Московский авиационный институт
(Национальный исследовательский университет)
Факультет "Информационные технологии и прикладная математика"
Кафедра "Вычислительная математика и программирование"

Курсовая работа по курсу “Операционные системы”

Студент: Кочкожаров Иван Вячеславович

Группа: М8О-208Б-22

Преподаватель: Миронов Евгений Сергеевич

Оценка: _____

Дата: _____

Подпись: _____

Москва, 2023

Содержание

1 Репозиторий **3**

2 Цель работы **3**

3 Задание **3**

4 Исходный код **4**

5 Тесты **13**

6 Демонстрация работы **15**

7 Выводы **17**

1 Репозиторий

https://github.com/kochkozharov/os-labs

2 Цель работы

- Приобретение практических навыков в использовании знаний, полученных в течении курса
- Проведение исследования в выбранной предметной области

3 Задание

Необходимо спроектировать и реализовать программный прототип в соответствии с выбранным вариантом. Произвести анализ и сделать вывод на основании данных, полученных при работе программного прототипа. Проектирование основе любой из выбранных технологий:

- Pipes
- Sockets
- Сервера очередей
- И другие

Создать собственную игру более, чем для одного пользователя. Игра может быть устроена по принципу: клиент-клиент, сервер-клиент.

«Быки и коровы» (угадывать необходимо числа). Общение между сервером и клиентом необходимо организовать при помощи memory map. При создании каждой игры необходимо указывать количество игроков, которые будут участвовать. То есть угадывать могут несколько игроков. Должна быть реализована функция поиска игры, то есть игрок пытается войти в игру не по имени, а просто просит сервер найти ему игру.

4 Исходный код

shared_memory.cpp

```
#include "shared_memory.h"

#include <errno.h>
#include <sys/mman.h>
#include <unistd.h>

#include <cstdio>
#include <cstdlib>
#include <iostream>

WeakSharedMemory::WeakSharedMemory(std::string_view name
   , std::size_t size)
    : _name(name), _size(size) {
    _wSemPtr = sem_open((_name + "W").c_str(), O_CREAT,
       S_IRUSR | S_IWUSR, 1);
    _rSemPtr = sem_open((_name + "R").c_str(), O_CREAT,
       S_IRUSR | S_IWUSR, 0);
    _FD = shm_open(_name.c_str(), O_CREAT | O_RDWR,
       S_IRUSR | S_IWUSR);
    if (_FD < 0) {
        switch (errno) {
            case EACCES:
                throw SharedMemoryException("Permission␣
                   Exception");
                break;
            case EINVAL:
                throw SharedMemoryException(
                    "Invalid␣shared␣memory␣name␣passed")
                       ;
                break;
            case EMFILE:
                throw SharedMemoryException(
                    "The␣process␣already␣has␣the␣maximum
                       ␣number␣of␣files␣"
                    "open");
                break;
            case ENAMETOOLONG:
                throw SharedMemoryException(
                    "The␣length␣of␣name␣exceeds␣PATH_MAX
                       ");
                break;
            case ENFILE:
                throw SharedMemoryException(
                    "The␣limit␣on␣the␣total␣number␣of␣
                       files␣open␣on␣the␣system␣"
                    "has␣been␣reached");
```

```
                break;
            default:
                throw SharedMemoryException(
                    "Invalid␣exception␣occurred␣in␣
                       shared␣memory␣creation");
                break;
        }
    }
    ftruncate(_FD, _size);
    _ptr = mmap(nullptr, _size, O_RDWR, MAP_SHARED, _FD,
        0);
    if (_ptr == nullptr) {
        throw SharedMemoryException(
            "Exception␣in␣attaching␣the␣shared␣memory␣
               region");
    }
}

static bool SemTimedWait(sem_t* sem) {
    struct timespec absoluteTime;
    if (clock_gettime(CLOCK_REALTIME, &absoluteTime) ==
       -1) {
        return false;
    }
    absoluteTime.tv_sec += 5;
    return sem_timedwait(sem, &absoluteTime) == 0;
}

bool WeakSharedMemory::writeLock(bool timed) {
    if (timed) {
        return SemTimedWait(_wSemPtr);
    } else {
        sem_wait(_wSemPtr);
        return true;
    }
}

void WeakSharedMemory::writeUnlock() { sem_post(_wSemPtr
   ); }

bool WeakSharedMemory::readLock(bool timed) {
    if (timed) {
        return SemTimedWait(_rSemPtr);
    } else {
        sem_wait(_rSemPtr);
        return true;
    }
}
```

```
void WeakSharedMemory::readUnlock() { sem_post(_rSemPtr)
   ; }

WeakSharedMemory::~WeakSharedMemory() {
    if (sem_close(_rSemPtr) < 0) {
        std::perror("sem_close");
        std::abort();
    }
    if (sem_close(_wSemPtr) < 0) {
        std::perror("sem_close");
        std::abort();
    }
    if (munmap(_ptr, _size) < 0) {
        std::perror("munmap");
        std::abort();
    }
}

SharedMemory::~SharedMemory() {
    if (shm_unlink(getName().c_str()) < 0) {
        std::perror("shm_unlink");
        std::abort();
    }
    if (sem_unlink((getName() + "W").c_str()) < 0) {
        std::perror("sem_unlink");
        std::abort();
    }
    if (sem_unlink((getName() + "R").c_str()) < 0) {
        std::perror("sem_unlink");
        std::abort();
    }
}
```

