Test Plan For MP4

Student: Yulin Xiao

CUID: C16278133

Student UserName: Yulinx

TEST Plan:(See MP4 lab4.c)

1.Do the Unit test 0 and 1 in default(Head_First) and (Rover_First) Method
and using -f best, first, and worst to test the correctness of the every Search method.
2.Do the every unit test 2,3,4 to the all kinds of the search method, the unit test plan is shown below:

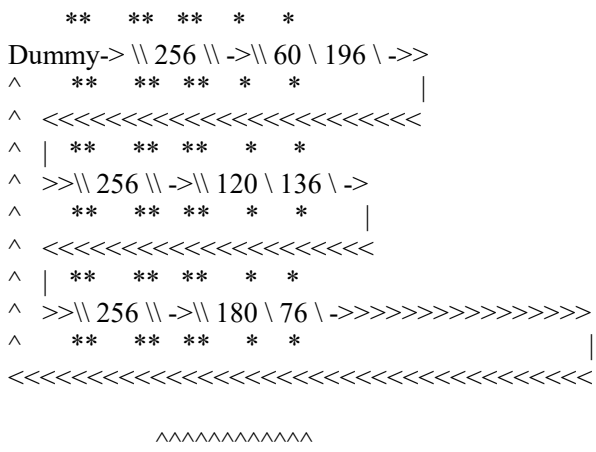
```
   else if (unit_driver == 2) {
printf("\n----- Begin unit driver 2 -----\n");
       /* I created three new unit drivers.

        Here is a unit test driver to test combinations of
       -- request the number of bytes that matches a whole page, and a
          size that is one unit smaller than a page
       -- request more bytes than in one page

        This test makes four allocations from the free list with the goal
        of making the allocation the whole page to test if the allocation is exact
        (one PAGESIZE - 1), and (one PAGESIZE - 2) so that the free list is left empty
        once. And then the third allocation is one PAGESIZE and one PAGESIZE + 1
        to test some bigger size of pages.
        */
       int unit_size = SIZEOF_CHUNK_T;
       int units_in_first_page = PAGESIZE/unit_size;
       assert(units_in_first_page * unit_size == PAGESIZE);
       printf("There are %d units per page, and the size of chunk_t is %d bytes\n",
              units_in_first_page, unit_size);

       int *p1, *p2, *p3, *p4;
       int num_bytes_1, num_bytes_2, num_bytes_3;
       int num_bytes_4;

       // allocate a 1st page - 1
       num_bytes_1 = (units_in_first_page - 1)*unit_size;
       p1 = (int *) Mem_alloc(num_bytes_1);
       printf("first: %d bytes (%d units) p=%p \n",
              num_bytes_1, num_bytes_1/unit_size, p1);
       Mem_print();

       // allocate for 2nd pointer to a page - 2
       num_bytes_2 = (units_in_first_page - 2)*unit_size;
       p2 = (int *) Mem_alloc(num_bytes_2);
       printf("second: %d bytes (%d units) p=%p \n",
              num_bytes_2, num_bytes_2/unit_size, p2);
       Mem_print();

       // allocate 3rd pointer to a page
       num_bytes_3 = (units_in_first_page)*unit_size;

       p3 = (int *) Mem_alloc(num_bytes_3);
       printf("third: %d bytes (%d units) p=%p \n",
```

```
            num_bytes_3, num_bytes_3/unit_size, p3);
        Mem_print();
        printf("unit driver 1: above Mem_print shows empty free list\n");

        // allocate for 4th pointer to a page + 1
        num_bytes_4 = (units_in_first_page + 1)*unit_size;
        p4 = (int *) Mem_alloc(num_bytes_4);
        printf("fourth: %d bytes (%d units) p=%p \n",
            num_bytes_4, num_bytes_4/unit_size, p4);
        Mem_print();

        // next put the memory back into the free list:

        printf("first free of one page - 1 p=%p \n", p1);
        Mem_free(p1);
        Mem_print();

        printf("second free of one page - 2 p=%p \n", p3);
        Mem_free(p2);
        Mem_print();

        printf("third free of one page p=%p \n", p2);
        Mem_free(p3);
        Mem_print();
        printf("fourth free of one page + 1 p=%p\n", p4);
