

Počítačová simulace

- Máme úlohu dostatečně těžkou k představení, chceme si vytvořit názor.
- Simulovat lze různé věci (úraz – tedy třeba jeho hojení, šíření alkoholu v organismu, jízdu výtahů s lidmi...).
- Počítačová simulace je simulace, při níž modelem je (počítačový) program.
- Úkolem programu je zjistit, jak se bude simulovaný systém chovat.
- Úkolem není situaci optimalizovat!
- Výsledky simulace mohou být různé, základním výsledkem je čas konce simulace.

Počítačová simulace II

- Simulace spojitá VS diskrétní,
- při spojitě simulaci je zpravidla potřeba vyřešit soustavu rovnic (nezřídka diferenciálních),
- nás bude zajímat simulace diskrétní,
- spojitou simulaci s jejím použitím lze zkusit aproximovat malými kroky a častým přepočítáváním (k tomu je ovšem potřeba stabilita řešení).

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.

Výtahy

- Podle D. Marxe (bývalého vedoucího MERL) je potřeba, aby výtahy přijely do půl minuty od zavolání,
- testy na živých lidech lezou výrobci do peněz (zákazníci utíkají),
- proto se hodí prostředí simulující požadavky lidí v domě na výtah.
- Zajímavá je distribuce dob čekání jednotlivých lidí.

Samoobsluha

- Jak v samoobsluze rozmístit zboží, aby lidé museli při běžném nákupu prolézt celý podnik (s vyhlídkou na to, že třeba koupí něco, co původně nechtěli)?
- Kolik lidí naštvoou příliš dlouhé fronty u pokladen a kolik lidí tudíž raději nakoupí jinde (a po kolika bude nutné uklízet nákupní košíky plné zboží)?
- Nakupující přecházejí po samoobsluze podle toho, co chtějí koupit,
- s množstvím vybraného zboží zpravidla roste trpělivost zákazníka.

Obecné řešení diskrétní simulace

V objektovém prostředí

- Všimneme si, že není třeba se starat o dobu, kdy nějaký proces běží,
- zajímavé je jen kdy proces začne/skončí. Proto sledujeme pouze tzv. události.
- Obvykle pro jednotlivé účastníky naprogramujeme třídy, které je nějakým způsobem reprezentují (pomocí atributů a metod),
- vyrobíme kalendář událostí (pomocí kterého se orientujeme v dění),
- vyrobíme simulační jádro schopné obsloužit konkrétní 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).
- 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.
- 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?
- 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.
- 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),
- Problémy z této rodiny jsou obzvlášť 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 neodnematelnost (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.
- 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.
- Neodnímatelnost 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.
- 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,
- 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}`
- 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).

Jak simulaci naprogramovat?

Samozřejmě v C nebo C++!

- V C++ s výhodou využijeme objekty.
- Procesy budeme simulovat pomocí objektů,
- události budou řešit metody dotyčných objektů (které se dostanou k atributům),
- kalendář událostí můžeme navrhnout pomocí předpřipravených datových struktur (a algoritmů implementovaných pro ně), kde hned využijeme šablony.



<https://www.geeksforgeeks.org/the-c-standard-template-1>

Odbočka zpět k C

pointery na funkce

```
void f(int a)
{
    printf("Funkce f s parametrem %d!\n",a);
}
void (*g)(int)=&f;// To je on!
...
(*g)(10);
g(10);// vetsinou projde taky
```