بسم تعالی

سیستم‌های عامل

تمرین هفتم

استاد:

دکتر حسین اسدی

نویسنده :

محمدهومان کشوری

شماره دانشجویی :

99105667

تمرینات عملی

سوال 1.

برای حل این سوال قسمت به قسمت جلو می‌رویم و توابع را توضیح می‌دهیم.

ابتدا باید لینک لیست و فایل مربوطه را بسازیم.

```
void insert(LinkedList_t *list, Element_t *element) {
    if (list->prev != NULL) {
        (list->prev)->next = element;
        element->prev = (list->prev);
        (list->prev) = element;
        element->next = list;
    } else {
        list->next = element;
        list->prev = element;
        element->next = list;
        element->prev = list;
    }
}
```

در تکه کد بالا قسمت insert لینک لیست است که در صورتی که عضو قبلی وجود نداشته باشد عضو جدیدی به لینک لیست اضافه کرده و در غیر این صورت لینک اخرین عضو را تغییر می‌دهد.

```
int delete(Element_t *element) {
    if (element->value == NULL) {
        return 1;
    } else {
        (element->next)->prev = element->prev;
        (element->prev)->next = element->next;
        free(element);
        return 0;
    }
}
```

در قطعه کد بالا تابع delete نشان داده شده و ارور همراه آن.

```
Element_t *lookup(LinkedList_t *list, const char *value) {
    if (list->next == NULL || list->next == list) {
        return NULL;
    }
    Element_t *found = list->next;
    while (found != list) {
        if (strcmp(found->value,value) == 0) {
            return found;
        }
        found = found->next;
    }
    return NULL;
}
```

در قطعه کد بالا نحوه پیدا کردن و lookup عنصر خاص در لینک لیست نشان داده شده

```
int get_length(LinkedList_t *list) {
    int ans = 0;
    if (list->next == NULL || list->next == list) {
        return 0;
    }
    Element_t *lili = list->next;
    while (lili != list->prev) {
        ans++;
        lili = lili->next;
    }
    ans++;
    return ans;
                                                                {
```

تابع بالا نیز برای گرفتن طول لینک لیست بکار می‌رود.

حال توابع main.c را توضیح می‌دهیم.

```
int parse_input(const char *arg, char eq) {
    char *quotPtr = strchr(arg, eq);
    if (quotPtr == NULL) {
    }
    int position = quotPtr - arg;
```

```
    char *attrValue = (char *) malloc((position + 1) * sizeof(char));
    memcpy(attrValue, arg, position);
    attrValue[position] = '\0';
    char *t = calloc(10, sizeof(char));
    char x = arg[position + 1];
    while (x != '\0') {
        t = strncat(t, &x, 1);
        position++;
        x = arg[position + 1];
    }
    int num = atoi(t);
    return num;
                                                                    {
```

اولین تابع ما parse_input است که عملا به عنوان command line argument parser به کار می‌رود و تعداد تردها و تکرارها را بدست می‌آورد.

```
void print_list() {
    Element_t *start = list->next;
    if (start == NULL) {
        printf("NO ITEMS IN LIST \n");
        return;
    }
    printf("------------------------ list
------------------------------\n");
    while (start != list) {
        printf("item with pointer number : %u and value : %s \n", (int)
(size_t) start, start->value);
        start = start->next;
    }
    printf("------------------------ end
------------------------------\n");
}
```

دومین تابع ما print_list است که اعضای لیست پیوندی را به همراه شماره پوینتر و مقدار آنها چاپ می‌کند.

```
Element_t *a = (Element_t *) calloc(1, sizeof(LinkedList_t));
    a->value = (char *) calloc(100 * 20, 200 * sizeof(char));
    a->value = "vc";
    Element_t *b = (Element_t *) calloc(1, sizeof(LinkedList_t));
    b->value = (char *) calloc(100 * 20, 200 * sizeof(char));
    b->value = "gc";
    Element_t *c = (Element_t *) calloc(1, sizeof(LinkedList_t));
    c->value = (char *) calloc(100 * 20, 200 * sizeof(char));
    c->value = "asdf";
    Element_t *d = (Element_t *) calloc(1, sizeof(LinkedList_t));
    d->value = (char *) calloc(100 * 20, 200 * sizeof(char));
    d->value = "dddd";
    insert(list, a);
    insert(list, b);
    insert(list, d);
```

در ابتدای امر ما 3 عنصر مورد نظر را به لیست پیوندی اضافه می‌کنیم.

حال در دو حالت تابع workerThread که به ازای هر ریسمان اجرا می‌شود را بررسی می‌کنیم و نتایج و نمودارهای مرتبط را رسم می‌کنیم.