utils.cpp

```
#include "utils.h"

#include <sys/types.h>

#include <algorithm>
#include <array>
#include <random>

#include "shared_memory.h"
#include "unistd.h"

bool operator==(const GuessResult &a, const GuessResult
   &b) {
    return (a.bulls == b.bulls) && (a.cows == b.cows);
}
```

```
GuessResult MakeGuess(int secret, int guess) {  // w/o
   checks
    int bullsCnt = 0;
    int cowsCnt = 0;
    constexpr int base = 10;
    std::array<int, base> secretArray{};
    std::array<int, base> guessArray{};
    for (int i = 0; i < 4; ++i) {
        auto c1 = secret % base;
        auto c2 = guess % base;
        if (c1 == c2) {
            ++bullsCnt;
        } else {
            ++secretArray[c1];
            ++guessArray[c2];
        }
        secret /= base;
        guess /= base;
    }
    for (int i = 0; i < base; ++i) {
        cowsCnt += std::min(secretArray[i], guessArray[i
           ]);
    }
    return {bullsCnt, cowsCnt};
}

bool operator<(const Game &a, const Game &b) {
    return a.freeSlots < b.freeSlots;
}

int GenMysteryNumber() {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::string digits = "0123456789";
    std::shuffle(digits.begin(), digits.end(), gen);
    std::string result = digits.substr(0, 4);
    int number = std::stoi(result);
    return number;
}
```

client.cpp

```
#include <unistd.h>

#include <iostream>
#include <vector>

#include "shared_memory.h"
#include "utils.h"
```

```
int main() {
    WeakSharedMemory req(REQUEST_SLOT_NAME, sizeof(
       Request));
    WeakSharedMemory rep(RESPONSE_SLOT_NAME, sizeof(
       Response));
    auto *reqPtr = static_cast<Request *>(req.getData())
       ;
    auto *repPtr = static_cast<Response *>(rep.getData()
       );
    int gameID;
    int maxSlots;
    std::string command;
    while (std::cin >> command) {
        if (command == "create") {
            std::cin >> maxSlots;
            req.writeLock();
            reqPtr->newGame = true;
            reqPtr->pid = getpid();
            reqPtr->maxSlots = maxSlots;
            req.readUnlock();
            if (!rep.readLock(true)) {
                exit(EXIT_SUCCESS);
            }
            gameID = repPtr->gameID;
            rep.writeUnlock();
        } else if (command == "connect") {
            req.writeLock();
            reqPtr->newGame = false;
            reqPtr->pid = getpid();
            req.readUnlock();
            if (!rep.readLock(true)) {
                exit(EXIT_SUCCESS);
            }
            gameID = repPtr->gameID;
            maxSlots = repPtr->maxSlots;
            rep.writeUnlock();
        } else if (command == "stop") {
            req.writeLock();
            reqPtr->pid = -1;
            req.readUnlock();
            exit(EXIT_SUCCESS);
        } else {
            std::cerr << "Unknown␣command\n";
            continue;
        }
        if (gameID != -1) {
            std::cout << "Connected␣to␣game␣" << gameID
               << '\n';
```

```
            break;
        }
        std::cerr << "No free games available. Try
           creating new\n";
    }

    WeakSharedMemory gameMemory(
        "BC" + std::to_string(gameID),
        sizeof(int) + maxSlots * sizeof(ConnectionSlot))
           ;
    auto *statusPtr = static_cast<int *>(gameMemory.
       getData());
    auto *gamePtr = reinterpret_cast<ConnectionSlot *>(
       statusPtr + 1);
    int conID = 0;
    bool connected = false;
    while (true) {
        int guess;
        std::cin >> guess;
        if (guess > 9999 || guess < -1) {
            std::cerr << "Incorrect format\n";
            continue;
        }
        gameMemory.writeLock();
        while (!connected && gamePtr[conID].pid != 0) {
            ++conID;
        }
        connected = true;
        *statusPtr = conID;
        gamePtr[conID].pid = getpid();
        gamePtr[conID].guess = guess;
        gameMemory.readUnlock();
    }
}
```

