Ročníkový projekt - report - letný semester

Komunikačná knižnica pre medziprocesovú komunikáciu

Počas letného semestra bola dokončená hlavná implementačná časť projektu. Výsledkom je funkčná knižnica pre komunikáciu medzi viacerými procesmi pomocou zdieľanej pamäte a semaforov. Knižnica poskytuje jednoduché API pre inicializáciu procesov, prípravu odosielacieho bufferu, odosielanie správ, prijímanie správ, multicast a korektné ukončenie komunikácie.

Okrem samotnej knižnice boli pripravené aj benchmarky pre jednoduché meranie správania systému pri komunikácii typu all-to-all. Benchmarky merajú celkový čas behu a priemernú latenciu jednej správy pri rôznych veľkostiach správ a pri rôznom počte procesov. Výsledky sú spracované do grafov, v ktorých je zobrazená aj štandardná odchýlka merania.

Čo bolo implementované

- Dokončená bola knižnica com.c s verejným rozhraním definovaným v com.h.
- Implementované boli operácie com_initialize, com_prepare_send_buffer, com_send, com_finish_send_buffer, com_recv, com_mcast a com_finalize.
- Komunikácia je založená na POSIX shared memory segmentoch, anonymnej zdieľanej pamäti a semaforoch.
- Bol doplnený jednoduchý router proces, ktorý sprostredkuje doručovanie metadát medzi odosielateľom a prijímateľom.
- Boli vytvorené dva all-to-all benchmarky: benchmark celkového času a benchmark priemernej latencie správy.
- Do benchmarkov bolo doplnené viacnásobné meranie jednej konfigurácie a výpočet štandardnej odchýlky, ktorá sa zobrazuje v grafoch ako error bar.
- V programe boli doplnené kontroly návratových hodnôt pre výpisové funkcie printf/fprintf cez pomocné obalové funkcie.

Rozhranie knižnice

Používateľ knižnice pracuje iba s rozhraním uvedeným v hlavičkovom súbore com.h. Vnútorná práca so zdieľanou pamäťou, semaformi, routerom a potvrdeniami je ukrytá v implementácii com.c.

```
#ifndef COM_H
#define COM_H


#include <stddef.h>


void com_initialize(int nr_proc, int *rank);
void *com_prepare_send_buffer(size_t size);
void com_send(int rank, size_t size);
void com_finish_send_buffer(void);
void com_recv(void **msg, size_t *size);
```

```
void com_mcast(void *msg, size_t size);
void com_finalize(void);

#endif
```

Základné funkcie knižnice

Funkcia	Úloha
com_initialize(int nr_proc, int *rank)	Inicializuje komunikačný systém, vytvorí zdieľanú riadiacu oblasť, spustí router proces a vytvorí používateľské procesy. Každý proces dostane svoj rank.
com_prepare_send_buffer(size_t size)	Pripraví zdieľaný odosielací buffer procesu. Pred jeho opätovným použitím počká, kým všetci príjemcovia skopírujú predchádzajúcu správu.
com_send(int rank, size_t size)	Odošle správu procesu s daným rankom. Do routera sa posielajú iba metadáta: odosielateľ, cieľ, veľkosť a názov shared memory segmentu.
com_finish_send_buffer(void)	Ukončí prácu s pripraveným odosielacím bufferom a uvoľní lokálny zámok pre ďalšie použitie.
com_recv(void **msg, size_t *size)	Počká na správu v inboxe procesu, otvorí shared memory segment odosielateľa, skopíruje dáta do lokálnej pamäte a potvrdí odosielateľovi, že segment už môže byť neskôr znovu použitý.
com_mcast(void *msg, size_t size)	Odošle jednu správu všetkým ostatným procesom. Využíva ten istý odosielací segment pre všetky cieľové procesy.
com_finalize(void)	Korektne ukončí proces, počká na nedokončené potvrdenia a posledný končiaci proces pošle routeru signál na ukončenie.

Vnútorný princíp fungovania

Knižnica používa centrálnu zdieľanú riadiacu štruktúru, ktorá obsahuje počet procesov, stav ukončovania, centrálnu schránku routera a samostatný slot pre každý proces. Každý slot obsahuje inbox pre prichádzajúcu správu, semafory na synchronizáciu, informáciu o vlastnom shared memory segmente procesu a počítadlo čakajúcich potvrdení.

