Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Катедра «Комп'ютерна інженерія та програмування»

ЗВІТ

про виконання лабораторної роботи №4

з навчальної дисципліни «Алгоритми та структури даних»

Варіант 5

Виконав студент:

Омельніцький Андрій Миколайович
Група: КН-1023б

Перевірив:

Старший викладач

Бульба Сергій Сергійович

Харків-2024

Зміст

1 Мета роботи **2**

2 Завдання **2**

3 Хід виконання **2**

3.1 Визначення структур даних 3

3.2 Приклад роботи програми 6

4 Висновки **7**

1 Мета роботи

Отримати та закріпити знання про внутрішнє подання інтегрованих структур даних у мовах програмування.

Теми для попередньої роботи:

- фізичне та логічне подання даних;
- інтегровані типи даних;
- вирівнювання даних.

2 Завдання

Написати програму, яка виводить на екран внутрішнє подання структури з варіантною частиною та з бітовими полями, а також масиву структур. Перелік властивостей та відповідні типи полів для об’єктів з табл. 4.1 обрати за своїм розсудом.

Дослідити, як виконується вирівнювання даних та полів структури.

Порівняти час доступу до даних з вирівнюванням та за умови відсутності вирівнювання. За результатами роботи підготувати звіт з лабораторної роботи, де навести отримані результати та дати щодо них пояснення, зробити висновки.

N	Об’єкт	Тип об’єкта	Стан (так/ні)
5	Гра	комп’ютерна, настільна	– для дітей, – колективна

Рис. 1. Завдання за варіантом (5)

3 Хід виконання

Для виконання завдання було обрано мову Rust. Увесь код також додатково був розміщений в GitHub репозитарії: https://github.com/blackgolyb/algos-labs.

3.1 Визначення структур даних

Для завдання визначимо структури для дослідження. Також скористаємося макросом для `impl_show_bytes` того, щоб відображати структуру в байтовому вигляді.

А також розглянемо як в Rust можна контролювати вирівнювання даних. Для цього в Rust використовується *repr*. Для того, щоб досягти врівнювання як в C треба використати `#[repr(C)]`, а для того, щоб повністю позбутися вирівнювання треба використати `#[repr(C, packed(1))]`.
Код програми:

```
use std::mem::ManuallyDrop;

use crate::libs::bytes::{impl_show_bytes,       show, ShowBytes};

pub struct StandardPc  {
    pub filed1:   u8,
    pub filed2:   i32,
    pub filed3:   u8,
}

pub struct StandardTable  {
    pub filed1:   u8,
    pub filed2:   i32,
}

pub union StandardUnion {
    pub pc:  ManuallyDrop<StandardPc>,
    pub table:   ManuallyDrop<StandardTable>,
}

pub struct Standard  {
    pub kind:  StandardUnion,
    pub for_child:   bool,
    pub cooperative:   bool,
}
impl_show_bytes!(Standard);
```

