Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Операционные системы и системное программирование

ОТЧЕТ
к лабораторной работе №1
на тему

СКРИПТЫ SHELL

Студент О. Л. Дайнович
Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

1 ПОСТАНОВКА ЗАДАЧИ

Написать скрипт(ы) для оболочки shell (например, bash, zsh, csh), который(ые) обеспечат получение заданным образом организованной выходной информации. Результаты выполнения скрипта(ов) записываются в файл для последующего анализа. Используются перенаправление ввода-вывода, внешние утилиты и фильтры, а также переменные и структуры данных shell для обработки данных.

Для редактирования скрипта(ов) рекомендуется использовать консольные редакторы текста, такие как vim или nano. Размер окна консоли считается жестко заданным и не изменяется во время выполнения (это существенно для некоторых вариантов заданий).

2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Операционные системы семейства Linux, как впрочем, и любые другие ОС, предполагают наличие интерфейса взаимодействия между компонентами компьютерной системы и конечным пользователем, т. е. наличие программного уровня, который обеспечивает ввод команд и параметров для получения желаемых результатов. Такой программный уровень получил название "оболочка" или, на английском языке – shell.

Командная оболочка (shell) обеспечивает взаимотействие между пользователем и средой операционной системы Linux. Она является специализированным программным продуктом, который обеспечивает выполнение команд и получения результатов их выполнения, или, если совсем уж упрощенно, оболочка – это программа, которая предназначена для обеспечения выполнения других программ по желанию пользователя. Примером оболочки может быть, например, интерпретатор команд command.com операционной системы MS DOS, или оболочка bash операционных систем Unix / Linux.

Все оболочки имеют схожие функции и свойства, в соответствием с их основным предназначением - выполнять команды пользователя и отображать результаты их выполнения:

– интерпретация командной строки;

– доступ к командам и результатам их выполнения;

– поддержка переменных, специальных символов и зарезервированных слов;

– обработка файлов, операций стандартного ввода и вывода;

– реализация специального языка программирования оболочки.

В системе может быть установлено несколько различных оболочек, и для каждого пользователя возможно использование своей, запускаемой по умолчанию, оболочки. Все это, естественно, выполняется автоматически в процессе загрузки и регистрации пользователя.

Пользовательский ввод в ответ на приглашение оболочки обычно называют командной строкой или командой. Команда Linux – это строка символов из имени команды и аргументов, разделенных пробелами. Аргументы предоставляют команде дополнительные параметры, определяющие ее поведение. Наиболее часто в качестве аргументов используются опции и имена файлов и каталогов. При использовании нескольких опций, их можно объединять. [1]

Наибольшее распространение получили POSIX-совместимые оболочки, ведущие родословную от Bourne shell (шелл Борна).

Bourne shell и его клоны:

1 Bourne shell. Исполняемый файл: sh. Командная оболочка названная в честь своего создателя Стивена Борна. Большая часть операторов была заимствована им из языка Алгол 68. Вышла в 7-м издании операционной системы UNIX, где была оболочкой по умолчанию. До сих пор подавляющее большинство Unix-подобных систем имеют /bin/sh — символическую или жесткую ссылку на sh-совместимую оболочку.

2 Bourne again shell. Исполняемый файл: bash. Название можно перевести, как «Возрождённый шел Борна». Скорее всего самая популярная оболочка на сегодняшний день. Де-факто стандарт для Linux. Не буду на ней останавливаться, т.к. в интернете много хороших статей про bash.

3 Z shell. Исполняемый файл: zsh. Свободная современная sh-совместимая оболочка. Имеет ряд преимуществ перед bash касающихся в основном работы в интерактивном режиме.

Кроме того существует довольно много оболочек попадающих в эту группу: Korn shell (ksh) и Almquist shell (ash) etc.

C shell:

1 C shell, исполняемый файл: csh Командная оболочка разработанная автором vi Биллом Джоем. За основу для скриптового языка csh был взят, как понятно из названия, язык C. Т.к. на тот момент, в 1978 г., это был наиболее популярный язык программирования среди разработчиков и пользователей BSD UNIX. В настоящий момент более популярна свободная реализация csh – tcsh.

2 TENEX C Shell, исполняемый файл: tcsh. Именно в tcsh когда-то впервые появилось автодополнение. Является оболочкой по умолчанию в FreeBSD. [2]

3 ОПИСАНИЕ ФУНКЦИЙ ПРОГРАММЫ

В ходе выполнения лабораторной работы было разработано консольное приложение с помощью командной оболочки shell. Приложение представляет из себя игру «Морской бой» и имеет возможность записывать расположение кораблей на поле в файл с расширением .txt.