در اولین حالت :

```
void *workerThread() {
    ssize_t readl;
    char *str = calloc(100, sizeof(char));
    sprintf(str, "%lu", pthread_self() % 10000);
    for (int i = 0; i < num_iterations; i++) {
        char x = 'a';
        str = strncat(str, &x, 1);
        Element_t *a = (Element_t *) calloc(1, sizeof(Element_t));
        a->value = (char *) calloc(100 * 20, 200 * sizeof(char));
        a->value = str;
        pthread_mutex_lock(&lock);
        insert(list, a);

        pthread_mutex_lock(&lock2);
        fprintf(result, "Log(insert(x)) -> thread %lu inserted %s \n",
pthread_self(), str);
        pthread_mutex_unlock(&lock2);
//        pthread_mutex_unlock(&lock);
```

```
        pthread_mutex_lock(&lock2);
        fprintf(result,
                "Log(get_length(x)) -> thread %lu : length is %d\n",
pthread_self(), get_length(list));
        pthread_mutex_unlock(&lock2);

        pthread_mutex_lock(&lock2);
        fprintf(result, "Log(lookup(x)) -> thread %lu found %s \n",
                pthread_self(), lookup(list, str)->value);
        pthread_mutex_unlock(&lock2);
//        pthread_mutex_lock(&lock);
        delete(a);

        pthread_mutex_lock(&lock2);
        fprintf(result, "Log(delete(x)) -> thread %lu deleted %s \n",
pthread_self(), str);
        pthread_mutex_unlock(&lock2);

        pthread_mutex_unlock(&lock);
    }
    free(str);
    return NULL;
}
```

در این حالت، اجرای کلی تمامی threadها درون یک لاک است به این منظور که در هنگامی که یک thread کار می‌کند، هیچ ترد دیگری نمی‌تواند از هیچ تابع دیگری استفاده کند. این روش از نظر امن بودن بسیار روش **امنی** است و می‌توان به قطعیت گفت هیچگونه مشکل **synchronization** و یا مشکل **deadlock** نداریم چرا که در هر قسمت یک ترد لاک مربوط به خود را گرفته، کار را انجام می‌دهد و لاک را به ترد بعدی می‌دهد.

تنها مشکل برنامه بالا **سرعت کند** اجرای آن است به این دلیل که عملا ما **همروندی** را فدای **امن** بودن برنامه کرده‌ایم.

حال اسکریپت زیر را نوشته، آنرا اجرا می‌کنیم و خروجی‌ها را در دایرکتوری outputs ذخیره می‌کنیم.

```
gcc main.c linked_list.c -o main.o
```

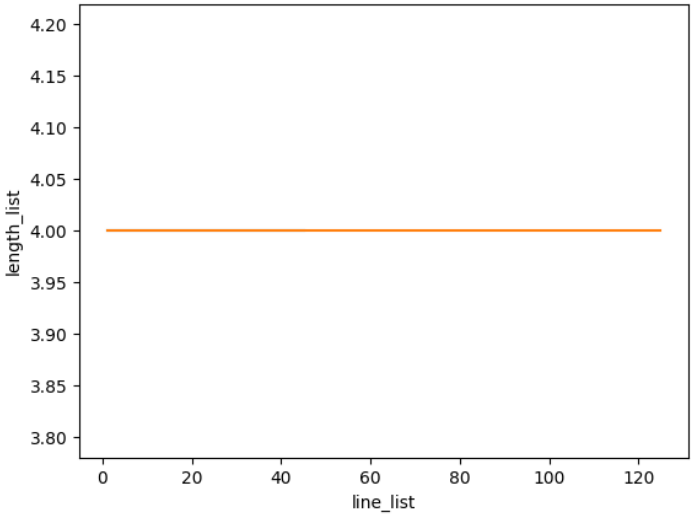
```
./main.o --num_threads=2 --num_iterations=6
mv outputs/result.txt outputs/result_1.txt
./main.o --num_threads=4 --num_iterations=8
mv outputs/result.txt outputs/result_2.txt
./main.o --num_threads=8 --num_iterations=10
mv outputs/result.txt outputs/result_3.txt
./main.o --num_threads=16 --num_iterations=12
mv outputs/result.txt outputs/result_4.txt
```

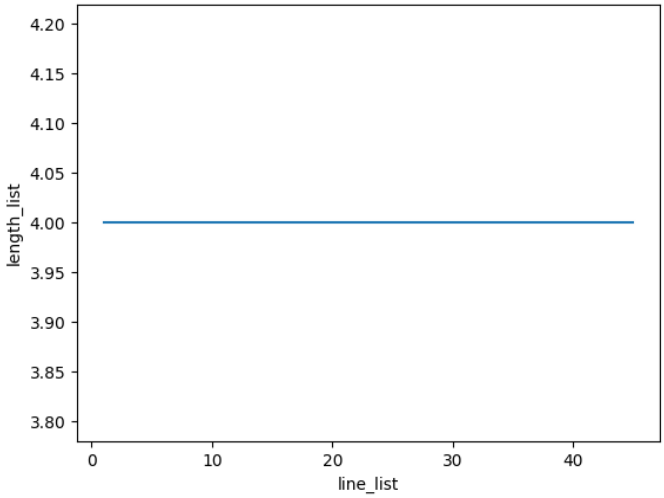
حال نتایج را با برنامه پایتون بررسی می‌کنیم.

