická sekce volná.

Způsoby řešení situací

- Sjedou-li se auta u kritické sekce, je třeba, aby jedno počkalo. Jak to udělat?
- Buďto čekající auto přejde do stavu čeká, nebo si spočítá čas, kdy projíždějící auto odjede a na tu dobu se naplánuje.
- V prvním případě musí auto někdo probudit,
- ve druhém případě musí znovu zjistit, zda je kritická sekce volná.
- Race condition – aneb proč (a kdy) mohou auta nabourat?

Způsoby řešení situací II

- Co když program běží ve více procesech a mezi okamžiky, kdy jedno auto zjistí, že KS je volná a vjede do ní, se na totéž zeptá další auto.

Způsoby řešení situací II

- Co když program běží ve více procesech a mezi okamžiky, kdy jedno auto zjistí, že KS je volná a vjede do ní, se na totéž zeptá další auto.
- Toto je obecný inženýrský problém zvaný "race-condition".

Způsoby řešení situací II

- Co když program běží ve více procesech a mezi okamžiky, kdy jedno auto zjistí, že KS je volná a vjede do ní, se na totéž zeptá další auto.
- Toto je obecný inženýrský problém zvaný "race-condition".
- Co když auto zjistí, že kritická sekce je obsazena, ale než se zafrontuje, projíždějící auto ji opustí?

Způsoby řešení situací II

- Co když program běží ve více procesech a mezi okamžiky, kdy jedno auto zjistí, že KS je volná a vjede do ní, se na totéž zeptá další auto.
- Toto je obecný inženýrský problém zvaný "race-condition".
- Co když auto zjistí, že kritická sekce je obsazena, ale než se zafrontuje, projíždějící auto ji opustí?
- Pak jde o jistou variantu problému uváznutí.

Odbočka – deadlock

aneb problém uváznutí

- Problém večeřících filosofů (kteří sdílejí vidličky),

Odbočka – deadlock

aneb problém uváznutí

- Problém večeřících filosofů (kteří sdílejí vidličky),
- Problémy z této rodiny jsou obzvláště nepříjemné v distribuovaném prostředí.

Odbočka – deadlock

aneb problém uváznutí

- Problém večeřících filosofů (kteří sdílejí vidličky),
- Problémy z této rodiny jsou obzvláště nepříjemné v distribuovaném prostředí.
- Coffmanovy podmínky jsou čtyři nutné podmínky vzniku deadlocku:

Odbočka – deadlock

aneb problém uváznutí

- Problém večeřících filosofů (kteří sdílejí vidličky),
- Problémy z této rodiny jsou obzvláště nepříjemné v distribuovaném prostředí.
- Coffmanovy podmínky jsou čtyři nutné podmínky vzniku deadlocku:
 - 1 Vzájemné vyloučení (prostředek smí být používán nejvýš k procesy),

Odbočka – deadlock

aneb problém uváznutí

- Problém večeřících filosofů (kteří sdílejí vidličky),
- Problémy z této rodiny jsou obzvláště nepříjemné v distribuovaném prostředí.
- Coffmanovy podmínky jsou čtyři nutné podmínky vzniku deadlocku:
 - 1 Vzájemné vyloučení (prostředek smí být používán nejvýš k procesy),
 - 2 drž a čekej (držíme-li určité prostředky, smíme žádat o jiné),

Odbočka – deadlock

aneb problém uváznutí

- Problém večeřících filosofů (kteří sdílejí vidličky),
- Problémy z této rodiny jsou obzvláště nepříjemné v distribuovaném prostředí.
- Coffmanovy podmínky jsou čtyři nutné podmínky vzniku deadlocku:
 - 1 Vzájemné vyloučení (prostředek smí být používán nejvýš k procesy),
 - 2 drž a čekej (držíme-li určité prostředky, smíme žádat o jiné),
 - 3 neodnímatelnost (držený prostředek nám nikdo nemůže odejmout),

Odbočka – deadlock

aneb problém uváznutí

- Problém večeřících filosofů (kteří sdílejí vidličky),
- Problémy z této rodiny jsou obzvláště nepříjemné v distribuovaném prostředí.
- Coffmanovy podmínky jsou čtyři nutné podmínky vzniku deadlocku:
 - 1 Vzájemné vyloučení (prostředek smí být používán nejvýš *k* procesy),
 - 2 drž a čekej (držíme-li určité prostředky, smíme žádat o jiné),
 - 3 neodnímatelnost (držený prostředek nám nikdo nemůže odejmout),
 - 4 čekání do kruhu (může vzniknout cyklická závislost exkluzivních prostředků).

