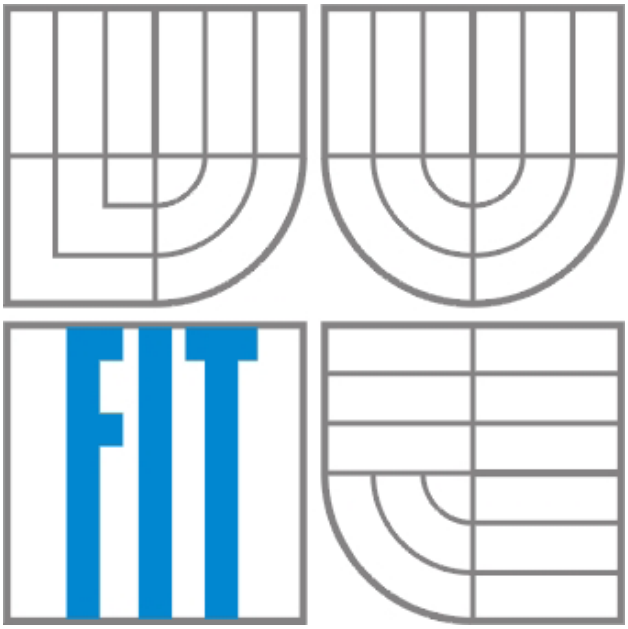
Fakulta informačních technologií, Vysoké učení technické v Brně



Dokumentácia projektu predmetu PRL

Projekt č.2 :

„Enumeration Sort“

27. Marec 2011

Autor: Matúš Kontra, xkontr00

1. Zadanie

Zadaním projektu bolo uskutočniť implementáciu algoritmu radenia „Enumeration Sort“ na lineárnej topológii procesorov so spoločnou zbernicou. Implemetácia mala byť uskutočnená s využitím knižnice OpenMPI (Open Message Passing Interface). Súčasťou zadania je aj táto dokumentácia a testovací unixový shell skript, zodpovedný za preklad zdrojových súborov, vytvorenie testovacích dát, spustenie programu, a následné vyčistenie.

2. Analýza a princíp fungovania algoritmu

Z teoretického uhla pohľadu pracuje algoritmus nad lineárnov topológiou n procesorov (, kde n je rovné počtu radených prvkov) so spoločnou zbernicou, ktorá je schopná rozdistribuovať v jednom kroku hodnotu na ňu umiestnenú všetkým procesorom. Každý z týchto procesorov má 4 registre:

- X_i – prvkok x_i
- Y_i – postupne prvky $x_1..x_n$
- C_i – pozícia prvku X_i vo výslednej postupnosti
- Z_i – prvok na správnej pozícii

Algoritmus následne pracuje podľa tohto pseudokódu:

1. Všetky procesory paralelne nastavia svoju hodnotu C na 1
2. Pre k v rozsahu 1 až 2*n vykoná každý procesor tieto akcie:
    a. Ak platí, že X a Y sú neprázdne a zároveň platí X > Y, inkrementuj C
    b. Posuň svoju hodnotu Y procesoru ďalej napravo v topológii a príjmi novú hodnotu Y od procesora naľavo v topológii
    c. Pokiaľ nie je vyčerpaný vstup vloží sa na zbernicu nový prvok, ktorý je načítaný do Y_1 a X_k
    d. Ak už na vstupe nie sú ďalšie hodnoty (k > n) nastav hodnotu Z procesora s indexom C_{k-n} na hodnotu X_{k-n}
3. Pre k v rozsahu 1 až n vykonaj:
    a. Z hodnota registra n sa pošle na výstup
    b. Všetky procesory posunú svoju hodnotu Z o jeden index doprava

Algoritmus pracuje v lineárnom čase. Overenie:

- Krok 1. sa vykoná v konštantnom čase c
- Krok 2. sa vykoná 2n krát
- Krok 3. a vykoná n krát

Platí: $3n + c = O(n)$, algoritmus pracuje v lineárnom čase. Cena paralelného riešenia je $O(n^2)$. Fakt, že algoritmus nie je optimálny a taktiež nie je schopný radiť postupnosti obsahujúce rovnaké hodnoty, môžeme zaradiť k jeho hlavným nedostatkom.