Pri inicializácii sa najprv vytvorí zdieľaná riadiaca oblasť pomocou mmap. Potom sa spustí samostatný router proces. Následne sa vytvoria používateľské procesy. Každý proces si vytvorí vlastný POSIX shared memory segment, ktorý používa ako odosielací buffer pre svoje správy.

Odosielanie neprebieha tak, že by sa celé dáta kopírovali cez centrálnu schránku. Odosielateľ zapíše dáta do svojho shared memory segmentu a routerovi odovzdá iba metadáta. Router tieto metadáta presunie do inboxu cieľového procesu. Prijímateľ potom otvorí shared memory segment odosielateľa, skopíruje obsah správy do vlastnej pamäte a po kopírovaní odošle potvrdenie.

Tento návrh oddeľuje smerovanie správ od samotného prenosu dát. Router pracuje iba s malými metadátami a veľké dáta ostávajú v shared memory segmente odosielateľa. To je výhodné najmä pri väčších správach, pretože sa znižuje počet zbytočných kópií v centrálnej časti systému.

Pre multicast sa používa rovnaký princíp. Jedna pripravená správa sa postupne odošle všetkým ostatným procesom. Keďže jeden shared memory segment môže používať viac prijímateľov, odosielateľ si pamätá počet čakajúcich potvrdení. Buffer môže znovu použiť až po tom, ako všetci príjemcovia skopírujú dáta.

All-to-all benchmark

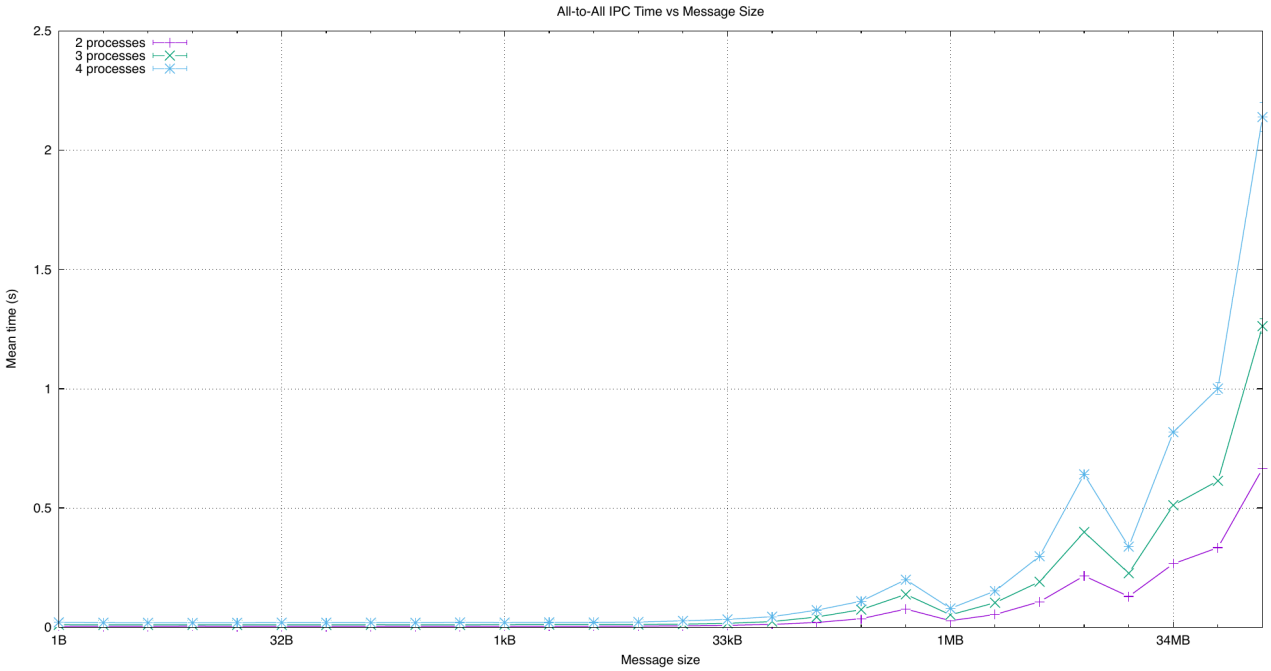
Pre overenie správania knižnice bol vytvorený jednoduchý benchmark komunikácie typu all-to-all. Pri tejto komunikácii každý proces v každom cykle pošle jednu správu každému inému procesu. Pre n procesov teda v jednom cykle vzniká n * (n - 1) správ. Benchmark sa spúšťa pre 2, 3 a 4 procesy a pre rôzne veľkosti správ.

