

**BATTLESHIP**

SEMESTRÁLNA PRÁCA

Vypracovali: **Jakub Pastorek, Dávid Kubaľák**

Študijná skupina: **5ZYI31**

Predmet: **Princípy operačných systémov**

Cvičiaci: **Ing. Michal Mrena, PhD.**

Obsah

[1. Štruktúra projektu 3](#_Toc187532267)

[1.1. Server 4](#_Toc187532268)

[1.2. Klient 4](#_Toc187532269)

[1.3. Medziprocesová komunikácia 5](#_Toc187532270)

[1.4. GameLogic 5](#_Toc187532271)

[2. Procesy aplikácie 5](#_Toc187532272)

[2.1. Serverový proces 5](#_Toc187532273)

[2.2. Klientsky proces 6](#_Toc187532274)

[2.3. Interakcia medzi procesmi 6](#_Toc187532275)

[3. Použitia a účel vlákien 7](#_Toc187532276)

[3.1. Vlákno na serveri 7](#_Toc187532277)

[3.2. Vlákna na klientovi 7](#_Toc187532278)

[4. Typy a účel použitých prostriedkov IPC 8](#_Toc187532279)

[4.1. FIFO (Named Pipes) 8](#_Toc187532280)

[4.2. Semafory 8](#_Toc187532281)

[4.3. Mutexy 9](#_Toc187532282)

[4.4. Quit Pipe 9](#_Toc187532283)

[4.5. Signály a funkcia fork 9](#_Toc187532284)

[5. Synchronizačné problémy 10](#_Toc187532285)

[5.1. Súčasný prístup k zdieľaným zdrojom 10](#_Toc187532286)

[5.2. Koordinácia medzi serverom a klientmi 10](#_Toc187532287)

[5.3. Striedanie ťahov hráčov 11](#_Toc187532288)

[6. Kľúčové problémy, ktoré sme riešili 11](#_Toc187532289)

[8. Používateľská dokumentácia 12](#_Toc187532290)

[8.1. Spustenie aplikácie 12](#_Toc187532291)

# Štruktúra projektu

Projekt Battleship je navrhnutý ako viacprocesová a viacvláknová aplikácia, ktorá využiva medziprocesovú komunikáciu (IPC) a obsahuje nasledovné komponenty.

1. config.h – obsahuje definície a konštanty potrebné na konfiguráciu aplikácie.

BUFFER\_SIZE – veľkosť bufferu na správy

BOARD\_SIZE – veľkosť hernej mriežky

Cesty k FIFO a šablóny pre semafory

1. client.h a client.c – definujú funkcie a dáta pre klientov

Hlavné funkcie:

run\_client() : inicializácia klienta a jeho komunikácia so serverom

handle\_commands() : spracovanie vstupu od klientov

handle\_updates() : spracovanie aktualizácii od servera

1. server.h a server.c – správa serverovej logiky hry

Hlavné funkcie:

run\_server() : inicializácia servera a správa klientov

handle\_client\_message() : spracovanie správ od klientov

1. communication.h a communication.c – implementácia posielania a príjmania správ

send\_message() : odoslanie správy cez FIFO

receive\_message() : prijatie správy z FIFO

1. pipe.h a pipe.c – implementácia operácií so súbormi FIFO teda dátovody

pipe\_init() : inicializácia FIFO

pipe\_open\_read() : otvorenie FIFO na čítanie

pipe\_open\_write() : otvorenie FIFO na zápisonie

pipe\_close() : zatvorenie FIFO

1. game-logic.h a game-logic.c – obsahujú implementáciu hernej logiky

initialize\_board() : inicializácia hernej mriežky

place\_ship() : umiestnenie lode

attack() : spracovanie útoku

is\_game\_over() : overenie či hra skončila

1. main.c – hlavný bod spustenia aplikácie rozhoduje či spustí klienta a server pokiaľ je to potrebné ho spustiť
2. CMakeLists.txt – Skript na zostavenie projektu pomocou CMake

Obsahuje definície cieľov: client, server, all, clean

A diagram of a server

Description automatically generated

Tento UML diagram ilustruje architektúru projektu:

## Server

Celková správa hry a implementácia hernej logiky. Obsahuje všetky funkcie na inicializáciu hry, spracovanie útokov, správu hráčov a rozhodovanie o stave hry. Server príjma správy od klientov, spracováva ich pomocou game logic a posiela spätnú väzbu klientom.