Расположение кораблей происходит случайным образом как на поле игрока, так и на поле бота. Каждая клетка поля для игры представляет собой отдельный emoji. Корабли на поле отмечены зелеными квадратами, подбитые корабли помечаются красными крестами, пустые клетки – белыми квадратами, а промахи – кругами. Пользователь имеет возможность играть с ботом, Ход игрока совершается посредством ввода в консоль координат клетки поля, для совершения «выстрела» (рисунок 1).

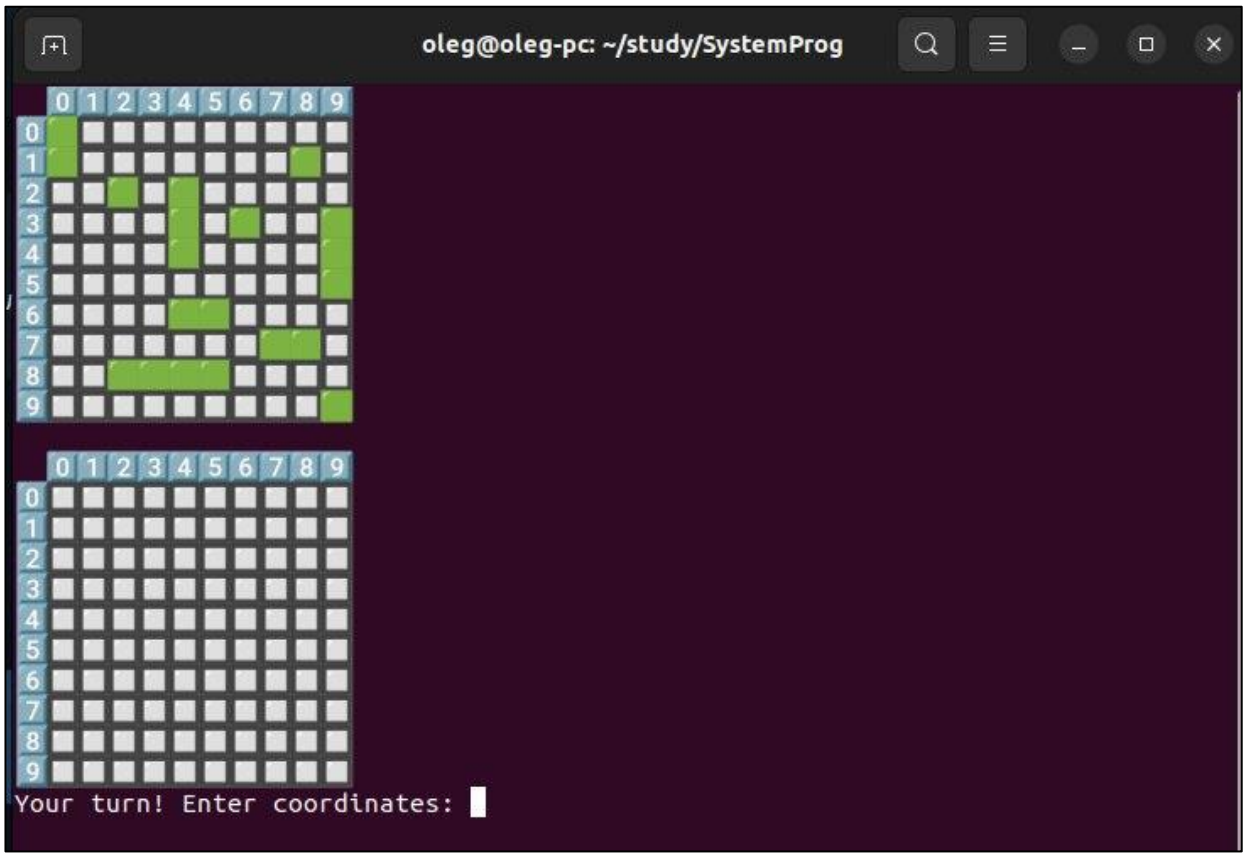

Рисунок 1 – Очередь хода игрока

Бот и игрок совершают ходы по очереди. В случае попадания по «кораблю» право хода остается у того, кто совершил попадание. Бот

совершает ходы произвольным образом. Во время очереди хода бота высвечивается соответствующее сообщение (рисунок 2).