V každom používateľskom procese bežia dve vlákna. Jedno vlákno odosiela správy ostatným procesom a druhé vlákno prijíma očakávaný počet správ. Prijaté správy sa v tomto benchmarku neukladajú do fronty, ale po prijatí a kontrole veľkosti sa hneď uvoľnia. Tým sa meranie sústredí hlavne na samotnú komunikáciu cez knižnicu.

Program vypisuje celkový čas behu meranej časti, počet správ, priemerný čas jedného cyklu, priemerný čas jednej správy a priepustnosť v MiB/s. Skripty následne tieto hodnoty spracujú do dátových súborov a grafov. Každá konfigurácia sa meria viackrát a z nameraných hodnôt sa počíta priemer a štandardná odchýlka.

Benchmark celkového času

Prvý pohľad na výkon ukazuje graf celkového času behu all-to-all komunikácie v závislosti od veľkosti správy. Os X zobrazuje veľkosť správy a os Y priemerný čas v sekundách. Graf obsahuje samostatné krivky pre 2, 3 a 4 procesy.

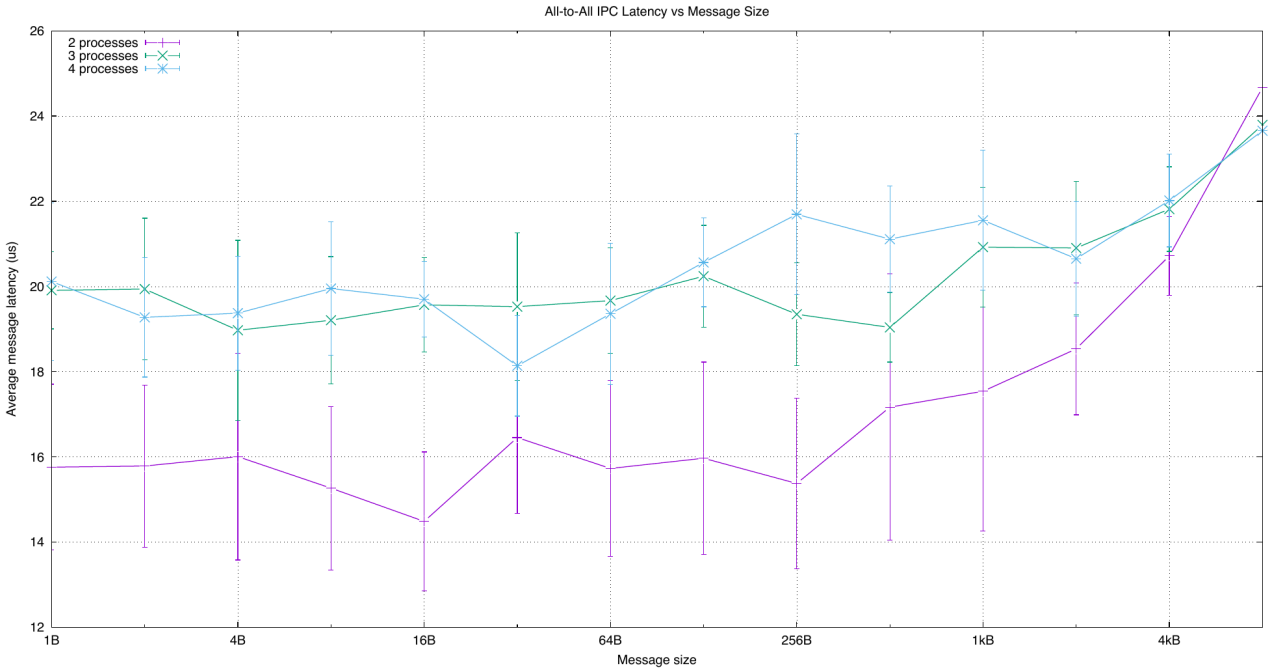


Obrázok 1: All-to-all IPC Time vs Message Size

Z grafu je vidieť, že pri malých správach je celkový čas veľmi nízky a rozdiely medzi veľkosťami správ sú malé. Pri väčších správach začne čas výrazne rásť, pretože sa kopíruje väčší objem dát. Zároveň platí, že pri väčšom počte procesov rastie aj počet odoslaných správ v jednom cykle, preto má konfigurácia so 4 procesmi najvyšší čas. Viditeľný je aj vplyv systému a plánovania procesov, preto krivka nie je úplne hladká.

