# 자료구조 (Data Structure)
# 11주차: 해시 테이블

# 지난 시간 요약

- direct address table: 키를 배열의 인덱스로 씀
- hash table: 해시값을 배열의 인덱스로 씀
- 해시값 충돌 해결 방법
  - 링크드 리스트
  - 다음 주소 계산

# 리눅스 커널에서 구현된 해시 테이블

include/linux/hash.h
- 해시 함수

include/linux/type.h
- 링크드리스트 구조체

include/linux/list.h
- 링크드리스트

include/linux/hashtable.h
- 해시 테이블

# 해시 함수

$h: \mathrm{U} \rightarrow \{0, 1, \ldots, m-1\}$

- 목표: key에 패턴이 있어도 해시값에는 패턴이 없다

$h(k) = m * (k\alpha$의 소수 부분$)$의 정수 부분

- $\alpha$로 가장 적당한 값은?

# 해시 함수

```
/* Although a random odd number will do,
  * it turns out that the golden ratio
  * phi = (sqrt(5)-1)/2, or its negative, is nice.
  * (See Knuth vol 3, section 6.4, exercise 9.)
  * This is the negative, (1 - phi) = phi**2 = (3 - sqrt(5))/2
  */

#define GOLDEN_RATIO 0x61C88647
```

# 해시 함수

$h(k) = m * (k\alpha$의 소수 부분)의 정수 부분
$= k * (m * \alpha)$을 m으로 나눈 나머지

m이 2의 지수일 경우,
$h(k) = k * (2^{32} * \alpha)$를 m으로 나눈 나머지

# 해시 함수

```
/* This exist only so lib/test_hash.c can test
    the arch-optimized versions with the generic. */
#ifndef HAVE_ARCH__HASH_32
#define __hash_32  __hash_32_generic
#endif
static inline u32 __hash_32_generic(u32 val)
{
        return val * GOLDEN_RATIO_32;
}
```

# 해시 함수

m이 2의 지수일 경우,

$h(k) = k * (2^{32} * \alpha)$를 m으로 나눈 나머지

$= k * (2^{32} * \alpha) \bmod 2^{32}$에서 비트 선택

# 해시 함수

```
static inline u32 hash_32(u32 val, unsigned int bits)
{
        /* High bits are more random, so use them. */
        return __hash_32(val) >> (32 - bits);
}
```

# 링크드 리스트 구조체

링크드 리스트 노드 구조체

- 다음 노드와 이전 노드의 정보가 있다.

# 링크드 리스트 구조체

```
struct hlist_node {
        struct hlist_node *next, **pprev;
};
```

# 링크드 리스트 구조체

링크드 리스트 구조체

- 첫번째 노드의 정보가 있다.

# 링크드 리스트 구조체

```
struct hlist_head {
        struct hlist_node *first;
};
```

# 링크드 리스트 구조체

링크드 리스트 구조체, 노드 구조체의 초기화

- 모두 NULL로 초기화한다.

# 링크드 리스트 구조체

```
#define INIT_HLIST_HEAD(ptr) ((ptr)->first = NULL)

static inline void INIT_HLIST_NODE(struct
hlist_node *h)
{
        h->next = NULL;
        h->pprev = NULL;
}
```

# 링크드 리스트

링크드 리스트에서 노드 추가하기

- 새로운 노드의 앞뒤 연결을 설정한다.
- 기존 노드의 앞뒤 연결을 재설정한다.

# 링크드 리스트

```
/**
 * hlist_add_head
 * - add a new entry at the beginning of the hlist
 * @n: new entry to be added
 * @h: hlist head to add it after
 *
 * Insert a new entry after the specified head.
 * This is good for implementing stacks.
 */
```

# 링크드 리스트

```
static inline void hlist_add_head(struct hlist_node *n,
struct hlist_head *h)
{
        struct hlist_node *first = h->first;
        WRITE_ONCE(n->next, first);
        if (first)
                WRITE_ONCE(first->pprev, &n->next);
        WRITE_ONCE(h->first, n);
        WRITE_ONCE(n->pprev, &h->first);
}
```

# 링크드 리스트

링크드 리스트에서 노드 삭제하기

- 이웃 노드의 앞뒤 연결을 재설정한다.
- 삭제될 노드의 앞뒤 연결을 재설정한다.

