Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работ №3 по курсу

«Операционные системы»

УПРАВЛЕНИЕ ПОТОКАМИ В ОС

Студент: Косогоров Владислав Валерьевич
Группа: М8О–206Б–18
Вариант: 11
Преподаватель: Соколов Андрей Алексеевич
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2019.

Содержание

Постановка задачи.

Составить программу на языке СИ, обрабатывающие данные в многопоточном режиме. При создании необходимо предусмотреть ключи, которые позволяли бы задать максимальное количество потоков, используемое программой. При возможности необходимо использовать максимальное количество возможных потоков.

Вариант 11: На вход программе подаются игровое поле для игры «Крестики-нолики» и ход какого игрока сейчас идет. Программа должна выдать наиболее оптимальный ход для заданного игрока (если их несколько, то выдать все).

Общие сведения о программе

Программа компилируется из одного файла main.c. В данном файле используются заголовочные файлы stdio.h, stdlib.h, pthread.h. В программе используются следующие системные вызовы для работы с потоками из заголовочного файла pthread.h:

pthread_create – для создания нового потока

pthread_join – синхронизации потоков в случае завершения их выполнения.

pthread_mutex_init – инициализация mutex.

pthread_mutex_lock – захват mutex.

pthread_mutex_unlock – освобождение mutex.

Программа считывает данные с stdin. Сначала пользователь вводит количество потоков, а затем текущее положение игры и символ игрока, который сейчас ходит.

Общий метод и алгоритм решения.

После получения входных данных, программа преобразует поле для игры в одномерный массив с индексами от 0 до 8. Далее происходит проверка, не выйграна ла партия заранее. Если это так, то выводится соответствующее сообщение и программа завершает свою работу. В противном случае, с помощью функции pthread_create() создается массив идентификаторов потоков, одним из аргументов для которой является функция ThreadFunction(), в которой в многопоточном режиме происходят основные вычисления.

Алгоритм вычисления: каждый поток вычисляет ход для своей на поле. Если она уже занята, то вычисления не требуются. Иначе происходит проверка, победит ли игрок, если поставит на крестик или нолик на данную клетку. Так же проверяется ситуация, не предотвратит ли игрок победу соперника, если сделает ход на данное поле. После окончания работы всех потоков, все потоки объединяются с помощью pthread_join(). Если есть ходы, которые позволяют выйграть, то выводятся только номера полей для этих ходов. Ниже приоритет у ходов, которые предотвращают победу противника. Если нельзя сделать ни то, ни другое, то выбор идет среди незанятых полей.

Основные файлы программы.

Файл main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

int NUM_THREADS;
pthread_mutex_t lock;
char field[9];
char player, opponent;
int idForWin = 0, idForLose = 0;
int winField[9];
int loseField[9];

int CheckForWin() {
        if ((field[0] == field[1] && field[0] == field[2] && (field[0] == 'x' || field[0] == 'o')) ||
                (field[3] == field[4] && field[5] == field[3] && (field[3] == 'x' || field[3] == 'o')) ||
                (field[6] == field[7] && field[8] == field[6] && (field[6] == 'x' || field[6] == 'o')) ||
                (field[0] == field[3] && field[6] == field[0] && (field[0] == 'x' || field[0] == 'o')) ||
                (field[1] == field[4] && field[7] == field[4] && (field[1] == 'x' || field[1] == 'o')) ||
                (field[2] == field[5] && field[8] == field[5] && (field[2] == 'x' || field[2] == 'o')) ||
                (field[0] == field[4] && field[8] == field[0] && (field[0] == 'x' || field[0] == 'o')) ||
                (field[2] == field[4] && field[6] == field[2] && (field[2] == 'x' || field[2] == 'o'))) {
```

```
            return 1;
        }
        return 0;
}

void* ThreadFunction(void* arg) {
        long cellNumber = (long)arg;

        if (field[cellNumber] == 'x' || field[cellNumber] == 'o') {
                return (void*)cellNumber;
        }

        if (pthread_mutex_lock(&lock) == -1) {
                perror("Mutex lock error");
                exit(-1);
        }

        switch (cellNumber) {
        case 0:
                if ((field[1] == field[2] && field[1] == player)
                 || (field[3] == field[6] && field[3] == player)
                 || (field[4] == field[8] && field[4] == player)) {
                        winField[idForWin++] = 0;
                } else if((field[1] == field[2] && field[1] == opponent)
                        || (field[3] == field[6] && field[3] == opponent)
                        || (field[4] == field[8] && field[4] == opponent)) {
                        loseField[idForLose++] = 0;
                }
```