Benchmark latencie správy

Druhý graf sa zameriava na priemernú latenciu jednej správy. Pre tento benchmark boli použité malé veľkosti správ, aby bol rozdiel medzi konfiguráciami lepšie viditeľný. Latencia je vypočítaná ako celkový nameraný čas vydelený celkovým počtom správ.



Obrázok 2: All-to-all IPC Latency vs Message Size

Na grafe je vidieť, že pri malých správach sa priemerná latencia pohybuje približne v desiatkach mikrosekúnd. Pri 2 procesoch je latencia nižšia ako pri 3 a 4 procesoch, čo zodpovedá menšiemu počtu komunikujúcich strán a menšej záťaži synchronizačných mechanizmov. Pri rastúcej veľkosti správy latencia postupne rastie, ale pri malých správach zohráva väčšiu úlohu réžia synchronizácie, prepínania procesov a práce routera ako samotná veľkosť dát.

Zvislé úsečky v grafoch predstavujú štandardnú odchýlku. Ukazujú, ako sa jednotlivé merania tej istej konfigurácie líšili od priemeru. Pri benchmarkoch medziprocesovej komunikácie je takáto variabilita prirodzená, pretože výsledok môže ovplyvniť plánovanie procesov operačným systémom, aktuálne zaťaženie stroja a práca so zdieľanou pamäťou.

Benchmark distribuovanej databázy

Popri jednoduchom all-to-all benchmarku bol vytvorený aj druhý benchmark, ktorý simuluje jednoduchú distribuovanú databázu. Cieľom nebolo implementovať reálnu databázu, ale vytvoriť praktickejší komunikačný scenár typu request-response, v ktorom jeden proces posiela požiadavku inému procesu a čaká na odpoveď s dátami určitej veľkosti.

Benchmark sa spúšťa s tromi parametrami: počtom procesov, počtom požiadaviek na proces a veľkosťou odpovede. Každý proces vystupuje zároveň ako klient aj ako server. Ako klient postupne posiela požiadavky ostatným procesom a ako server obsluhuje požiadavky, ktoré prídu od iných procesov.

Princíp fungovania benchmarku

V každom procese bežia tri hlavné vlákna. Compute vlákno generuje požiadavky a meria čas od odoslania požiadavky po prijatie odpovede. Receiver vlákno prijíma všetky správy z komunikačnej knižnice a podľa typu ich ďalej rozdeľuje. Request handler vlákno spracúva prijaté požiadavky a posiela späť odpovede.

Správy majú jednoduchú hlavičku, v ktorej je uložený typ správy, identifikátor požiadavky, zdrojový rank, kľúč a požadovaná veľkosť odpovede. Používajú sa typy REQUEST, RESPONSE, STOP a LOCAL_DONE.

Požiadavky sa posielajú na cieľové procesy round-robin spôsobom, pričom proces nikdy neposiela databázovú požiadavku sám sebe.

Pri spracovaní požiadavky serverová časť procesu vytvorí odpoveď s požadovanou veľkosťou. Dáta v odpovedi sú umelo vyplnené, aby benchmark neprenášal iba prázdne metadáta. Prijímacia časť pri odpovedi prečíta celý payload a vypočíta kontrolný súčet. Tým sa zabezpečí, že meranie zahŕňa aj reálne čítanie prijatých bajtov.

Na synchronizáciu medzi vláknami v rámci jedného procesu sa používa fronta požiadaviek a samostatný response slot. Fronta odovzdáva prijaté REQUEST správy handler vláknu. Response slot slúži na odovzdanie informácie o prijatej RESPONSE správe compute vláknu, ktoré na ňu čaká.