Тепер реалізуємо код який буде робити порівняння всіх типів структур: з вирівнюванням, без вирівнювання, а також додатково розглянемо стандартну структуру Rust.
Код програми:

```
use std::mem::ManuallyDrop;
use std::time::Instant;

use super::variants::aligned::{Aligned,       AlignedPc,  AlignedTable,  AlignedUnion};
use super::variants::packed::{Packed,      PackedPc, PackedTable,  PackedUnion};
use super::variants::sandard::{Standard,       StandardPc,  StandardTable,   StandardUnion};
use crate::libs::bytes::ShowBytes;

macro_rules! create_pc  {
    ($st:ident,    $u:ident,    $pc:ident)   => {
        $st  {
            for_child:   false,
            cooperative:   true,
            kind:  $u {
                pc:  ManuallyDrop::new($pc {
                    filed1:   12,
                    filed2:   !0,
                    filed3:   26,
                }),
            },
        }
    };
```

```
}

macro_rules! create_table {
    ($st:ident, $u:ident, $table:ident) => {
        $st {
            for_child: false,
            cooperative: true,
            kind: $u {
                table: ManuallyDrop::new($table {
                    filed1: 12,
                    filed2: !0,
                }),
            },
        }
    };
}

macro_rules! display {
    ($t:ident, $pc:ident, $table:ident) => {
        println!("{:=^30}", stringify!($t));
        println!("size␣of␣{}", size_of::<$t>());
        print!("PC␣bytes␣representation:␣");
        $pc.show_bytes();
        println!();
        print!("␣␣␣␣for_child:␣");
        $pc.for_child.show_bytes();
        println!();
        print!("␣␣␣␣cooperative:␣");
        $pc.cooperative.show_bytes();
        println!();
        unsafe {
            let f1 = $pc.kind.pc.filed1;
            let f2 = $pc.kind.pc.filed2;
            let f3 = $pc.kind.pc.filed3;
            print!("␣␣␣␣filed1:␣");
            f1.show_bytes();
            println!();
            print!("␣␣␣␣pc␣filed2:␣");
            f2.show_bytes();
            println!();
            print!("␣␣␣␣pc␣filed3:␣");
            f3.show_bytes();
            println!();
        }

        println!();

        print!("Table␣bytes␣representation:␣");
        $table.show_bytes();
        println!();
        print!("␣␣␣␣for_child:␣");
        $pc.for_child.show_bytes();
        println!();
        print!("␣␣␣␣cooperative:␣");
        $pc.cooperative.show_bytes();
        println!();
        unsafe {
            let f1 = $pc.kind.table.filed1;
            let f2 = $pc.kind.table.filed2;
            print!("␣␣␣␣filed1:␣");
            f1.show_bytes();
            println!();
            print!("␣␣␣␣table␣filed2:␣");
            f2.show_bytes();
            println!();
        }

        let n = 10000000;

        let now = Instant::now();
        for _ in 0..n {
            unsafe {
                $pc.kind.pc.filed3;
            }
            $pc.for_child;
            $pc.cooperative;
```

```
99          }
100         println!("PC␣time:␣{}␣ns␣␣repeats:␣{n}", now.elapsed().as_nanos().to_string());
101
102         let now = Instant::now();
103         for _ in 0..n {
104             unsafe {
105                 $table.kind.table.filed2;
106             }
107             $table.for_child;
108             $table.cooperative;
109         }
110         println!("Table␣time:␣{}␣ns␣␣repeats:␣{n}", now.elapsed().as_nanos().to_string());
111         println!("{:=^30}\n", "");
112     };
113 }
114
115 pub fn main() {
116     let packed_pc = create_pc!(Packed, PackedUnion, PackedPc);
117     let packed_table = create_table!(Packed, PackedUnion, PackedTable);
118
119     let aligned_pc = create_pc!(Aligned, AlignedUnion, AlignedPc);
120     let aligned_table = create_table!(Aligned, AlignedUnion, AlignedTable);
121
122     let sandard_pc = create_pc!(Standard, StandardUnion, StandardPc);
123     let sandard_table = create_table!(Standard, StandardUnion, StandardTable);
124
125     display!(Packed, packed_pc, packed_table);
126     display!(Aligned, aligned_pc, aligned_table);
127     display!(Standard, sandard_pc, sandard_table);
128 }
```

3.2 Приклад роботи програми

Для кращого розуміння зображено на картинках відповідність між байтами кольором. Білим кольором позначено невикористовувану пам’ять.