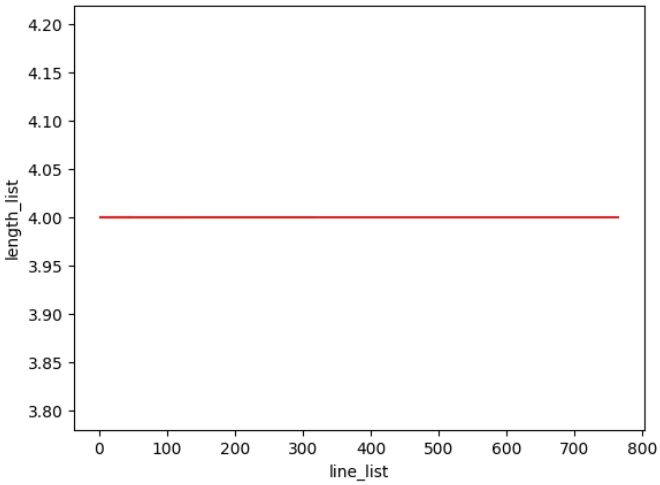
برنامه پایتون نوشته شده به صورت زیر است که نتایج را به صورت چارت رسم می‌کند.

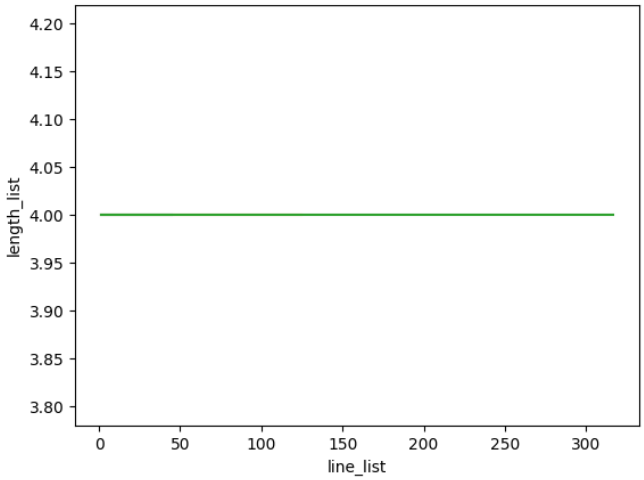
```python
import matplotlib.pyplot as plt
import os
import re
directory = 'outputs'

for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
    if os.path.isfile(f):
        line_list = []
        length_list = []
        file = open(f)
        for i, line in enumerate(file):
            if "length" in line:
                line_list.append(i)
                m = re.search('length is ([\d]+)', line)
                m.group(0)
                length_list.append(int(m.group(1)))
        plt.plot(line_list, length_list)
        plt.ylabel('length_list')
        plt.xlabel('line_list')
        # plt.show()
        plt.savefig(f'{directory}/{filename}.png')
```

حال نتایج را بررسی می‌کنیم.









نتایج بالا برای برنامه ما منطقی است چرا که بخاطر ساختار برنامه همواره فقط یک ترد درحال اجرا شدن است و بعد از insert عملیات delete را انجام می‌دهد که یعنی ما همواره فقط 4 عنصر در لینک لیست داریم.

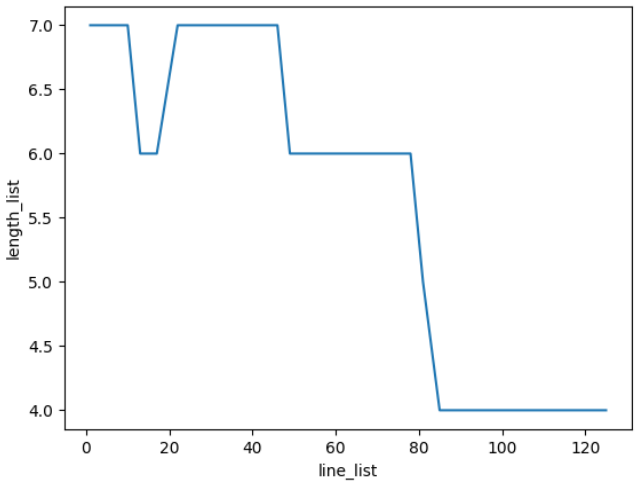
حال برنامه اصلی را کمی عوض می‌کنیم و کمی از امنیت آن می‌کاهیم و همروندی آن را افزایش می‌دهیم.