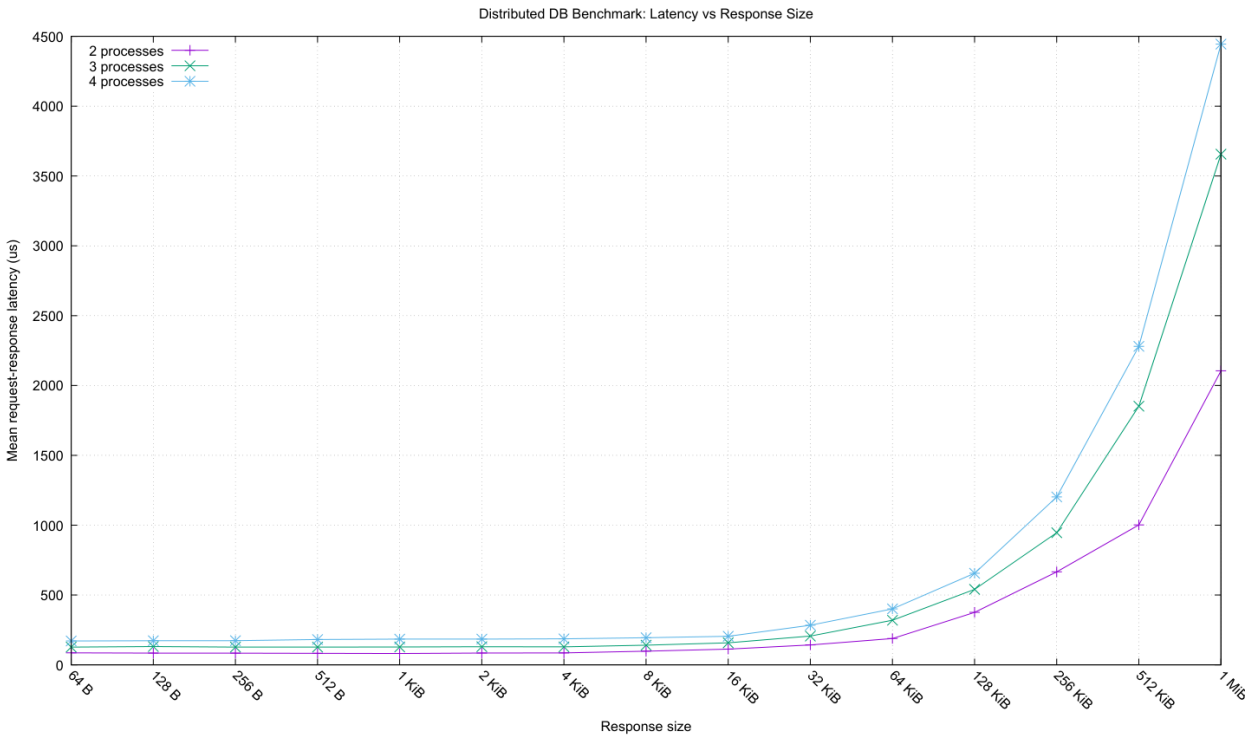
Merané hodnoty

Benchmark vypisuje pre každý proces samostatný riadok s nameranými hodnotami. Z týchto výstupov sa následne pripravujú grafy. Merané sú najmä tieto veličiny:

- priemerná request-response latencia v mikrosekundách, teda čas od odoslania požiadavky po prijatie odpovede,
- štandardná odchýlka latencie, ktorá ukazuje stabilitu jednotlivých meraní,
- počet vybavených požiadaviek za sekundu,
- priepustnosť odpovedí v MiB/s podľa veľkosti prenesených odpovedí.