```
            break;
        case 1:
            if ((field[0] == field[2] && field[0] == player)
                    || (field[4] == field[7] && field[4] == player)) {
                    winField[idForWin++] = 1;
            }
            else if ((field[0] == field[2] && field[0] == opponent)
                    || (field[4] == field[7] && field[4] == opponent)) {
                    loseField[idForLose++] = 1;
            }
            break;
        case 2:
            if ((field[0] == field[1] && field[1] == player)
                    || (field[5] == field[8] && field[8] == player)
                    || (field[4] == field[6] && field[6] == player)) {
                    winField[idForWin++] = 2;
            }
            else if ((field[0] == field[1] && field[1] == opponent)
                    || (field[5] == field[8] && field[8] == opponent)
                    || (field[4] == field[6] && field[6] == opponent)) {
                    loseField[idForLose++] = 2;
            }
            break;
        case 3:
            if ((field[0] == field[6] && field[0] == player)
                    || (field[6] == field[5] && field[5] == player)) {
                    winField[idForWin++] = 3;
            }
```

```
        else if ((field[0] == field[6] && field[0] == opponent)
                || (field[6] == field[5] && field[5] == opponent)) {
                loseField[idForLose++] = 3;
        }
        break;
case 4:
        if ((field[0] == field[8] && field[8] == player)
                || (field[2] == field[6] && field[6] == player)
                || (field[3] == field[5] && field[5] == player)
                || (field[1] == field[7] && field[7] == player)) {
                winField[idForWin++] = 4;
        }
        else if ((field[0] == field[8] && field[8] == opponent)
                || (field[2] == field[6] && field[6] == opponent)
                || (field[3] == field[5] && field[5] == opponent)
                || (field[1] == field[7] && field[7] == opponent)) {
                loseField[idForLose++] = 4;
        }
        break;
case 5:
        if ((field[3] == field[4] && field[4] == player)
                || (field[8] == field[2] && field[2] == player)) {
                winField[idForWin++] = 5;
        }
        else if ((field[3] == field[4] && field[4] == opponent)
                || (field[8] == field[2] && field[2] == opponent)) {
                loseField[idForLose++] = 5;
        }
```

```
            break;
        case 6:
            if ((field[0] == field[3] && field[3] == player)
                    || (field[7] == field[8] && field[8] == player)
                    || (field[4] == field[2] && field[2] == player)) {
                    winField[idForWin++] = 6;
            }
            else if ((field[0] == field[3] && field[3] == opponent)
                    || (field[7] == field[8] && field[8] == opponent)
                    || (field[4] == field[2] && field[2] == opponent)) {
                    loseField[idForLose++] = 6;
            }
            break;
        case 7:
            if ((field[1] == field[4] && field[4] == player)
                    || (field[8] == field[6] && field[6] == player)) {
                    winField[idForWin++] = 7;
            }
            else if ((field[1] == field[4] && field[4] == opponent)
                    || (field[8] == field[6] && field[6] == opponent)) {
                    loseField[idForLose++] = 7;
            }
            break;
        case 8:
            if ((field[0] == field[4] && field[4] == player)
                    || (field[2] == field[5] && field[5] == player)
                    || (field[7] == field[6] && field[6] == player)) {
                    winField[idForWin++] = 8;
```

```
            }
            else if ((field[0] == field[4] && field[4] == opponent)
                    || (field[2] == field[5] && field[5] == opponent)
                    || (field[7] == field[6] && field[6] == opponent)) {
                    loseField[idForLose++] = 8;
            }
            break;
        }

        if (pthread_mutex_unlock(&lock) == -1) {
            perror("Mutex unlock error");
            exit(-1);
        }

        return (void*)cellNumber;
}

void Solution() {

        if (CheckForWin()) {
            printf("Someone has already won\n");
            return;
        }

        if (pthread_mutex_init(&lock, NULL) == -1) {
            perror("Mutex init error");
            exit(-1);
        }
```