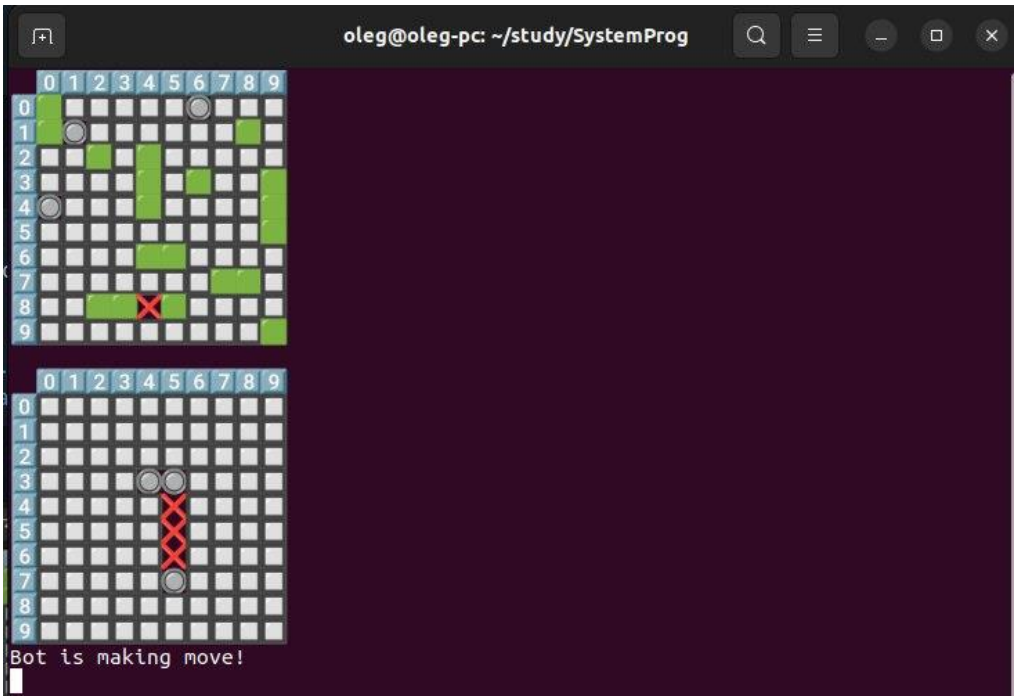

Рисунок 2 – Очередь хода бота

После генерации полей для игры сгенерированные поля записываются в соответствующие файлы: поле игрока – в файл player.txt, поле бота – в файл bot.txt. Запись полей происходит с помощью специальных символов: символ «-» обозначает пустую клетку, символ «*» – клетку с «кораблем» (рисунок 3).

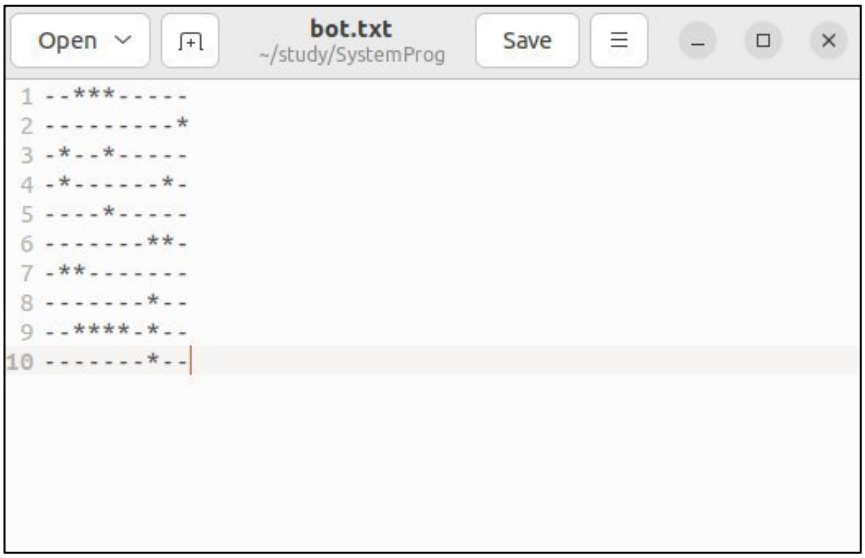

Рисунок 3 – Файл с записанным полем

Игра заканчивается, если на одном из полей все «корабли» будут уничтожены. В таком случае высветится соответствующее сообщение (рисунок 4).