Odbočka – deadlock

aneb problém uváznutí

- Problém večeřících filosofů (kteří sdílejí vidličky),
- Problémy z této rodiny jsou obzvláště nepříjemné v distribuovaném prostředí.
- Coffmanovy podmínky jsou čtyři nutné podmínky vzniku deadlocku:
 - 1 Vzájemné vyloučení (prostředek smí být používán nejvýš k procesy),
 - 2 drž a čekej (držíme-li určité prostředky, smíme žádat o jiné),
 - 3 neodnímatelnost (držený prostředek nám nikdo nemůže odejmout),
 - 4 čekání do kruhu (může vzniknout cyklická závislost exkluzivních prostředků).
- Deadlock je možno detekovat (zjistíme čekání do kruhu), nebo mu zabránit (znegováním některé z Coffmanových podmínek).

Deadlock

je prevít

- Vzájemnému vyloučení příliš zabránit nelze, s ostatními podmínkami to jde lépe.

Deadlock

je prevít

- Vzájemnému vyloučení příliš zabránit nelze, s ostatními podmínkami to jde lépe.
- Podmínku "drž a čekej" lze znegovat tak, že žádáme vždy o vše najednou a chceme-li požádat o další prostředek, musíme všechny držené prostředky vrátit.

Deadlock

je prevít

- Vzájemnému vyloučení příliš zabránit nelze, s ostatními podmínkami to jde lépe.
- Podmínku "drž a čekej" lze znegovat tak, že žádáme vždy o vše najednou a chceme-li požádat o další prostředek, musíme všechny držené prostředky vrátit.
- Neodnematelnost je také problematická (nicméně lze navrhnout interface, kterým nám operační systém sdělí, že nám něco odebral).

Deadlock

lze vyřešit mnoha způsoby

- Čekání do kruhu lze zakázat například očíslováním prostředků a není možno žádat o prostředek s ID vyšším, než je minimální ID námi držených prostředků.

Deadlock

lze vyřešit mnoha způsoby

- Čekání do kruhu lze zakázat například očíslováním prostředků a není možno žádat o prostředek s ID vyšším, než je minimální ID námi držených prostředků.
- Zjištěný deadlock lze řešit buďto odnímáním prostředků, nebo násilím (wait/die – například jednoho účastníka deadlocku systém zabije).

Diskrétní simulace

zpátky na stromy - totiž k autu s pískem

Implementace I:

- Kalendář událostí obousměrným cyklickým spojovým seznamem,

Diskrétní simulace

zpátky na stromy - totiž k autu s pískem

Implementace I:

- Kalendář událostí obousměrným cyklickým spojovým seznamem,
- plánujeme procesy (o jakou událost jde si pamatuje proces),

Diskrétní simulace

zpátky na stromy - totiž k autu s pískem

Implementace I:

- Kalendář událostí obousměrným cyklickým spojovým seznamem,
- plánujeme procesy (o jakou událost jde si pamatuje proces),
- čekání ve frontách (např. samoobsluha): Proces odcházející aktivuje svého následníka.

Diskrétní simulace

zpátky na stromy - totiž k autu s pískem

Implementace I:

- Kalendář událostí obousměrným cyklickým spojovým seznamem,
- plánujeme procesy (o jakou událost jde si pamatuje proces),
- čekání ve frontách (např. samoobsluha): Proces odcházející aktivuje svého následníka.
- Čekáme tedy pasívně. Výhody a nevýhody:

Diskrétní simulace

zpátky na stromy - totiž k autu s pískem

Implementace I:

- Kalendář událostí obousměrným cyklickým spojovým seznamem,
- plánujeme procesy (o jakou událost jde si pamatuje proces),
- čekání ve frontách (např. samoobsluha): Proces odcházející aktivuje svého následníka.
- Čekáme tedy pasívně. Výhody a nevýhody:
- Pasívní čekání (oproti aktivnímu zvanému busy-waiting) nezatěžuje procesor,

Diskrétní simulace

zpátky na stromy - totiž k autu s pískem

Implementace I:

- Kalendář událostí obousměrným cyklickým spojovým seznamem,
- plánujeme procesy (o jakou událost jde si pamatuje proces),
- čekání ve frontách (např. samoobsluha): Proces odcházející aktivuje svého následníka.
- Čekáme tedy pasívně. Výhody a nevýhody:
- Pasívní čekání (oproti aktivnímu zvanému busy-waiting) nezatěžuje procesor,
- Aktivní čekání: Proces se neustále ptá, jestli nemůže pokračovat, tedy méně často uvázne přehlédnut ve frontě (než se zafrontuje, skončí jediný jeho předchůdce).