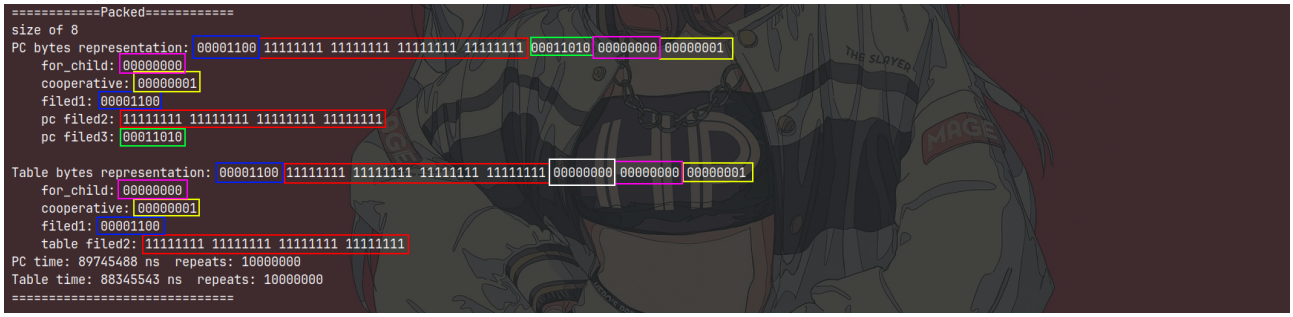

Рис. 2. Приклад роботи для структури без вирівнювання

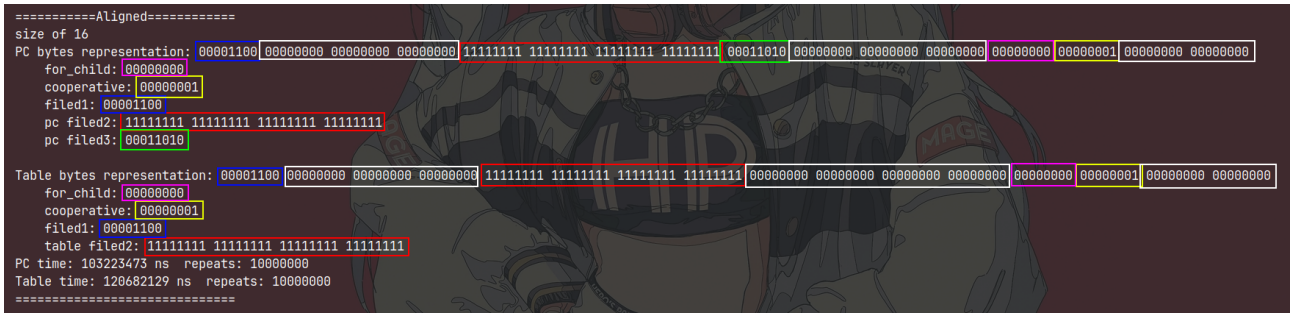

Рис. 3. Приклад роботи для структури з вирівнювання як в C

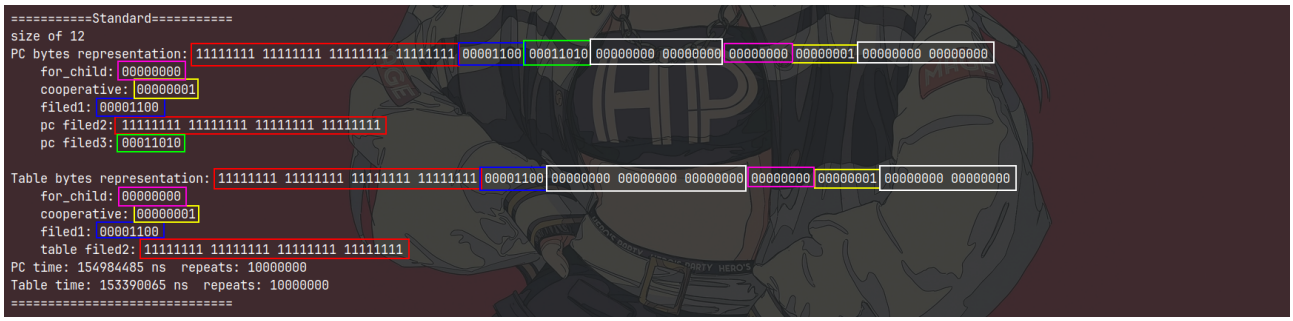

Рис. 4. Приклад роботи для структури з вирівнювання як в Rust

4 Висновки

В ході виконання лабораторної робити було порівняно бітове подання структур, а також час доступу до їх полів. Після багаторазового порівняння часу звертання до полів вирівняних та не вирівняних структур, не було помічено якоїсь різниці з поправкою на похибку.