HW9. Linked List 구현하기

(Blocking / Non-blocking)

학번: 20170404

이름: 한종수

제출일: 2021. 6. 1

1.과제 개요

Linked List 의 두 가지 버전을 구현하고 성능평가를 수행한다.

2.소스코드

(수정한 코드를 캡처하고 간단히 설명함)

1) linked_list_init

```
int linked_list_init(linked_list_t **ll)
{
        if (ll == NULL)
        {
                printf("linked list pointer error \n");
                return -1;
        }

        /* do program */
        /* allocate linked list */
        /* initialize list head */
        /* initialize mutex lock for BLINKED_LIST version */

        (*ll) = (linked_list_t *)malloc(sizeof(linked_list_t));
        (*ll)->list_head = (node_t *)malloc(sizeof(node_t));
        (*ll)->list_head->key = 0;
        (*ll)->list_head->value = 0;
        (*ll)->list_head->level = 0;
        (*ll)->list_head->next = NULL;

#ifdef BLINKED_LIST
        int rc = pthread_mutex_init(&((*ll)->list_lock), NULL);
        assert(rc == 0);
#endif
        srand((unsigned)time(NULL));

        return 0;
}
```

(*ll) 에 linked_list_t를 할당받아 linked_list_t 를 만든다. 그리고 list_head 를 동적할당받아 linked_list에 연결한다. 그리고 헤드의 값을 초기화한다. 그리고 BLINKED_LIST 옵션이 들어올 때 lock을 초기화시킨다.

2) linked_list_destroy

```c
void linked_list_destroy(linked_list_t *ll)
{
        /* do program */
        /* free all nodes in the linked list */
        /* free the linked list */

        node_t **ptr;
        node_t **tmp;
        node_t *_tmp;

        ptr = ll->list_head->next;

        while (ptr != NULL)
        {
                tmp = ptr;
                _tmp = *ptr;
                ptr = (*ptr)->next;

                free(_tmp);
                free(tmp);
        }

        free(ll->list_head);
        free(ll);

        return;
}
```

list 헤드에서부터 next를 따라가며 동적할당 한 것을 해제한다. 링크드 리스트 next에 연결된 모든 노드들을 할당 해제하고 list_head, ll 을 할당 해제한다.

```c
long linked_list_get(long key, linked_list_t *ll)
{
        /* do program */
        /* if key is found, return value */
        /* if key is not found, return -1 */

        node_t **ptr;

        ptr = &(ll->list_head);
        while (ptr != NULL)
        {
                if ((*ptr)->key != key)
                {
                        ptr = (*ptr)->next;
                        continue;
                }

                return (*ptr)->value;
        }

        return -1;
}
```

3) linked_list_get

key가 주어지면 list_head node 부터 next를 들어가며 node의 key와 인자로 들어온 key를 비교하여 일치하는 경우 value를 리턴하고, 찾지 못했으면 -1을 리턴한다.

4) linked_list_put

```
long linked_list_put(long key, long value, linked_list_t *ll)
{
        /* do program */
        /* if succeeds, return 0 */
        /* if fails, return -1 */

#ifdef BLINKED_LIST
        // lock
        pthread_mutex_lock(&(ll->list_lock));

        node_t **ptr;
        node_t **tmp;

        ptr = ll->list_head->next;

        tmp = (node_t **)malloc(sizeof(node_t *));
        if (tmp == NULL)
        {
                return -1;
        }
        *tmp = (node_t *)malloc(sizeof(node_t));
        if (*tmp == NULL)
        {
                return -1;
        }
        (*tmp)->key = key;
        (*tmp)->value = value;
        (*tmp)->level = 0;
        (*tmp)->next = ptr;

        (ll->list_head->next) = tmp;

        // unlock
        pthread_mutex_unlock(&(ll->list_lock));
#else
        while (1)
        {
                node_t **ptr;
                node_t **tmp;

                ptr = ll->list_head->next;

                tmp = (node_t **)malloc(sizeof(node_t *));
                if (tmp == NULL)
                {
                        return -1;
                }
                *tmp = (node_t *)malloc(sizeof(node_t));
                if (*tmp == NULL)
                {
                        return -1;
                }
                (*tmp)->key = key;
                (*tmp)->value = value;
                (*tmp)->level = 0;
                (*tmp)->next = ptr;

                if (CAS(&(ll->list_head->next), ptr, tmp) == 1)
                {
                        break;
                }

                free(*tmp);
                free(tmp);
        }
#endif

        return 0;
}
```

BLINKED_LIST 옵션 시에 ll->list_lock 의 lock을 취득하는 방식으로 put 동작을 수행한다. 삽입시에 헤드의 바로 뒤편에 새 노드를 추가한다. 새 노드 할당 실패 시에 return -1을 실행한다. 성공적으로 삽입이 되면, 0을 리턴한다.

CAS를 수행하는 경우 대부분의 동작은 위의 경우와 같다. 다만 링크드 리스트의 헤드에 새로 만들어진 노드를 넣을 때 방식이 다르다. 새 노드를 할당하기전, ptr변수에 미리 헤드가 가리키는 다음 노드의 주소를 저장한다. 그리고 새 노드를 할당받아 초기화 시킨다. 이후 CAS 에 list_head->next의 주소값을 넘기고 미리 담아둔 ptr을 두번째 인자로 넘긴다. 그럼 CAS가 ptr과 현재 list_head->head의 값을 비교하여 일치하면 다른 스레드가 공유변수에 접근하지 않았다는 것이므로 list_head->next에 새로 할당한 노드를 넣고 1을 리턴한다. 만약 값이 다르면, 0을 리턴하고 다시 일치하는 경우가 나올때까지 반복문을 실행한다. 반복문을 돌때, 새로할당된 노드를 할당해제하여 메모리 누수를 줄인다.

3.결과

(테스트 실행결과를 캡처)

```
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab9_Linked_List/linked_list$ ./b_linked 16 1 100000
PUT_RAND ops: 100000 time(ms):      25.00 iops: 4000000.00
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab9_Linked_List/linked_list$ ./nb_linked 16 1 100000
PUT_RAND ops: 100000 time(ms):       9.00 iops: 11111111.11
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab9_Linked_List/linked_list$ ./b_linked 16 3 100000
GET_RAND ops: 100000 time(ms):    1209.00 iops:   82712.99
jongsoo@DESKTOP:~/2021_OperatingSystem/Lab9_Linked_List/linked_list$ ./nb_linked 16 3 100000
GET_RAND ops: 100000 time(ms):     504.00 iops:  198412.70
```