Popis události

výčtové datové typy

- `enum nazev{konstanty,oddelene,carkami}`

Popis události

výčtové datové typy

- `enum nazev{konstanty, oddelene, carkami}`
- `nebo {konstanta=hodnota,`

Popis události

výčtové datové typy

- `enum nazev{konstanty,oddelene,carkami}`
- `nebo {konstanta=hodnota,.....}`
- Příklad: `enum stav{stoji,jede,vyklada};`

Popis události

výčtové datové typy

- `enum nazev{konstanty,oddelene,carkami}`
- `nebo {konstanta=hodnota,.....}`
- Příklad: `enum stav{stoji,jede,vyklada};`
- `enum stav{stoji=0,jede=1,vyklada=3};`

Popis události

výčtové datové typy

- `enum nazev{konstanty,oddelene,carkami}`
- `nebo {konstanta=hodnota,.....}`
- Příklad: `enum stav{stoji,jede,vyklada};`
- `enum stav{stoji=0,jede=1,vyklada=3};`
- Lze inkrementovat a dekrementovat (`++`, `--`).

Popis události

výčtové datové typy

- `enum nazev{konstanty,oddelene,carkami}`
- nebo `{konstanta=hodnota,.....}`
- Příklad: `enum stav{stoji,jede,vyklada};`
- `enum stav{stoji=0,jede=1,vyklada=3};`
- Lze inkrementovat a dekrementovat (`++`, `--`).
- Pozor, nikdo se nekontroluje s přetékáním (a podtékáním).

Seznamy

- Seznamy (spojové) a základní datové struktury byly probrány v Pascalu,

Seznamy

- Seznamy (spojové) a základní datové struktury byly probrány v Pascalu,
- protože v C# jsou už předimplementované:

Seznamy

- Seznamy (spojové) a základní datové struktury byly probrány v Pascalu,
- protože v C# jsou už předimplementované:
- `System.Collections.ArrayList` je univerzální seznam.

Seznamy

- Seznamy (spojové) a základní datové struktury byly probrány v Pascalu,
- protože v C# jsou už předimplementované:
- `System.Collections.ArrayList` je univerzální seznam.
- Instance této třídy jsou osazeny například metodami:

Seznamy

- Seznamy (spojové) a základní datové struktury byly probrány v Pascalu,
- protože v C# jsou už předimplementované:
- `System.Collections.ArrayList` je univerzální seznam.
- Instance této třídy jsou osazeny například metodami:
 - `Add` – přidání prvku.

Seznamy

- Seznamy (spojové) a základní datové struktury byly probrány v Pascalu,
- protože v C# jsou už předimplementované:
- `System.Collections.ArrayList` je univerzální seznam.
- Instance této třídy jsou osazeny například metodami:
 - `Add` – přidání prvku.
 - `Remove` – odebrání prvku (jednoho výskytu),

Seznamy

- Seznamy (spojové) a základní datové struktury byly probrány v Pascalu,
- protože v C# jsou už předimplementované:
- `System.Collections.ArrayList` je univerzální seznam.
- Instance této třídy jsou osazeny například metodami:
 - `Add` – přidání prvku.
 - `Remove` – odebrání prvku (jednoho výskytu),
 - `Sort` – seřídění (by default integerů, se stringy jsem měl potíže),

Seznamy

- Seznamy (spojové) a základní datové struktury byly probrány v Pascalu,
- protože v C# jsou už předimplementované:
- `System.Collections.ArrayList` je univerzální seznam.
- Instance této třídy jsou osazeny například metodami:
 - `Add` – přidání prvku.
 - `Remove` – odebrání prvku (jednoho výskytu),
 - `Sort` – seřídění (by default integerů, se stringy jsem měl potíže),
 - `IndexOf` – vyhledání prvku, při nálezů vrací nezáporné číslo, není-li prvek nalezen, vrátí se -1.

Příklad

```
using System.Collections;
ArrayList AL = new ArrayList();
AL.Add("První");
AL.Add(222);
AL.Add(100);
AL.Add(1);
AL.Add(null);
AL.Remove("První");
```

Příklad pokračování

Všechny prvky lze vypsat pomocí foreach:

```
Console.WriteLine(" Počet: 0", AL.Count );  
Console.WriteLine(" Prvni prvek: 0", AL[0]);
```

```
AL.Sort();
```

```
System.Console.Write(" Všechny prvky: ");  
foreach (object obj in AL)  
System.Console.Write(" 0", obj);  
System.Console.WriteLine();
```

Typový seznam a generika

- Často potřebujeme udělat to samé pro různé typy (například spojový seznam).