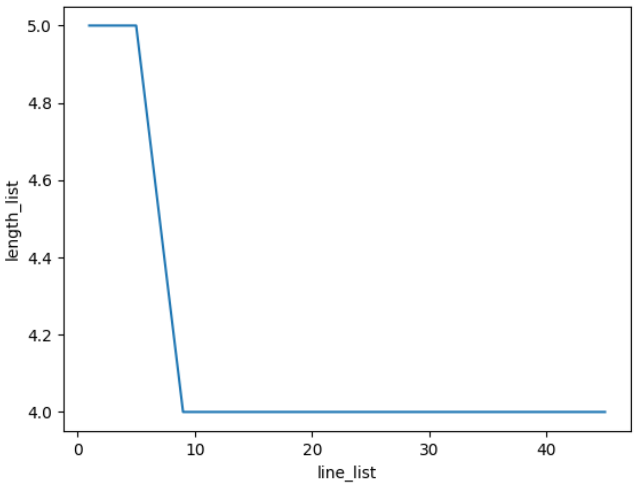
عکس‌ها و نتایج مربوطه در قسمت Safe است.

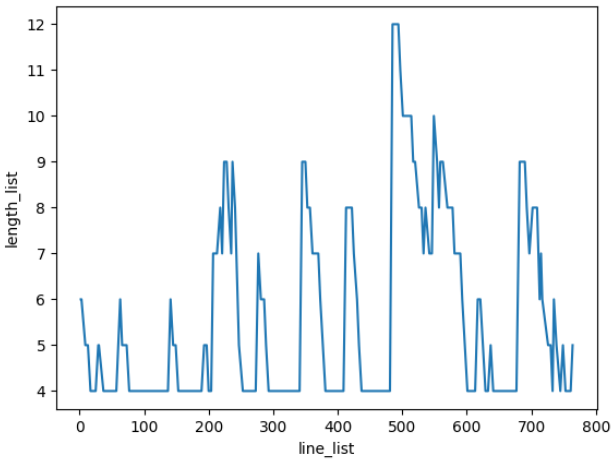
حال برنامه را کمی عوض می‌کنیم و لاک را فقط در قسمت insert و delete اعمال می‌کنیم و برای نوشتن‌ها نیز لاک قرار می‌دهیم.

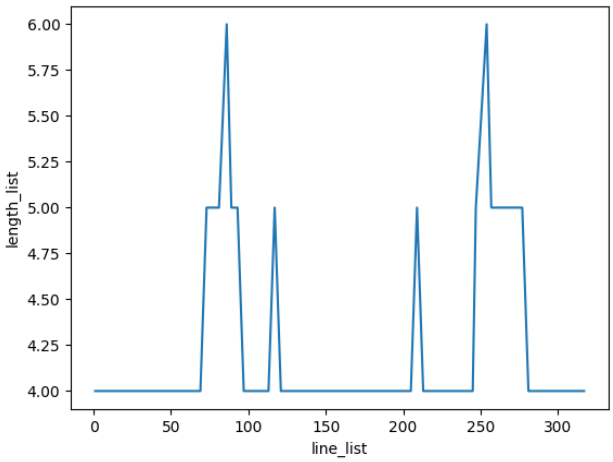
```
pthread_mutex_lock(&lock);
        insert(list, a);
        pthread_mutex_unlock(&lock);

        pthread_mutex_lock(&lock2);
        fprintf(result, "Log(insert(x)) -> thread %lu inserted %s \n",
pthread_self(), str);
        pthread_mutex_unlock(&lock2);
//        pthread_mutex_unlock(&lock);

        pthread_mutex_lock(&lock2);
        fprintf(result,
                "Log(get_length(x)) -> thread %lu : length is %d\n",
pthread_self(), get_length(list));
        pthread_mutex_unlock(&lock2);

        pthread_mutex_lock(&lock2);
        fprintf(result, "Log(lookup(x)) -> thread %lu found %s \n",
                pthread_self(), lookup(list, str)->value);
        pthread_mutex_unlock(&lock2);
        pthread_mutex_lock(&lock);
        delete(a);
        pthread_mutex_unlock(&lock);
        pthread_mutex_lock(&lock2);
        fprintf(result, "Log(delete(x)) -> thread %lu deleted %s \n",
pthread_self(), str);
        pthread_mutex_unlock(&lock2);
```

حال با برنامه پایتون نتایج را بررسی می‌کنیم.









نتایج بالا نوسانات درج و حذف از برنامه ما را نشان می‌دهد.

این برنامه امنیت برنامه قبلی را ندارد اما بسیار سریع‌تر و بهتر عمل می‌کند.

در نهایت تمامی کدها ضمیمه شده‌اند و نتایج و عکس‌های مربوط به هر دو اجرا قرار داده شده‌اند.

کد برنامه امن‌تر نیز ضمیمه شده است.