## Klient

Slúži ako rozhranie pre hráča. Jeho úlohou je zhromaždiť vstup od používateľa, umiestnenie lode, zadávanie útokov, ukončenie a odosielať tieto údaje serveru. Okrem toho klient spracováva aktualizácie od serveru, aby hráč videl aktuálny stav hry.

## Medziprocesová komunikácia

Medzi klientom a serverom je realizovaní pomocou FIFO (modul Pipe) a správ (modul Communication). Tieto mechanizmy zabezpečujú efektívny a synchronizovaný prenos údajov.

## GameLogic

Poskytuje základné funkcie pre logiku hry, ako inicializácia, umiestnenie lodi, spracovanie útokov a kontrola stavu hry. Tento modul používa vylúčne server, pretože logika hry je centralizovaná na strane servera.

# Procesy aplikácie

Aplikácia Battleship je rozdelená na dva hlavné procesy, server a klient, ktoré spolu komunikujú pomocou mechanizmov IPC (FIFO, semafory). Každý z týchto procesov má svoje vlastné funkcie a zodpovednosti:

## Serverový proces

Server je zodpovedný za celú hernú logiku a správu klientov. Po spustení vykonáva nasledovné kroky:

1. **Inicializácia**:

Vytvára FIFO súbory na komunikáciu medzi klientmi a serverom.

Inicializuje semafory na synchronizáciu.

Pripraví hernú logiku.

1. **Pripojenie klientov:**

Čaká na pripojenie maximálne dvoch klientov.

Priraďuje klientom jedinečne ID a informuje ich o úspešnom pripojení.

1. **Spracovanie hernej logiky:**

Po pripojení oboch klientov začne hra.

Server spracováva správy od klientov a aktualizuje stav hry.

Kontroluje, či hra skončila, a v prípade výhry alebo ukončenia hráčom, cez správu QUIT, informuje oboch klientov.

1. **Ukončenie:**

Po skončení hry server uvoľní všetky zdroje FIFO, semafory a ukonči proces.

## Klientsky proces

Klient predstavuje rozhranie pre hráča. Jeho hlavné funkcie zahŕňajú:

1. **Inicializácia:**

Otvára FIFO na komunikáciu so serverom.

Pripravuje herný stav klienta.

1. **Interakcia s používateľom:**

Umožňuje hráčovi zadávať príkazy, ako:

Umiestnenie lodí (PLACE x y orientation)

Útok na súpera (ATTACK x y)

Ukončenie hry (QUIT)

1. **Spracovanie odpovedí servera:**

Klient prijíma správy zo servera napríklad výsledky útokov, aktuálny stav hry.

Na základe odpovedí aktualizuje svoju hernú mriežku a zobrazí hráčovi výsledok.

1. **Ukončenie:**

Po skončení hry klient uvoľní všetky zdroje a ukonči proces.

## Interakcia medzi procesmi

Klient a server spolu komunikujú pomocou týchto mechanizmov“

1. **FIFO (Named Pipes):**

Používa sa na obojsmernú komunikáciu medzi klientom a serverom.

Každý klient má vlastný FIFO súbor na čítanie, zápis správ.

1. **Semafory:**

Používajú sa na synchronizáciu prenosu správ, aby nedochádzalo ku kolíziám pri čítaní alebo zapisovaní do FIFO.

# Použitia a účel vlákien

V aplikácii Battleship sú vlákna využívané na podporu paralelného spracovania a zabezpečenie efektívnej komunikácie medzi klientom a serverom. Každý proces, server aj klient, má svoje vlastné vlákna, ktoré sú navrhnuté na splnenie špecifických úloh:

## Vlákno na serveri

Server využíva iba jedno vlákno na spracovanie správ od klientov a správu hernej logiky. Funkcia run\_server() beži v hlavnom vlákne.

1. **Spracovanie príkazov klientov:**

Toto vlákno čaká na príchod správ cez FIFO a spracováva príkazy pomocou funkcie handle\_client\_message().

1. **Zabezpečenie synchronizácie:**

Synchronizácia je zabezpečená pomocou mutexov napríklad prthread\_mutex\_lock v metode initialize\_game() a handle\_client\_message().

1. **Odosielanie odpovedí:**

Vlákno posiela odpovede klientom na základe ich príkazov.

## Vlákna na klientovi

Klient používa dve vlákna na paralelne spracovanie:

1. **Vlákno na spracovanie vstupu používateľa (handle\_commands)**

Spracováva príkazy od hráča, umiestnenie lodi, útoky a ukončenie hry.