Typový seznam a generika

- Často potřebujeme udělat to samé pro různé typy (například spojový seznam).
- Víme ale předem, že prvky budou homogenní (tedy jednoho typu).

Typový seznam a generika

- Často potřebujeme udělat to samé pro různé typy (například spojový seznam).
- Víme ale předem, že prvky budou homogenní (tedy jednoho typu).
- K tomu slouží generika (v C++ šablony). Poznají se podle parametru ve špičatých závorkách:

Typový seznam a generika

- Často potřebujeme udělat to samé pro různé typy (například spojový seznam).
- Víme ale předem, že prvky budou homogenní (tedy jednoho typu).
- K tomu slouží generika (v C++ šablony). Poznají se podle parametru ve špičatých závorkách:
- `List<int> ciska=new List<int>();`

Typový seznam a generika

- Často potřebujeme udělat to samé pro různé typy (například spojový seznam).
- Víme ale předem, že prvky budou homogenní (tedy jednoho typu).
- K tomu slouží generika (v C++ šablony). Poznají se podle parametru ve špičatých závorkách:
- `List<int> cisla=new List<int>();`
- Po definici s generikem pracujeme jako s obyčejnou proměnnou.

Typový seznam a generika

- Často potřebujeme udělat to samé pro různé typy (například spojový seznam).
- Víme ale předem, že prvky budou homogenní (tedy jednoho typu).
- K tomu slouží generika (v C++ šablony). Poznají se podle parametru ve špičatých závorkách:
- `List<int> cisla=new List<int>();`
- Po definici s generikem pracujeme jako s obyčejnou proměnnou.
- `cisla.Add(10);`

Typový seznam a generika

- Často potřebujeme udělat to samé pro různé typy (například spojový seznam).
- Víme ale předem, že prvky budou homogenní (tedy jednoho typu).
- K tomu slouží generika (v C++ šablony). Poznají se podle parametru ve špičatých závorkách:
- `List<int> cisla=new List<int>();`
- Po definici s generikem pracujeme jako s obyčejnou proměnnou.
- `cisla.Add(10);`
- Dnes si ukážeme jen jak generika použít.

Generikum List

- Je nástupcem ArrayListu od C# 2.0, proto se ovládá podobně.

Generikum List

- Je nástupcem ArrayListu od C# 2.0, proto se ovládá podobně.
- ```
List<int> cela_cisla=new List<int>();
cela_cisla.Add(5);
cela_cisla.Add(3);
cela_cisla.Add(1);
foreach(int i in cela_cisla)
 Console.WriteLine(i);
Console.WriteLine("Celkem 0",cela_cisla.Count);
```

# Komplexní čísla

... jenže jenže, mně integery nestačí

```
class Komplexni
{
 public double Re,Im;
 public Komplexni(double Re,double Im)
 { this.Re=Re; this.Im=Im;}
}
List<Komplexni> s=newList<Komplexni>();
s.Add(new Komplexni(1,0));
s.Add(new Komplexni(0,1));
```

# Generická třída List

- je v `System.Collections.Generic`,
- obsahuje řadu metod, například:
- `Add`, `Contains`, `Sort`, `BinarySearch`
- Příklad:

```
List<string> s1 = new List<string>();
s1.Add("abcd");
s1.Add("efgh");
if(s1.Contains("abcd"))
 Console.WriteLine("Je tam!");
```

## 2. možnost

implementace diskrétní simulace

- Kalendář je seznam událostí `List<Udalost>`,

## 2. možnost

implementace diskrétní simulace

- Kalendář je seznam událostí `List<Udalost>`,
- jednotlivé procesy jsou postrkovány ovladači událostí,

## 2. možnost

implementace diskrétní simulace

- Kalendář je seznam událostí `List<Udalost>`,
- jednotlivé procesy jsou postrkovány ovladači událostí,
- v tom případě fronty na čekající procesy nejsou potřeba,



## 2. možnost

### implementace diskrétní simulace

- Kalendář je seznam událostí `List<Udalost>`,
- jednotlivé procesy jsou postrkovány ovladači událostí,
- v tom případě fronty na čekající procesy nejsou potřeba,
- čekání lze realizovat tak, že událost naplánujeme na nejbližší čas, kdy může nastat,

## 2. možnost

### implementace diskrétní simulace

- Kalendář je seznam událostí `List<Udalost>`,
- jednotlivé procesy jsou postrkovány ovladači událostí,
- v tom případě fronty na čekající procesy nejsou potřeba,
- čekání lze realizovat tak, že událost naplánujeme na nejbližší čas, kdy může nastat,
- musíme dát pozor na race-condition.