Списки в стиле университета Беркли (вариант реализации из ядра Linux)

Списки Беркли: реализация

Текущая реализация:

https://github.com/torvalds/linux/blob/master/include/linux/list.h

Используемая реализация есть в примерах. Взята из какой-то версии ядра и немного «адаптирована».

Отличия между используемой реализацией и текущей предлагается найти самостоятельно.

Списки Беркли: идея

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

```
struct data
{
    int i;
    struct list_head list;
    ...
};
```

Списки Беркли: описание

```
#include “list.h”

struct data
{
    int num;
    struct list_head list;
};
```

Следует отметить следующее:

- Структуру struct list_head можно поместить в любом месте в определении структуры.
- struct list_head может иметь любое имя.
- В структуре может быть несколько полей типа struct list_head.

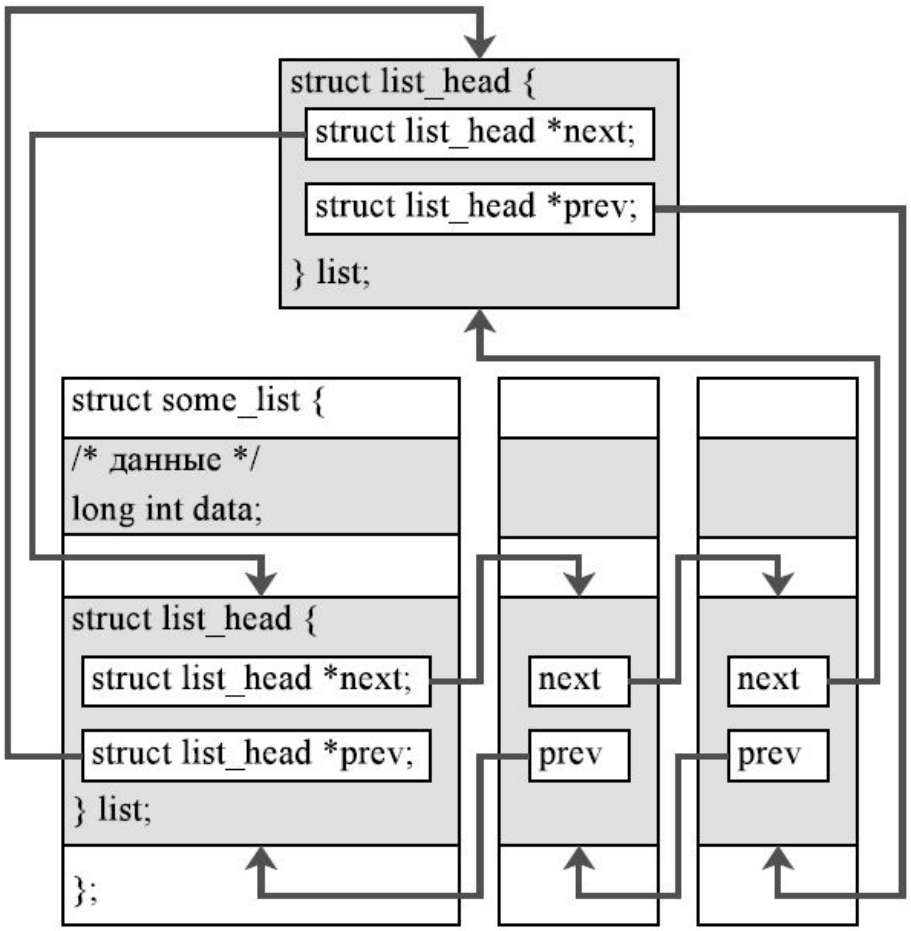
Списки Беркли: создание «головы»

```
#define LIST_HEAD_INIT(name) { &(name), &(name) }

#define LIST_HEAD(name) \
    struct list_head name = LIST_HEAD_INIT(name)

static inline void INIT_LIST_HEAD(struct list_head *list)
{
    list->next = list;
    list->prev = list;
}
```

Списки в стиле Беркли



Списки Беркли: добавление

```
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);
}
```

Списки Беркли: обход

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

#define list_for_each_prev(pos, head) \
    for (pos = (head)->prev; pos != (head); pos = pos->prev)

#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)
```

Списки Беркли: 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)
```

offsetof: идея

```
struct s
{
    char c;
    int i;
    double d;
};

...

printf("offset of i is %d\n", offsetof(struct s, i));
```

В нашем случае TYPE – struct s, MEMBER – i, size_t – unsigned int. После работы препроцессора получим

```
printf("offset of i is %d\n",
            (unsigned int) (&((struct s*) 0)->i));
```

offsetof: идея

```
int offset = (int) (&((struct s*) 0)->i);
```

- ((struct s*) 0)
  - Приводим число ноль к указателю на структуру s. Эта строчка говорит компилятору, что по адресу 0 располагается структура, и мы получаем указатель на нее.
- ((struct s*) 0)->i
  - Получаем поле i структуры s. Компилятор думает, что это поле расположено по адресу 0 + смещение i.
- &((struct s*) 0)->i
  - Вычисляем адрес поля i, т.е. смещение i в структуре s.
- (unsigned int) (&((struct s*) 0)->i)
  - Преобразовываем адрес члена i к целому числу.

Списки в стиле Беркли: анализ «+» и «-»

+ **Одно выделение памяти на узел списка.**

- **Независимо от того в списке узел или нет присутствуют два дополнительных указателя.**