        Mem_free(p4);
        printf("unit driver 2 has returned all memory to free list\n");
        Mem_print();
        Mem_stats();
        printf("\n----- End unit test driver 2 -----\n");
    }

    else if (unit_driver == 3) {
        printf("\n----- Begin unit driver 3 -----\n");
        /* I created three new unit drivers.

        Here is a unit test driver to test combinations of requests and frees such that the free list is
empty

        This test makes four allocations from the free list with the goal
        of making the allocation the whole page to test if the allocation is exact
        (one PAGESIZE - 1), and (two PAGESIZE - 1)so that the free list is left empty
        twice. And then the third allocation is some units of 2 * PAGESIZE + 224 - 2 and 31
        to test some bigger size of pages.
        */
        int unit_size = SIZEOF_CHUNK_T;
        int units_in_first_page = PAGESIZE/unit_size;
        assert(units_in_first_page * unit_size == PAGESIZE);
        printf("There are %d units per page, and the size of chunk_t is %d bytes\n",
            units_in_first_page, unit_size);

        int *p1, *p2, *p3, *p4;
        int num_bytes_1, num_bytes_2, num_bytes_3;
        int num_bytes_4;

        // allocate 1st pointer a page - 1
        num_bytes_1 = (units_in_first_page - 1)*unit_size;
```

```
    p1 = (int *) Mem_alloc(num_bytes_1);
    printf("first: %d bytes (%d units) p=%p \n",
          num_bytes_1, num_bytes_1/unit_size, p1);
    Mem_print();

    // allocate for 2nd pointer to two pages - 1
    num_bytes_2 = (units_in_first_page * 2 - 1)*unit_size;
    p2 = (int *) Mem_alloc(num_bytes_2);
    printf("second: %d bytes (%d units) p=%p \n",
          num_bytes_2, num_bytes_2/unit_size, p2);
    Mem_print();

    // allocate for 3rd pointer to two pages + 224 - 1
    num_bytes_3 = (units_in_first_page*2 + 224 - 1)*unit_size;

    p3 = (int *) Mem_alloc(num_bytes_3);
    printf("third: %d bytes (%d units) p=%p \n",
          num_bytes_3, num_bytes_3/unit_size, p3);
    Mem_print();
    printf("unit driver 1: above Mem_print shows empty free list\n");

    // allocate for 4th pointer to 31 in
    num_bytes_4 = (31)*unit_size;
    p4 = (int *) Mem_alloc(num_bytes_4);
    printf("fourth: %d bytes (%d units) p=%p \n",
          num_bytes_4, num_bytes_4/unit_size, p4);
    Mem_print();

    // next put the memory back into the free list:

    printf("first free of one page - 1 p=%p \n", p1);
    Mem_free(p1);
    Mem_print();

    printf("second free of two pages - 1 p=%p \n", p3);
    Mem_free(p2);
    Mem_print();

    printf("third free of one page + 31 - 1 p=%p \n", p2);
    Mem_free(p3);
    Mem_print();

    printf("fourth free of two pages + 224 - 2 p=%p\n", p4);
    Mem_free(p4);
    printf("unit driver 3 has returned all memory to free list\n");
    Mem_print();
    Mem_stats();
    printf("\n----- End unit test driver 3 -----\n");
}

else if (unit_driver == 4) {
    printf("\n----- Begin unit driver 4 -----\n");
    /* I created three new unit drivers.

     Here is a unit test driver to test the difference between the best fit ,first fit
     and worst fit.
```