server.cpp

```
#include <csignal>
#include <deque>
#include <format>
#include <iostream>
#include <queue>

#include "shared_memory.h"
#include "thread"
#include "utils.h"


static void GameLoop(SharedMemory &gameMemory, int
   maxSlots) {
```

```
    int mysteryNumber = GenMysteryNumber();
    auto *statusPtr = static_cast<int *>(gameMemory.
       getData());
    auto *gamePtr = reinterpret_cast<ConnectionSlot *>(
       statusPtr + 1);
    while (true) {
        gameMemory.readLock();
        auto which = *statusPtr;
        if (gamePtr[which].guess == -1) {
            break;
        }
        auto res = MakeGuess(mysteryNumber, gamePtr[
           which].guess);
        auto outputStr = std::format(
            "Player␣{}:\n\tGuess␣{}\tBulls␣{}\tCows␣{}\n
               ", gamePtr[which].pid,
            gamePtr[which].guess, res.bulls, res.cows);
        for (int i = 0; i < maxSlots; ++i) {
            auto pid = gamePtr[i].pid;
            std::cerr << pid << '\n';
            if (pid != -1) {
                auto fd =
                    open((std::format("/proc/{}/fd/0",
                       pid)).c_str(), O_WRONLY);
                write(fd, outputStr.c_str(), outputStr.
                   size() + 1);
            }
        }
        std::cerr << '\n';
        gameMemory.writeUnlock();
    }
    int i = 0;
    while (gamePtr[i].pid != 0){
        auto pid = gamePtr[i].pid;
        kill(pid, SIGTERM);
        ++i;
    }
}

int main() {
    SharedMemory req(REQUEST_SLOT_NAME, sizeof(Request))
       ;
    SharedMemory rep(RESPONSE_SLOT_NAME, sizeof(Response
       ));
    auto *reqPtr = static_cast<Request *>(req.getData())
       ;
    auto *repPtr = static_cast<Response *>(rep.getData()
       );
```

```
std::priority_queue<Game> pq;
std::deque<SharedMemory> games;
std::vector<std::thread> threads;
int gamesCount = 0;

int gameID;
int maxSlots;
while (true) {
    req.readLock();
    if (reqPtr->pid == -1) {
        break;
    }
    if (reqPtr->newGame) {
        pq.emplace(gamesCount, reqPtr->maxSlots,
           reqPtr->maxSlots);
        gameID = gamesCount;
        maxSlots = reqPtr->maxSlots;
        gamesCount++;
        games.emplace_back("/BC" + std::to_string(
           gameID),
                           sizeof(int) + maxSlots *
                              sizeof(ConnectionSlot)
                              );
        threads.emplace_back(GameLoop, std::ref(
           games[gameID]), maxSlots);
        std::cout << "Created␣new␣game␣" << gameID
           << '\n';
    } else {
        auto freeGame = pq.top();
        if (freeGame.freeSlots == 0) {
            gameID = -1;
        } else {
            gameID = freeGame.gameID;
            pq.pop();
            freeGame.freeSlots--;
            pq.push(freeGame);
            maxSlots = freeGame.freeSlots;
            std::cout << "Connected␣to␣game␣" <<
               gameID << '\n';
        }
    }
    req.writeUnlock();
    rep.writeLock();
    repPtr->maxSlots = maxSlots;
    repPtr->gameID = gameID;
    rep.readUnlock();
}
for (auto &t : threads) {
    t.join();
```

```
        }
    }
```