Výsledky benchmarku

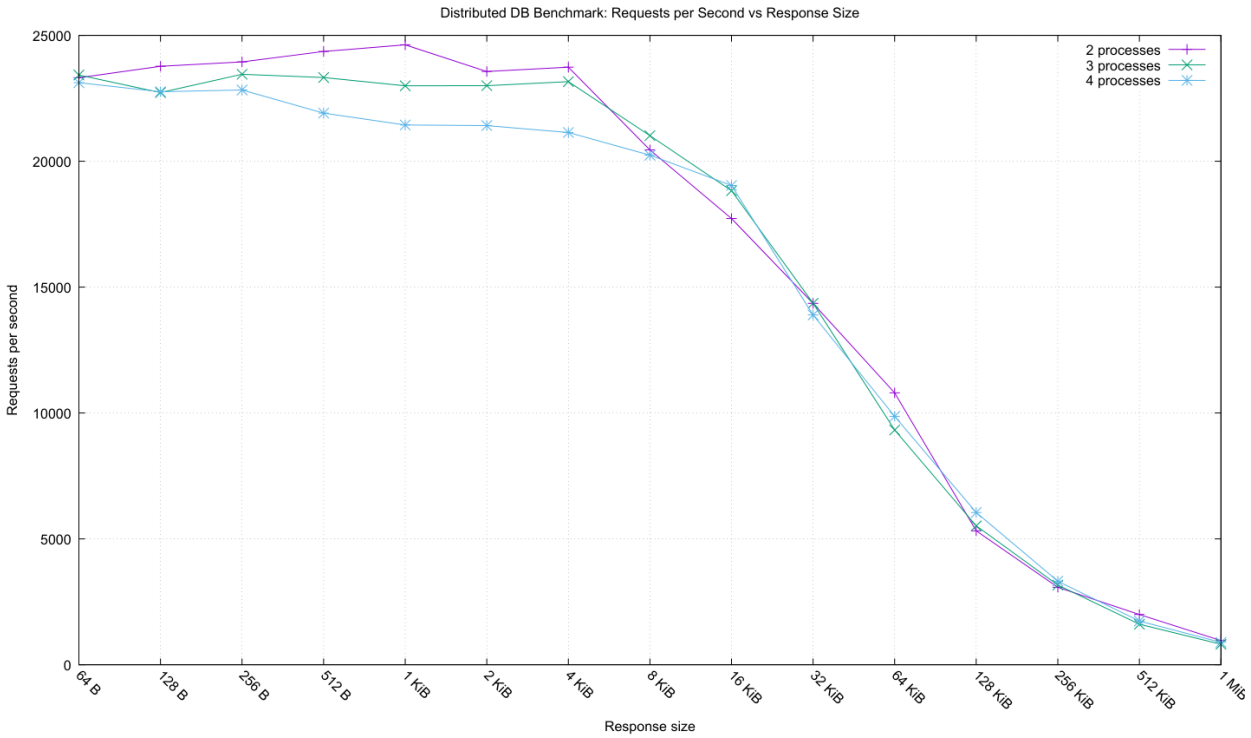
Prvý graf zobrazuje priemernú request-response latenciu v závislosti od veľkosti odpovede. Pri malých odpovediach je latencia nízka a rastie len pomaly. Pri väčších odpovediach, najmä od stoviek KiB po 1 MiB, rastie latencia výraznejšie, pretože sa prenáša a číta väčší objem dát.



Obrázok 3: Distributed DB Benchmark - Latency vs Response Size

Z grafu je tiež vidieť, že pri väčšom počte procesov je latencia spravidla vyššia. Je to očakávané, pretože v systéme súčasne prebieha viac komunikácie a viac procesov súťaží o systémové prostriedky.