1. **Vlákno na prijímanie správ od servera (handle\_updates)**

Čaká na odpovede zo servera ako výsledok útokov, informácie o stave hry, ukončenie hry a spracováva ich.

Obe vlákna sú vytvorené vo funkcii handle\_client\_threads() pomocou pthread\_create.

# Typy a účel použitých prostriedkov IPC

## FIFO (Named Pipes)

FIFO je základný prostriedok IPC v našej práci a je používaný na komunikáciu medzi serverom a klientmi.

1. **Použitie:**

Server používa FIFO na príjem správ od klientov a na odosielanie odpovedí klientom. FIFO sa inicializuje pomocou funkcie pipe\_init().

Klient používa FIFO na odosielanie a na prijmanie odpovedí zo servera.

1. **Implementácia:**

Funkcie pipe\_open\_read a pipe\_open\_write otvárajú FIFO na čítanie a zápis, pričom používame režim O\_RDWR, aby sa zabránilo blokovaniu.

Správy sú odosielané a prijímané funkciami send\_message a receive\_message.

1. **Účel:**

FIFO slúži na obojsmernú komunikáciu medzi procesmi servera a klientov.

Príklad komunikácie – Klient odosiela príkaz ako ATTACK alebo QUIT serveru a server odosiela výsledky útokov alebo informácie o ukončení hry klientovi.

## Semafory

Semafory sa používajú na synchronizáciu medzi serverom a klientmi.

1. **Použitie:**

Server vytvára semafory pre jednotlivé klientske pripojenia a operácie.

SEM\_CONNECT – synchronizuje inicializáciu servera a klienta.

SEM\_COMMAND – server čaká na príkazy od klientov.

SEM\_RESPONSE – server signalizuje odpoveď klientom.

Klienti používajú semafory na čakanie na odpovede od servera alebo na signalizovanie príkazov.

1. **Implementácia**

Funkcia initialize\_semaphore vytvára semafory pomocou sem\_open a nastavuje ich počiatočné hodnoty.

Semafory sa odstraňujú pomocou sem\_unlink pri čistení.

1. **Účel**

Semafory zabezpečia, aby server spracovával príkazy klientov postupne, čaká na príkaz klienta pomocou sem\_wait() a signalizujú udalosti medzi serverom a klinetom pomocou sem\_post()

## Mutexy

Mutexy sa používajú na synchronizáciu prístupu k zdieľaným zdrojom v serveri.

1. **Použitie:**

Server používa mutex game\_mutex na ochranu zdieľaných dát hry (game\_data).

pthread\_mutex\_lock a pthread\_mutex\_unlock chránia kritické sekcie, napríklad inicializáciu hry.

1. **Implementácia:**

Mutex sa inicializuje ako globálna premenná PTHREAD\_MUTEX\_INITIALIZER.

1. **Účel:**

Predísť súčasnému prístupu viacerých vlákien servera k rovnakým zdrojom.

## Quit Pipe

Pipe (quit\_pipe) sa používa na signalizáciu ukončenia procesov.

1. **Použitie:**

Klient vytvára pipe pomocou pipe systémového volania.

Vlákno handle\_commands monitoruje tento pipe pomocou select, aby zistilo, či má ukončiť prácu.

1. **Implementácia:**

Pipe sa inicializuje vo funkcii initialize\_quit\_pipe a na zápis signálu na ukončenie sa používa write.

1. **Učel:**

Zabezpečiť ukončenie klientskych vlákien pri ukončení hry.

## Signály a funkcia fork

Signály a fork sa používajú na vytváranie nového serverového procesu. Ak server ešte neexistuje, klient vytvorí nový proces pomocou fork. Dieťa procesu spustí server run\_server.

Umožňuje to dynamické spustenie servera pri pripojení prvého klienta.

# Synchronizačné problémy

## Súčasný prístup k zdieľaným zdrojom

Zdieľané dáta, ako napríklad herná štruktúra game\_data na serveri, môžu byť upravené viacerými vláknami súčasne. To by mohlo viesť k nekonzistentným údajom, napríklad ak by jeden klient aktualizoval stav hry, zatiaľ čo druhý klient pristupoval k týmto údajom.

**Riešenie:**

Použili sme **mutexy** (pthread\_mutex\_t) na ochranu zdieľaných zdrojov.

Mutex game\_mutex sa používa na serveri na uzamknutie prístupu k štruktúre game\_data počas jej aktualizácie.