5 Тесты

```
#include <gtest/gtest.h>

#include "shared_memory.h"
#include "utils.h"

TEST(Lab5Tests, CalculationTest) {
    EXPECT_EQ(MakeGuess(1234, 1234), GuessResult(4, 0));
    EXPECT_EQ(MakeGuess(1243, 1234), GuessResult(2, 2));
    EXPECT_EQ(MakeGuess(1243, 9847), GuessResult(1, 0));
    EXPECT_EQ(MakeGuess(1243, 9147), GuessResult(1, 1));
    EXPECT_EQ(MakeGuess(2301, 127), GuessResult(0, 3));
}

TEST(Lab5Tests, SharedMemoryTest) {
    SharedMemory a("test", 10);
    for (int i = 0; i < 10; ++i) {
        static_cast<char *>(a.getData())[i] = i + '0';
    }
    auto pid = fork();
    if (pid == 0) {
        WeakSharedMemory b("test", 10);
        for (int i = 0; i < 10; ++i) {
            EXPECT_EQ(static_cast<char *>(b.getData())[i
               ], i + '0');
        }
        exit(EXIT_SUCCESS);
    }

    wait(nullptr);
}

TEST(Lab5Tests, lockTest) {
    auto pid = fork();
    if (pid == 0) {
        WeakSharedMemory b("test", 10);
        for (int i = 0; i < 10; ++i) {
            static_cast<char *>(b.getData())[i] = i + '0
               ';
        }
        b.readUnlock();
        exit(EXIT_SUCCESS);
    }
    SharedMemory a("test", 10);
    a.readLock();
    for (int i = 0; i < 10; ++i) {
        EXPECT_EQ(static_cast<char *>(a.getData())[i], i
            + '0');
```

```
    }
}

#include <gtest/gtest.h>

#include "shared_memory.h"
#include "utils.h"

TEST(Lab5Tests, CalculationTest) {
    EXPECT_EQ(MakeGuess(1234, 1234), GuessResult(4, 0));
    EXPECT_EQ(MakeGuess(1243, 1234), GuessResult(2, 2));
    EXPECT_EQ(MakeGuess(1243, 9847), GuessResult(1, 0));
    EXPECT_EQ(MakeGuess(1243, 9147), GuessResult(1, 1));
    EXPECT_EQ(MakeGuess(2301, 127), GuessResult(0, 3));
}

TEST(Lab5Tests, SharedMemoryTest) {
    SharedMemory a("test", 10);
    for (int i = 0; i < 10; ++i) {
        static_cast<char *>(a.getData())[i] = i + '0';
    }
    auto pid = fork();
    if (pid == 0) {
        WeakSharedMemory b("test", 10);
        for (int i = 0; i < 10; ++i) {
            EXPECT_EQ(static_cast<char *>(b.getData())[i], i + '0');
        }
        exit(EXIT_SUCCESS);
    }

    wait(nullptr);
}

TEST(Lab5Tests, lockTest) {
    auto pid = fork();
    if (pid == 0) {
        WeakSharedMemory b("test", 10);
        for (int i = 0; i < 10; ++i) {
            static_cast<char *>(b.getData())[i] = i + '0';
        }
        b.readUnlock();
        exit(EXIT_SUCCESS);
    }
    SharedMemory a("test", 10);
    a.readLock();
    for (int i = 0; i < 10; ++i) {
        EXPECT_EQ(static_cast<char *>(a.getData())[i], i + '0');
    }
}
```

6 Демонстрация работы

Консоль 1

```
ivan@asus-vivobook ~/c/o/b/cp (cp)> game_serv
er
Created new game 0
Connected to game 0
704945
0
0
0

704945
704963
0
0

704945
704963
0
0

704945
704963
0
0

704945
704963
0
0
```

Консоль 3

```
ivan@asus-vivobook ~/c/o/b/cp (cp)> game_clien
t
create 4
Connected to game 0
1234
Player 704945:
        Guess 1234      Bulls 0 Cows 2
Player 704963:
        Guess 1324      Bulls 0 Cows 2
Player 704963:
        Guess 1324      Bulls 0 Cows 2
Player 704963:
        Guess 5326      Bulls 1 Cows 0
3456
Player 704945:
        Guess 3456      Bulls 1 Cows 1
```

Консоль 2

```
ivan@asus-vivobook ~/c/o/b/cp (cp)> game_cli
ent
connect
Connected to game 0
1324
Player 704963:
        Guess 1324      Bulls 0 Cows 2
1324
Player 704963:
        Guess 1324      Bulls 0 Cows 2
5326
Player 704963:
        Guess 5326      Bulls 1 Cows 0
Player 704945:
        Guess 3456      Bulls 1 Cows 1
```

7 Выводы

В результате выполнения данной курсовой работы были приобретены практические навыки в использовании знаний, полученных в течении курса, проведение исследования в выбранной предметной области. Были улучшены навыки использования memory map для разделения памяти между процессами и написания "оберток"для низкоуровневых системных вызовов.