1. Сигналы и слоты

Слот (Slot<Args...>) — это абстрактный вызываемый объект вида void (Args...), передающийся в качестве параметра

Виды Слотов:

- Указатель на Функцию

```
template <typename...Args>
struct FunctionDelegator{
        void (*Function) (Args...);
};
```

- Указатель на функцию член класса и экземпляр класса

```
template <typename...Args>
struct ClassDelegator{
        SomeClass * Reciever;
        void (SomeClass::*Function) (Args...);
};
```

- Лямбда функция (Тело лямюды и указатель на функцию оператора вызова)

```
template <typename...Args>
struct LambdaDelegator{
        LambdaBody_type LambdaBody;
        void (LambdaClass::*Function) (Args...);
};
```

Задание (Task<Args...>) — это Слот и аргументы, с которвми он должен быть вызван.

Сигнал (Signal) — это лист из Слотов, которые вызываются при наступлении события, вызывающего сигнал. (Signal<Args...>) Может быть 2 вида сигналов:

- Прямой (Direct) — Вызов сигнала приводит непосредственно к вызову слота

- Запланированный (Sheduled) — вызов сигнала составляет Task, который добавляется в цикл обработки событий.

Эквивалентность Task<Args...> и Slot<>: Задание Task<Args...> эквивалетно Слоту типа Lambda. Функция вложения (гомоморфизм) будет следующей:

```
template <typename SlotType,typename...Args>
auto make_task(SlotType Slot,Args...args){
        return [=](){return Slot(args...);};
}
```

Система Сигналов и слотов: Для реализации системы сигналов и слотов необходимо:

- Хранить все соединенные слоты
- Хранить все задания (Task)
- иметь очередь заданий
- цикл обработки событий: примерная реализация следующая:

```
while(!error_occured){
        if(!task_queue.empty() ){
                call(task_queue.pop_first());
        }else{
                sleep();
        }
}
```

Если для хранения очереди заданий достаточно статического массива необходимой длины, то для хранение Слотов нужна динамическая аллокация. Однако для систем с малой памятью.

Возможно, наилучший способ — иметь единое пространство для линамической аллокации, не подверженное фпагментации.

1.1. структура microset

Для динамической аллокации будем использовать страницы размером N байт. Соответствующая структура эквивалетна классу

```
T = std::array<char,N>
```

Для управления страницами будем использовать битовую маску. В итоге получается структура памяти

```
template <typename T,size_t _size>
struct micro_set{
        T data[_size];
        bitmask_t mask;
};
```

Для выделения памяти необходимо найти минимальный ненулевой бит (1 = ячейка свободна) в mask и вернуть его номер и присвоит ему 0, а для освобождения памяти необходимо соответствующий бит вернуть в состояние 1.

Для реализации битовой маски неоюходимо иметь класс целых чисел с длиной compile-time.

```
template <typename size_type,size_t length>
struct bitmask_type;
```

Для осуществления контроля памяти необходимо уметь:

- искать как можно быстрее единичный бит.
- выставлять бит в 0 или в 1.

Битовая маска	1	0	0	0	1