3. Implementácia

Program mpirun spustí náš kód na *n+1* procesoroch, kde procesor s rank 0 neprevádza výpočet, ale vystupuje v roli mastera a riadi vstup a výstup hodnôt. Na začiatku master načíta hodnoty do pomocného poľa pre jednoduchšiu manipuláciu. Registre X,Y,Z,C sú reprezentované dátovým typom integer. Okrem týchto registrov má každý procesor premenné aj na indikáciu neprázdnosti X, Y – hasx, hasy. Význam premennej zcount bude ozrejmený v kapitole o ošetrení radenia rovnakých prvkov.

Kód vykonávajúci radenie sa veľmi málo líši od pseudokódu algoritmu. Krok 1 je implementovaný ako jednoduché priradenie do c(jeden príkaz). Iterácia cez premennú k v rozsahu 1 až 2*n kroku 2 je implementovaná ako cyklus for, v ktorom môžeme rozoznať podbloky realizujúce:

1. Porovnanie – príkazom if sa overí neprázdnosť registrov X a Y, a zároveň ich vzťah na základe čoho buď inkrementuje C alebo nie
2. Posuv hodnôt – všetky procesory s rankom 1 až min(k, n-1) odošlú volaním MPI_Send svoju hodnotu procesoru s rankom o jedna vyšším a zároveň všetky procesory okrem prvého príjmu hodnotu od ranku o jedna nižším od svojho.
3. Vloženie nových prvkov postupnosti – master zašle nový prvok procesoru 1 ktorý ho príjme do svojho Y a zároveň umiestni tento prvok na zbernicu, odkial si ho prevezme procesor k do registra X. (toto nie je implementované broadcastom, nakoľko je zrejmé kto bude prijímať)
4. Odoslanie prvku procesoru so správnim rankom – prvok, ktorý už dopočítal správnu hodnotu C pre svoje X (index prijímajúceho procesora), simuluje zaslanie dát na zbernicu volaním broadcastu s hodnotou svojho C. Procesory si uložia túto hodnotu do pomocnej premennej a porovnajú ju so svojim rankom. Procesor ktorému vyhovie daný booleovský výraz, začne očakávať hodnotu do registra Z. Túto príjme od procesora, ktorý broadcastoval. Poznámka: tento popis sa v kóde líši od toho čo sa skutočne deje, nakoľko som v tomto mieste implementoval funkčnosť algoritmu raidť postupnosti s rovnakými prvkami. Táto činnosť bude opísaná v samostanej podkapitole

Krok 3 je implementovaný podobne ako podkrok 2 kroku 2, s tým rozdielom, že posledný procesor posiela svoju hodnotu mastrovi.

3.1 Radenie ekvivalentných prvkov

Táto funkčnosť môže byť implementovaná na rôznych miestach v kóde. Nakoľko však riešenia , ktoré manipulujú so vstupmi a výstupmi, nemenia fungovanie algoritmu, problém stále pretrváva.

Procesory sú rožšírené o register ZCOUNT, udávajúci počet výskytov prvku na tejto pozícii. K zmene oproti štandardnému algoritmu dochádza v 4 podkroku, kroku 2. Tu, po tom čo broadcaster odošle svoju hodnotu C, čaká na zaslanie hodnoty ZCOUNT procesora, ktorý ma prijímať X. Po tom čo prijíme túto hodnotu, pripočíta k nej svoje C a opet broadcastuje, ale už C + ZCOUNT. Prijímajúci procesor je tentokrát, už ten správny.

Tu treba poznamenať, že interakcie medzi nesusednými procesormi, sú v teoretickej rovine riešené zbernicou, čiže stále sa pohybujeme v možnostiach našej topológie.