Рисунок 4 – Сообщение об окончании игры

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была изучена оболочка shell, а также написаны скрипты, реализующие консольное приложение, которое имеет функционал игры «Морской бой» и может записывать данные игры в файлы для последующего анализа имеющихся данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Оболочка пользователя (shell) в Linux [Электронный ресурс]. – Режим доступа: https://white55.ru/shells.html – Дата доступа: 12.02.2024.

[2] О разных командных оболочках Linux [Электронный ресурс]. – Режим доступа: https://habr.com/ru/articles/157283/ – Дата доступа: 12.02.2024.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг кода

Листинг 1 – Файл lab1.sh

```
#!/bin/bash

# Размер поля
size=10
ship_arr=(4 3 3 2 2 2 1 1 1 1)
num_arr=(0 1 2 3 4 5 6 7 8 9)

# Инициализация полей
declare -A my_field
declare -A bot_field

set_field() {
# Заполнение полей
for ((i=0; i<$size; i++)); do
      for ((j=0; j<$size; j++)); do
            my_field[$i,$j]="-"
            bot_field[$i,$j]="-"
      done
done

ships=0
while [[ $ships -lt $size ]]; do
      x=$((RANDOM % size))
      y=$((RANDOM % size))
      dir=$((RANDOM % 4))

      x_tmp=$x
      y_tmp=$y

      is_possible=1

      for ((i=0; i<ship_arr[ships]; i++)); do
            if [[ "$x" -lt 0 || "$x" -gt 9 || "$y" -lt 0 || "$y" -gt 9
]]; then
                  is_possible=0
            fi

            if [[ "${my_field[$x,$y]}" == "*" ||
                  "${my_field[$(($x+1)),$y]}" == "*" ||
                  "${my_field[$(($x-1)),$y]}" == "*" ||
                  "${my_field[$x,$(($y+1))]}" == "*" ||
                  "${my_field[$(($x+1)),$(($y+1))]}" == "*" ||
                  "${my_field[$(($x-1)),$(($y+1))]}" == "*" ||
                  "${my_field[$x,$(($y-1))]}" == "*" ||
                  "${my_field[$(($x+1)),$(($y-1))]}" == "*" ||
```

```
                        "${my_field[$(($x-1)),$(($y-1))]}" == "*" ]]; then
                        is_possible=0
                fi

                if [[ "$dir" == 0 ]]; then
                        x=$(($x+1))
                elif [[ "$dir" == 1 ]]; then
                        y=$(($y+1))
                elif [[ "$dir" == 2 ]]; then
                        x=$(($x-1))
                else
                        y=$(($y-1))
                fi
        done

        x=$x_tmp
        y=$y_tmp

        if [[ "$is_possible" == 1 ]]; then
                for ((i=0; i<ship_arr[ships]; i++)); do
                        my_field[$x,$y]="*"
                        if [[ "$dir" == 0 ]]; then
                                x=$(($x+1))
                        elif [[ "$dir" == 1 ]]; then
                                y=$(($y+1))
                        elif [[ "$dir" == 2 ]]; then
                                x=$(($x-1))
                        else
                                y=$(($y-1))
                        fi
                done
                ships=$(($ships+1))
        fi
done

bot_ships=0
while [[ $bot_ships -lt $size ]]; do
        x=$((RANDOM % size))
        y=$((RANDOM % size))
        dir=$((RANDOM % 4))

        x_tmp=$x
        y_tmp=$y

        is_possible=1

        for ((i=0; i<ship_arr[bot_ships]; i++)); do
                if [[ "$x" -lt 0 || "$x" -gt 9 || "$y" -lt 0 || "$y" -gt 9
]]; then
                        is_possible=0
                fi

                if [[ "${bot_field[$x,$y]}" == "*" ||
                        "${bot_field[$(($x+1)),$y]}" == "*" ||
```

```
                  "${bot_field[$(($x-1)),$y]}" == "*" ||
                  "${bot_field[$x,$(($y+1))]}" == "*" ||
                  "${bot_field[$(($x+1)),$(($y+1))]}" == "*" ||
                  "${bot_field[$(($x-1)),$(($y+1))]}" == "*" ||
                  "${bot_field[$x,$(($y-1))]}" == "*" ||
