36. Списки из ядра операционной системы Linux (списки Беркли).
План ответа:
???	1. идеи реализации (циклический двусвязный список, интрузивный список,универсальный список);
???	2. описание типа;
	3. добавление элемента в начало и конец (list_add, list_add_tail), итерирование по списку
		(list_for_each, list_for_each_entry), освобождение памяти (list_for_each_safe);
	4. особенности реализации макроса list_entry.

1. идеи реализации (циклический двусвязный список, интрузивный список,универсальный список);
	Список Беркли – это циклический двусвязный список, в основе которого лежит следующая структура:

```
	struct list_head
	{
	    struct list_head *next, *prev;
	};
```

	В отличие от обычных списков, где данные содержатся в элементах списка, структура list_head должна быть частью сами данных
	Для оптимизации работы над списками их делают кольцевыми, то есть head->prev == last && last->next == head.
	Поскольку двусвязный кольцевой список не имеет настоящих начала и конца, операции создания списка, добавления и удаления элементов имеют сложность O(1), а очистка списка, как обычно, O(N).

2. описание типа;

3. добавление элемента в начало и конец (list_add, list_add_tail), итерирование по списку
	(list_for_each, list_for_each_entry), освобождение памяти (list_for_each_safe);

```
	static inline void __list_add(struct list_head *new, struct list_head *prev, struct list_head *next)
	{
		next->prev = new;
		new->next = next;
		new->prev = prev;
		prev->next = new;
	}

	static inline void list_add(struct list_head *new, struct list_head *head)
	{
		__list_add(new, head, head->next);
	}

	static inline void list_add_tail(struct list_head *new, struct list_head *head)
	{
		__list_add(new, head->prev, head);
	}

	#define list_for_each(pos, head) \
	for (pos = (head)->next; pos != (head); pos = pos->next)


	#define list_for_each_entry(pos, head, member) \
		for (pos = list_entry((head)->next, typeof(*pos), member); \
		     &pos->member != (head); \
		     pos = list_entry(pos->member.next, typeof(*pos), member))

	#define list_for_each_safe(pos, n, head) \
		for (pos = (head)->next, n = pos->next; pos != (head); \
			pos = n, n = pos->next)
```

4. особенности реализации макроса list_entry.

```
#define list_entry(ptr, type, member) \
container_of(ptr, type, member)

#define container_of(ptr, type, field_name) ( \
   (type *) ((char *) (ptr) - offsetof(type, field_name)))

#define offsetof(TYPE, MEMBER) \
     ((size_t) &((TYPE *)0)->MEMBER)
```