Daľej treba poznamenať, že nedochádza k zmene časovej náročnosti algoritmu, nakoľko zvyšujeme len počet krokov v iterácii druhého kroku.

4. Analýza zložitosti

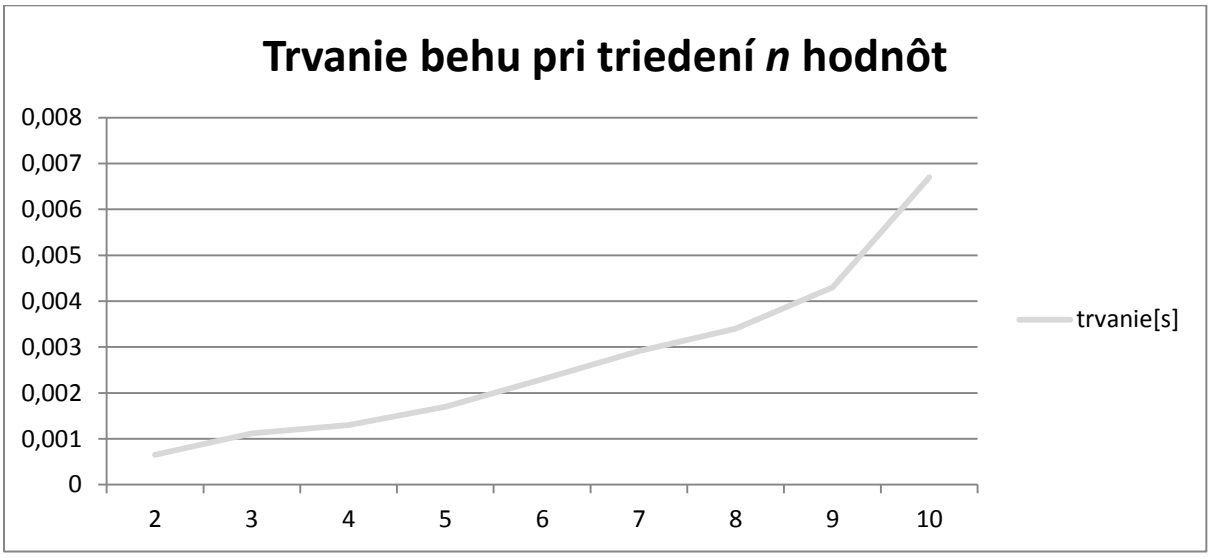
Ako bolo spomýnané jedná sa o algoritmus pracujúci v lineárnom čase, teraz však vypočítajme celkový počet krokov potrebných k výpočtu. Doba trvania jedného kroku je tkrok.

t_{krok} (krok 1) + 2*n*4* t_{krok} (krok 2) + n*2* t_{krok} (krok 3) = t_{krok} + 10*n* t_{krok} = O(n)

U upraveného algoritmu radiaceho aj postupnosti s rovnakými prvkami , sú akcie spojené s dalšími príkazmi zahrnuté v pokroku 4. Tu treba poznamenať, že i keď sa vzorec nezmenil, pravdepodobne sa posledný podkrok bude vykonávať dlhšie.

5. Dĺžka trvania

Do kódu boli pridané na miesta volania funkcii MPI_Wtime() – prvé pred začiatok vykonávania algoritmu, druhé na koniec. Boli zaznamenávané doby behu výpočtu jednotlivých procesorov, viacerých behov. Tieto boli zpriemerované a použité ako výsledok. Testovanie prebiehalo na serveri merlin, preto dochádza k zmene charakteru dát ked počet požadovaných procesorov presiahol počet 8.