                  "${bot_field[$(($x+1)),$(($y-1))]}" == "*" ||
                  "${bot_field[$(($x-1)),$(($y-1))]}" == "*" ]]; then
                  is_possible=0
            fi

            if [[ "$dir" == 0 ]]; then
                  x=$(($x+1))
            elif [[ "$dir" == 1 ]]; then
                  y=$(($y+1))
            elif [[ "$dir" == 2 ]]; then
                  x=$(($x-1))
            else
                  y=$(($y-1))
            fi
      done

      x=$x_tmp
      y=$y_tmp

      if [[ "$is_possible" == 1 ]]; then
            for ((i=0; i<ship_arr[bot_ships]; i++)); do
                  bot_field[$x,$y]="*"
                  if [[ "$dir" == 0 ]]; then
                        x=$(($x+1))
                  elif [[ "$dir" == 1 ]]; then
                        y=$(($y+1))
                  elif [[ "$dir" == 2 ]]; then
                        x=$(($x-1))
                  else
                        y=$(($y-1))
                  fi
            done
            bot_ships=$(($bot_ships+1))
      fi
done

}

draw_field() {
clear
# Вывод полей
echo "  0 1 2 3 4 5 6 7 8 9"
for ((i=0; i<$size; i++)); do
      echo -n "${num_arr[$i]} "
      for ((j=0; j<$size; j++)); do
            if [[ "${my_field[$i,$j]}" == "*" ]]; then
                  echo -n "■"
            elif [[ "${my_field[$i,$j]}" == "-" ]]; then
```

```
                  echo -n "🔲"
            elif [[ "${my_field[$i,$j]}" == "." ]]; then
                  echo -n "🔘"
            elif [[ "${my_field[$i,$j]}" == "x" ]]; then
                  echo -n "❌"
            fi
      done
      echo
done

echo

echo "  0️⃣ 1️⃣ 2️⃣ 3️⃣ 4️⃣ 5️⃣ 6️⃣ 7️⃣ 8️⃣ 9️⃣"
for ((i=0; i<$size; i++)); do
      echo -n "${num_arr[$i]} "
      for ((j=0; j<$size; j++)); do
            if [[ "${bot_field[$i,$j]}" == "." ]]; then
                  echo -n "🔘"
            elif [[ "${bot_field[$i,$j]}" == "x" ]]; then
                  echo -n "❌"
            else
                  echo -n "🔲"
            fi
      done
      echo
done
}

write_fields_to_file() {
echo -n "" > player.txt
echo -n "" > bot.txt

for ((i=0; i<$size; i++)); do
      for ((j=0; j<$size; j++)); do
            echo -n "${my_field[$i,$j]}" >> player.txt
            echo -n "${bot_field[$i,$j]}" >> bot.txt
      done
      echo "" >> player.txt
      echo "" >> bot.txt
done
}

is_end_game() {
is_win=1
is_defeat=1

for ((i=0; i<$size; i++)); do
      for ((j=0; j<$size; j++)); do
            if [[ "${bot_field[$i,$j]}" == "*" ]]; then
                  is_win=0
            fi

            if [[ "${my_field[$i,$j]}" == "*" ]]; then
```

```
                  is_defeat=0
            fi
      done
done

if [[ "$is_win" == 1 ]]; then
      draw_field
      echo "YOU WON!"
      exit
fi
if [[ "is_defeat" == 1 ]]; then
      draw_field
      echo "YOU LOSE!"
      exit
fi
}

bot_move() {
is_end=0

while [[ "$is_end" == 0 ]]; do
      draw_field

      echo "Bot is making move!"
      sleep 1

      x_move=$((RANDOM % size))
      y_move=$((RANDOM % size))

      if [[ "${my_field[$x_move,$y_move]}" == "-" ]]; then
            is_end=1
            my_field[$x_move,$y_move]="."
      elif [[ "${my_field[$x_move,$y_move]}" == "*" ]]; then
            my_field[$x_move,$y_move]="x"
      fi

      is_end_game
done
}

my_move() {
is_continue=1

while [[ "$is_continue" == 1 ]]; do
      is_continue=0

      draw_field

      read -p "Your turn! Enter coordinates: " y_move x_move

      if [[ "${bot_field[$x_move,$y_move]}" == "-" ]]; then
            bot_field[$x_move,$y_move]="."
      elif [[ "${bot_field[$x_move,$y_move]}" == "*" ]]; then
            bot_field[$x_move,$y_move]="x"
```

```
            is_continue=1
        fi

        is_end_game
done
}

set_field
write_fields_to_file

while [[ 1 ]]; do
draw_field
my_move
bot_move
done
```