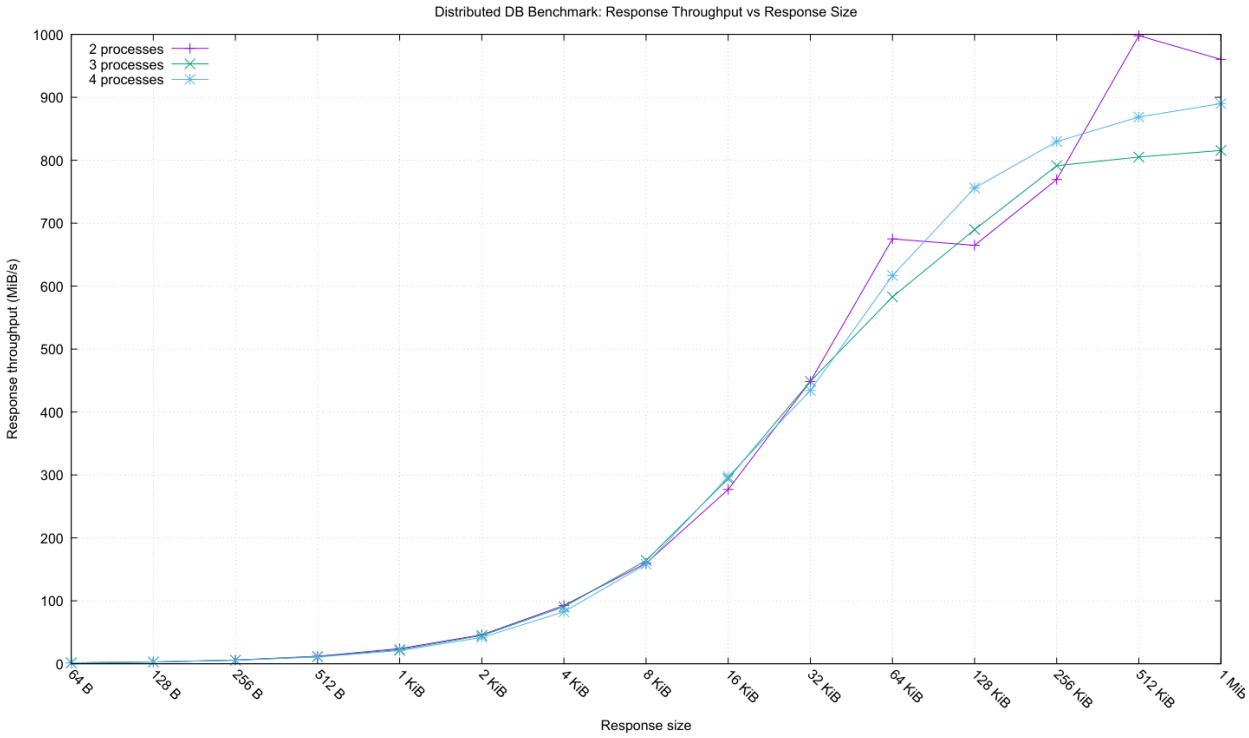
Druhý graf ukazuje počet požiadaviek vybavených za sekundu. Pri malých odpovediach je počet požiadaviek za sekundu vysoký, pretože jedna odpoveď obsahuje málo dát. S rastúcou veľkosťou odpovede počet požiadaviek za sekundu postupne klesá.



Obrázok 4: Distributed DB Benchmark - Requests per Second vs Response Size

Tento graf ukazuje hranicu medzi scenárom s malými správami, kde dominuje réžia synchronizácie a smerovania správ, a scenárom s väčšími odpoveďami, kde začína dominovať samotný prenos dát.

Tretí graf zobrazuje priepustnosť odpovedí v MiB/s. Pri malých odpovediach je priepustnosť nízka, pretože sa prenáša malý objem dát a významná časť času pripadá na réžiu komunikácie. Pri väčších odpovediach priepustnosť rastie, pretože v každej odpovedi sa prenesie viac dát.



Obrázok 5: Distributed DB Benchmark - Response Throughput vs Response Size

Výsledok ukazuje, že knižnica je použiteľná nielen na jednoduché odosielanie správ medzi všetkými procesmi, ale aj na scenár, ktorý sa viac podobá reálnej klient-server komunikácii. Benchmark zároveň umožňuje sledovať rozdiel medzi latenciou malých odpovedí a priepustnosťou pri väčších dátach.

Záver

V letnom semestri bola dokončená základná komunikačná knižnica pre výmenu správ medzi procesmi. Knižnica poskytuje jednoduché rozhranie na inicializáciu komunikácie, prípravu odosielacieho bufferu, odosielanie, prijímanie, multicast a korektné ukončenie práce.

Projekt bol rozšírený aj o benchmarky, ktoré umožňujú otestovať správanie knižnice pri rôznych komunikačných scenároch. Prvý benchmark meria all-to-all komunikáciu a ukazuje správanie celkového času a latencie pri rôznych veľkostiach správ. Druhý benchmark simuluje jednoduchú distribuovanú databázu a meria latenciu požiadaviek, počet požiadaviek za sekundu a priepustnosť odpovedí.

Výsledkom semestra je funkčný základ pre ďalšie experimentovanie s medziprocesovou komunikáciou. Na projekt je možné nadviazať ďalšími meraniami, porovnaním s inými IPC mechanizmami alebo doplnením zložitejších komunikačných vzorov.