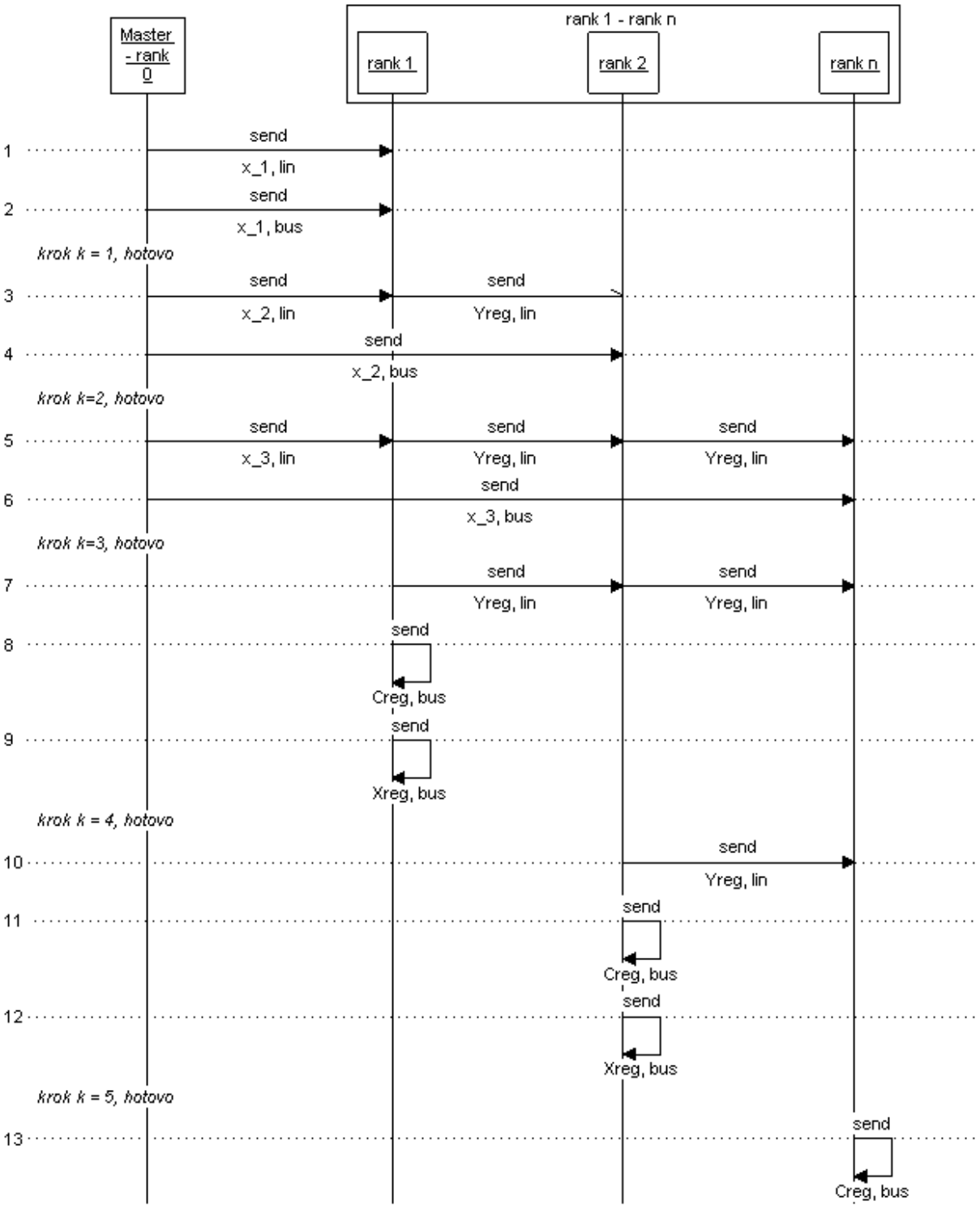
6.Sekvenčné diagrami

Vzhľadom na rozmery vygenerovaných diagramov sú tieto umiestnené na samostatných stranách za Záverom.