Funkcie initialize\_game alebo handle\_client\_message používajú pthread\_mutex\_lock na uzamknutie a pthread\_mutex\_unlock na odomknutie kritických sekcií.

## Koordinácia medzi serverom a klientmi

Server musí čakať na príkazy od klientov (napríklad ATTACK, QUIT) a spracovať ich postupne. Rovnako klient musí čakať na odpovede od servera. Bez synchronizácie by mohli nastať problémy, napríklad server spracovával príkaz od klienta skôr, než klient dokončíl svoju akciu alebo viaceré príkazy sa spracuvali v nesprávnom poradí.

**Riešenie:**

Použili sme **semafory** na synchronizáciu udalostí.

Server teraz čaká na príkazy klientov.

Semafor sem\_command na serveri signalizuje, že klient odoslal nový príkaz.

Server volá sem\_wait, aby čakal na príkaz.

Klient už čaká na odpovede od servera.

Semafor sem\_response umožňuje klientovi čakať na odpoveď servera, kým server nedokončí spracovanie príkazu.

## Striedanie ťahov hráčov

Počas hry sa musia hráči striedať v ťahoch teda pri zadávaní ATTACK. Ak by obaja hráči mohli vykonať ATTACK naraz, narušilo by to pravidlá hry.

**Riešenie:**

Premenná player\_turn na serveri určuje, ktorý hráč je na ťahu.

Každý klient pri odoslaní príkazu ATTACK kontroluje, či je na rade, inak dostane od servera odpoveď WRONG\_TURN.

Synchronizácia sa dosahuje pomocou semaforov sem\_continue, ktoré signalizujú, že hráč môže pokračovať v hre.

Mutexu game\_mutex, ktorý uzamkne prístup k player\_turn počas kontroly a aktualizácie.

# Kľúčové problémy, ktoré sme riešili

Ukončenie hry a správne uvoľnenie všetkých systémových zdrojov.

Detekcia ukončenia hry pri manuálnom ukončení hráčom a pri výhre hráča.

Synchronizácia prístupu ku kritickým sekciám pomocou mutexov a semaforov.

Inicializácia a manažment systémových zdrojov (FIFO, semafory) pred spustením hry a ich správne odstránenie po ukončení hry.

Validácia vstupných údajov a spracovanie chybových stavov pri komunikácii medzi klientom a serverom.

# Používateľská dokumentácia

## Spustenie aplikácie

1. **Kompilácia projektu**

Aplikáciu je potrebné najskôr skompilovať pomocou nástroj CMake.

Postup:

Spustite nasledujúce príkazy v adresári s projektom:

mkdir build

cd build

cmake ..

make

Po úspešnej kompilácii sa vytvorí spustiteľný súbor client.

1. **Spustenie hry**

Spustiť príkazom: ./client <nazov\_serveru>

Tým sa nám inicializuje server a spusti prvý klient, druhého klienta pripojíme tým istým príkazom, musíme ale dávať pozor iba na to aby, sme pre druhého klienta použili ten istý názov serveru: ./client <nazov\_serveru>

Server a klient musia byť spustené na rovnakom systéme, ktorý podporuje FIFO a semafory.

1. **Ovládanie hry**

Po spustení klienta je hráč vyzvaný na umiestnenie svojich lodí na hernú mriežku.

Každé umiestnenie sa zadáva v tvare: (PLACE x y orientation)

X a Y sú súradnice na hernej mriežke.

Orientation predstavuje orientáciu lode (H - horizontálne, V - vertikálne).

Veľkosť lode nemusíme zadávať, pretože postupne ukladáme lode z flotily.

Počas príkazu PLACE nemusí hráč čakať na druhého hráča iba si vyplni všetky lode kým nebude flotila prázdna. Nasledne sa mu zobrazia dve mapy, svoja a súperová.

Ako náhle majú obaja hráči uložené všetky lode môžu jeden na druhého utočiť.

Hráči sa striedajú v útokoch.

Útok sa zadáva príkazom (ATTACK x y)

Server oznámi výsledok útoku a hra pokračuje, až kým nie sú všetky lode jedného hráča potopené.

Výhercom je hráč, ktorý potopil všetky lode ako prvý.

Následne sa hra ukončí.

1. **Manuálne ukončenie hry**

Hru je možné ukončiť manuálne príkazom (QUIT), v ktorejkoľvek fáze hry.

Server oznámi druhému hráčovi, že súper ukončil hru.