```
	pthread_t threadID[NUM_THREADS];

	long i;

	for (i = 0; i < NUM_THREADS; i++) {
		pthread_create(&threadID[i], NULL, ThreadFunction, (void*)
(long)i);
	}

	for (i = 0; i < NUM_THREADS; i++) {
		pthread_join(threadID[i], NULL);
	}

	if (idForWin > 0) {
		for (int i = 0; i < idForWin; i++) {
			printf("Best move: %d\n", winField[i]);
		}
	}
	else if(idForLose > 0) {
		for (int i = 0; i < idForLose; i++) {
			printf("Best move: %d\n", loseField[i]);
		}
	}
	else {
		if (field[4] != 'x' && field[4] != 'o') {
			printf("Best move: 4\n");
		}
		else {
```

```
                unsigned found = 0;
                if (field[0] != 'x' && field[0] != 'o') {
                        printf("Best move: %d\n", 0);
                        found = 1;
                }
                if (field[2] != 'x' && field[2] != 'o') {
                        printf("Best move: %d\n", 2);
                        found = 1;
                }
                if (field[6] != 'x' && field[6] != 'o') {
                        printf("Best move: %d\n", 6);
                        found = 1;
                }
                if (field[8] != 'x' && field[8] != 'o') {
                        printf("Best move: %d\n", 8);
                        found = 1;
                }
                if (!found) {
                        for (int i = 0; i < 9; ++i) {
                                if (field[i] != 'x' && field[i] != 'o') {
                                        printf("Best move: %d\n", i);
                                }
                        }
                }
        }
    }
}
```

```
int main(int argc, char** argv) {

	if (argc < 2) {
		printf("Usage: %s <number of threads>\n", argv[0]);
		return -1;
	}

	if ((NUM_THREADS = atoi(argv[1])) < 1) {
		perror("Invalid number of threads");
		return -1;
	}

	printf("Enter the field:\n");

	for (int i = 0; i < 9; ++i) {
		scanf("%c ", &field[i]);
	}

	scanf("%c", &player);
	if (player == 'o') {
		opponent = 'x';
	} else {
		opponent = 'o';
	}

	Solution();

	return 0;
}
```

Демонстрация работы программы.

```
…/prog_3_sem/os/lab3 ./a.out 9

Enter the field:
xox
oxo
---
x
Best move: 6
Best move: 8


…/prog_3_sem/os/lab3 ./a.out 9

Enter the field:
---
---
---
o
Best move: 4


…/prog_3_sem/os/lab3 ./a.out 10

Enter the field:
xx-
o--
---
o
Best move: 2


…/prog_3_sem/os/lab3 ./a.out 16

Enter the field:
oo-
xx-
---
x
Best move: 5


…/prog_3_sem/os/lab3 ./a.out 16

Enter the field:
oo-
```

```
xx-
---
o
Best move: 2


…/prog_3_sem/os/lab3 ./a.out 16

Enter the field:
xxx
---
---
x
Someone has already won


…/prog_3_sem/os/lab3 strace -c ./a.out 25 <result.txt

Enter the field:
Best move: 6
Best move: 8
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 0.00    0.000000           0         3           read
 0.00    0.000000           0         3           write
 0.00    0.000000           0         3           close
 0.00    0.000000           0         5           fstat
 0.00    0.000000           0         1           lseek
 0.00    0.000000           0        34           mmap
 0.00    0.000000           0        31           mprotect
 0.00    0.000000           0        22           munmap
 0.00    0.000000           0         3           brk
 0.00    0.000000           0         2           rt_sigaction
 0.00    0.000000           0         1           rt_sigprocmask
 0.00    0.000000           0         4         4 access
 0.00    0.000000           0        25           clone
 0.00    0.000000           0         1           execve
 0.00    0.000000           0         1           arch_prctl
 0.00    0.000000           0         1           set_tid_address
 0.00    0.000000           0         3           openat
 0.00    0.000000           0         1           set_robust_list
 0.00    0.000000           0         1           prlimit64
------ ----------- ----------- --------- --------- ----------------
100.00    0.000000                   145         4 total
```

Вывод

Одно из главных отличий использования многопоточности от использования процессов состоит в том, что потоки делят между собой одно адресное пространство, но параллельная обработка одних и тех же данных требует ответственного подхода к написанию программы. В данной лабораторной

работе была продемонстрирована обработка всевозможных вариантов исхода игры «крестики-нолики» в многопоточном режиме.