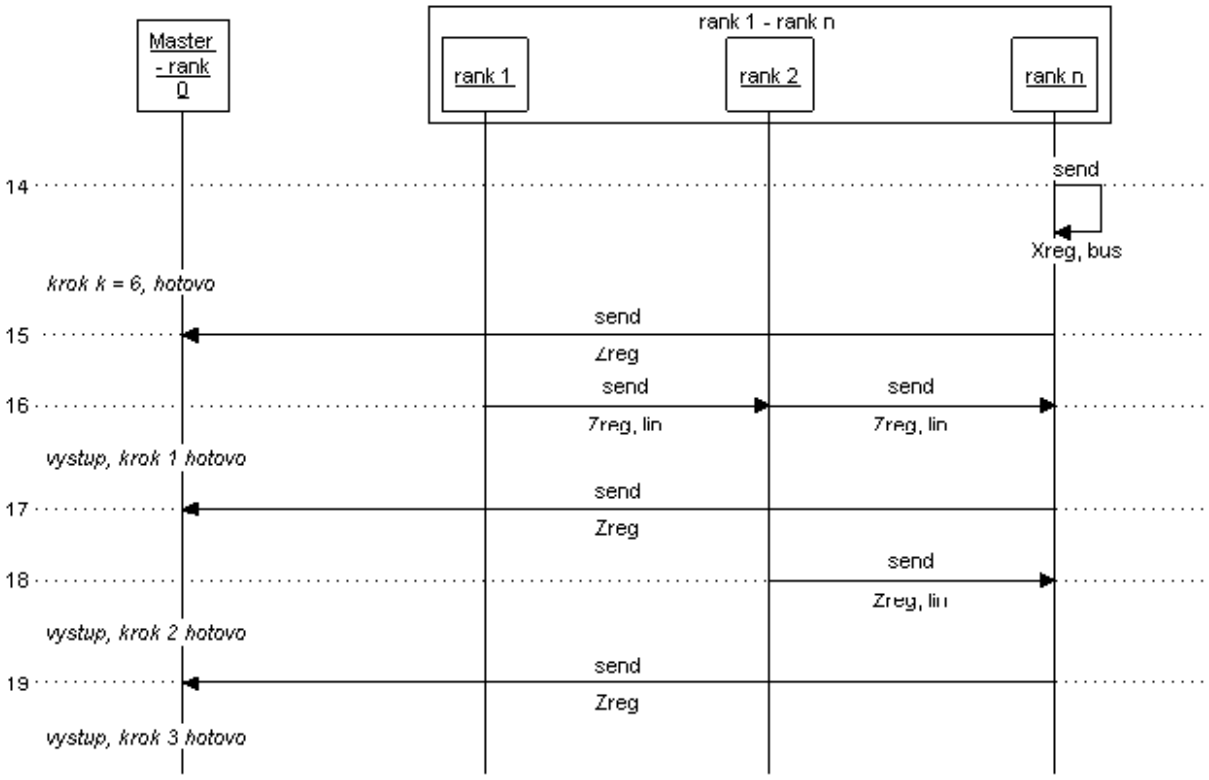
7. Záver

V projekte sme sa podrobnejšie oboznámili s technikami prograovania viacprocesorových aplikácii. Daľej sme overili, že algoritmus enumeration sort pracuje v linearnom case. Tento fakto bolo možné pozorovať, ako na grafe dĺžky výpočtu tak aj na sekvenčnom diagrame (odvoditeľné z počtu správ).

