# Zadání

Několik bank provádí operace na 1 účtě: příjem peněz, výdej peněz, úročení. Každou operaci je potřeba distribuovat do ostatních bank a správně zaúčtovat - všechny operace se musí zaúčtovat ve všech bankách ve shodném pořadí, neboť např. ((10+2)\*1.05)!=((10\*1.05)+2). Pořadí zajistí sekvencer.

## Požadavky

* Implementace v C/C++.
* Překlad programem make.
* Žádné GUI, jen příkazový řádek, neinteraktivní - spustit, spočítat, zalogovat.
* OS Linux.
* Paralelizace - volitelně vlákna nebo procesy (dále v textu jen vlákna).
* Aplikace se sestává ze 2 spustitelných programů - banka a sekvencer.
* Na 1 sekvencer se může připojit více bank.
* Sekvencer obsahuje frontu, do které se přidávají přijaté operace a odebírají k rozeslání.
* Banka v náhodných intervalech vysílá operace na sekvencer.
* Generované operace ponesou informaci, zda se jedná o příjem, výdej nebo úročení a v jaké výši.
* Sekvencer bude rozesílat přijaté operace všem připojeným bankám v pořadí, ve kterém je přijal, a se sekvenčním číslem.
* Každá banka bude mít vlastní vyrovnávací paměť přijatých operací, kterou bude zpracovávat tak, aby bylo zachováno pořadí podle sekvenčního čísla - operace od sekvenceru můžou dorazit přeházené.
* Při spuštění banky nastavit konto na předem definovanou částku. Pokud už nějaká banka běží a proběhlo několik operací, je potřeba stav nové banky sesynchronizovat.
* Všechny banky po zaúčtování operace se sekvenčním číslem N budou mít shodný stav na účtě (toto není přímo požadavek, ale důsledek správné implementace).
* Ke komunikaci mezi klientem a sekvencerem využijte soketů a protokolu UDP bez potvrzování.
* Vygenerujte datový soubor pro net\_flow\_vizu.
* Pro simulaci zpoždění komunikačních linek využijte socket\_retarder.
* Změřte průměrné délky front/keší a průměrné doby setrvání zpráv v nich.
* Časová zpoždění volte tak, aby byla vizualizace dostatečně názorná.
* Příjem a vysílání zpráv musí být vhodně paralelizované, neměly by se vzájemně blokovat.
* Dokumentace jen stručně v bodech, použité algoritmy, mechanizmy, ...

# Vypracování

## Banka

Po spuštění banky se nejprve zkontrolují parametry zadané z příkazové řádky a následně se snaží program připojit na sekvencer. Po připojení se odešle operace JOIN, která upozorní sekvencer, že se připojuje nová banka. Banka obdrží od sekvenceru stejnou operaci, včetně unikátního identifikátoru banky. Tento identifikátor si uloží pro pozdější použití (např. pro logování). V odpovědi se samozřejmě nacházejí další potřebné položky, jako například číslo transakce, čas odeslání, apod. Právě číslo transakce je zde důležité, protože pokud není banka jediná přihlášená na sekvenceru, obdrží v zápětí informaci o stavu účtu včetně původního čísla transakce pro přihlášení. Tato informace o účtu je aktuální pouze pro číslo přihlašovací transakce. Nyní již banka je zařazena mezi příjemce zpráv, ale momentálně nezná aktuální stav účtu. Pokud je baTedy pokusí se dotázat na tento stav operací BALANCE. Banka tento požadavek samozřejmě dostane zpět od sekvenceru a zapamatuje si číslo transakce provádění požadavku, aby výsledek , který přijde od ostatních bank prostřednictvím sekvenceru, mohla svázat s původním dotazem. Pokud nepřijde odpověď do pěti vteřin, tak banka vyhodnotí, že je připojena na sekvenceru sama a nastaví počáteční částku účtu. Nyní se již banka nachází v synchronizovaném stavu, čímž může provádět bankovní operace. Banka dbá i na správné pořadí vykonávání operací, které je dáno číslem transakce. Toho je dosaženo použitím prioritní fronty.

## Sekvencer

Na implementaci samotného sekvenceru prakticky nic zajímavého není. Když na sekvencer přijde požadavek, je zařazen do fronty přijímacím vláknem. Odesílací vlákno jej následně z fronty vyzvedne očísluje pomocí čísla transakce a odešle na do všech připojených bank.

# Závěr