# 링크드 리스트

```
static inline void __hlist_del(struct hlist_node *n)
{
        struct hlist_node *next = n->next;
        struct hlist_node **pprev = n->pprev;

        WRITE_ONCE(*pprev, next);
        if (next)
                WRITE_ONCE(next->pprev, pprev);
}
```

# 링크드 리스트

```
/**
 * hlist_del_init - Delete the specified hlist_node
 *                  from its list and initialize
 * @n: Node to delete.
 *
 * This function leaves the node in unhashed state.
 */
```

# 링크드 리스트

```
static inline void hlist_del_init(struct hlist_node *n)
{
        if (!hlist_unhashed(n)) {
                __hlist_del(n);
                INIT_HLIST_NODE(n);
        }
}
```

# 링크드 리스트

```
/**
 * hlist_unhashed - Has node been removed from list
 *                  and reinitialized?
 * @h: Node to be checked
 *
 * Not all removal functions will leave a node in
 * unhashed state. hlist_nulls_del_init_rcu() does leave
 * the node in unhashed state, but hlist_nulls_del()
 * does not.
 */
```

# 링크드 리스트

```
static inline int hlist_unhashed(const struct
hlist_node *h)
{
        return !h->pprev;
}
```

# 링크드 리스트

```
static inline void INIT_HLIST_NODE(struct
hlist_node *h)
{
        h->next = NULL;
        h->pprev = NULL;
}
```

# 링크드 리스트

링크드 리스트 노드 사용법

- 데이터 구조체의 멤버로 노드를 쓴다.
- 노드로 부터 원래 구조체 주소를 역산한다.

# 링크드 리스트

```
struct fox {
        char data;
        struct hlist_node node;
};
```

# 해시 테이블

해시 테이블 선언

- 링크드 리스트의 배열로 선언한다

# 해시 테이블

```
#define DECLARE_HASHTABLE(name, bits)                   \
        struct hlist_head name[1 << (bits)]
```

# 해시 테이블

해시 테이블 초기화

- 첫번째 데이터들을 모두 NULL로 설정한다

# 해시 테이블

```
static inline void __hash_init(struct hlist_head *ht,
unsigned int sz)
{
        unsigned int i;

        for (i = 0; i < sz; i++)
                INIT_HLIST_HEAD(&ht[i]);
}
```

# 해시 테이블

해시 테이블에 데이터 추가

- 해시값으로 링크드리스트를 찾아서 추가한다

# 해시 테이블

```
/**
 * hash_add - add an object to a hashtable
 * @hashtable: hashtable to add to
 * @node: &struct hlist_node of the object to be added
 * @key: the key of the object to be added
 */

#define hash_add(hashtable, node, key)                  \
        hlist_add_head(node, &hashtable[                \
        hash_32(key, HASH_BITS(hashtable))])
```

# 해시 테이블

```
int ilog2(unsigned int v)
{
    int l = 0;
    while ((1U << l) < v)
        l++;
    return l;
}
#define ARRAY_SIZE(arr) (sizeof(arr) / sizeof(arr[0]))
#define HASH_SIZE(name) (ARRAY_SIZE(name))
#define HASH_BITS(name) ilog2(HASH_SIZE(name))
```

# 요약

- 해시함수
- 링크드리스트
- 해시테이블

# 기말고사 기출문제

- Bucket이 11개, bucket 당 slot이 1개인 hash table
- Hash function : $h(k) = k \bmod 11$
- 입력값: 12, 44, 13, 88, 23, 94, 11, 39, 20, 16

1) 언제 overflow가 발생?

2) linear probing에서, 11 추가시 bucket을 몇 개 점검?

3) quadratic probing에서 11 추가, 몇 개 bucket 점검?

4) 26을 탐색할 때, 몇 개의 bucket을 점검해야 하는가?