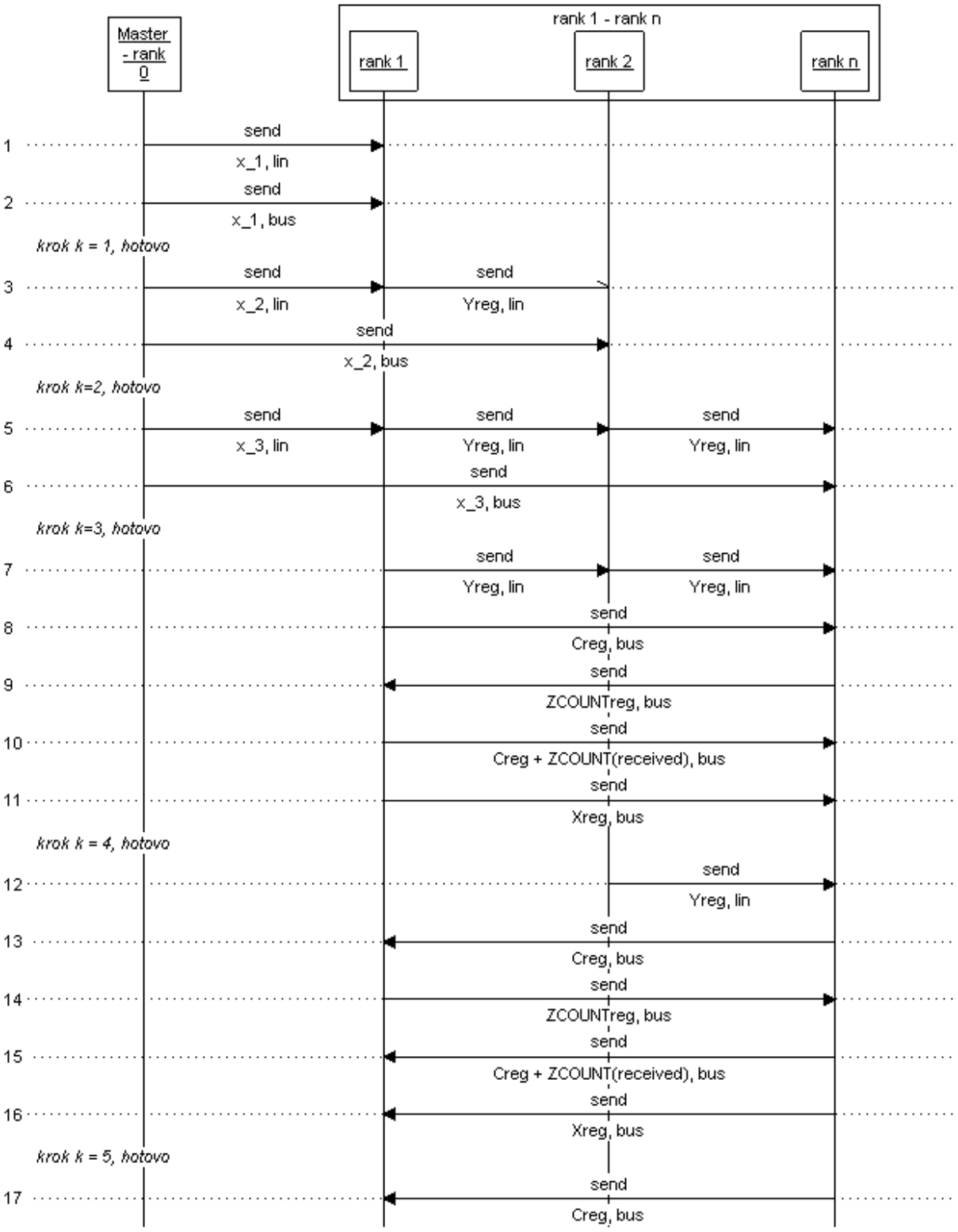
Sekvenčný diagram, n = 3, postupnosť 1,2,3, bez implementovaného radenia sekvencii s rovnakými prvkami



Sekvenčný diagram, n = 3, postupnosť 1,2,3, bez implementovaného radenia sekvencii s rovnakými prvkami(pokračovanie)



Sekvenčný diagram, n = 3, postupnosť 3,1,1, s implementovaním radením rovnakých prvkov



Sekvenčný diagram, n = 3, postupnosť 3,1,1, s implementovaním radením rovnakých prvkov(pokračovanie)