This test makes six allocations from the free list with the goal
of making the allocation to have the significant difference between three
SearchPolicy.

first allocate: (PAGESIZE/SIZEOF_CHUNK_T + 60) - 1 unit size
second allocate: (PAGESIZE/SIZEOF_CHUNK_T + 120) - 1 unit size
third allocate: (PAGESIZE/SIZEOF_CHUNK_T + 180) - 1 unit size
The memory chunk size after FIRST THREE allocation will be:



consider the right chunks

Fourth allocate: (76 - 1) unit size
Fifth allocate: (60 - 1) unit size
Sixth allocate: (76 - 1) unit size

The memory chunk size after LAST THREE allocation will be:

(1).best fit: 1* 196 136 (76 - 76 = nil)
2* 196 76 (136 -60 = 76)
3* 196 (76 - 76 = nil)Rover_first or head_first are the same

(2).first fit:1* 120 136 76 (196 - (76))
2* 60 136 76 (120 - (60))
3* 60 60 76 (136 - (76)) This is head_first. Rover_first will be:

1* 196 136 (76 - 76 = nil) (Rover is on the last new node!)
2* 136 136 (196 - 60) (Rover is moved to next available!)
3* 60 136 (136 - 76) (Rover stays 'cause the node is ready!)

(3).worst fit:1* 120 136 76 (196 - (76))
2* 120 76 76 (136 - (60))
3* 44 76 76 (120 - (76)) Rover_first or head_first are the same

I wanna show that the first fit and the worst fit are very inefficient, they generate
very similar consequences when in head first, and the best fit is the most efficient.
But the first_fit Policy will definitely different from where the Rover starts.
*/

```
int unit_size = SIZEOF_CHUNK_T;
int units_in_first_page = PAGESIZE/unit_size;
assert(units_in_first_page * unit_size == PAGESIZE);
printf("There are %d units per page, and the size of chunk_t is %d bytes\n",
       units_in_first_page, unit_size);
```

```
int *p1, *p2, *p3, *p4, *p5, *p6;
int num_bytes_1, num_bytes_2, num_bytes_3;
int num_bytes_4, num_bytes_5, num_bytes_6;

// allocate 1st pointer to a page + 60 - 1 units
num_bytes_1 = (units_in_first_page + 60 - 1)*unit_size;
p1 = (int *) Mem_alloc(num_bytes_1);
printf("first: %d bytes (%d units) p=%p \n",
      num_bytes_1, num_bytes_1/unit_size, p1);
Mem_print();

// allocate for 2nd pointer to a page + 120 - 1 units
num_bytes_2 = (units_in_first_page + 120 - 1)*unit_size;
p2 = (int *) Mem_alloc(num_bytes_2);
printf("second: %d bytes (%d units) p=%p \n",
      num_bytes_2, num_bytes_2/unit_size, p2);
Mem_print();

// allocate for 3rd pointer to a page + 180 - 1 units
num_bytes_3 = (units_in_first_page + 180 - 1)*unit_size;

p3 = (int *) Mem_alloc(num_bytes_3);
printf("third: %d bytes (%d units) p=%p \n",
      num_bytes_3, num_bytes_3/unit_size, p3);
Mem_print();
printf("unit driver 4: above are FIRST THREE allocation\n");
// allocate 4th pointer to 76 - 1 units
num_bytes_4 = (76 - 1)*unit_size;
p4 = (int *) Mem_alloc(num_bytes_4);
printf("fourth: %d bytes (%d units) p=%p \n",
      num_bytes_4, num_bytes_4/unit_size, p4);
Mem_print();

// allocate 5th pointer to 60 - 1 units
num_bytes_5 = (60 - 1)*unit_size;
p5 = (int *) Mem_alloc(num_bytes_5);
printf("fifth: %d bytes (%d units) p=%p \n",
      num_bytes_5, num_bytes_5/unit_size, p5);
Mem_print();

// allocate remaining memory in free list
num_bytes_6 = (76 - 1)*unit_size;

p6 = (int *) Mem_alloc(num_bytes_6);
printf("sixth: %d bytes (%d units) p=%p \n",
      num_bytes_6, num_bytes_6/unit_size, p6);
Mem_print();
printf("unit driver 4: above are LAST THREE allocation\n");

// next put the memory back into the free list:

printf("first free 256 + 60 - 1 p=%p \n", p1);
Mem_free(p1);
Mem_print();

printf("second free 256 + 120 - 1 p=%p \n", p2);
```

```
        Mem_free(p2);
        Mem_print();

        printf("third free 256 + 180 - 1 p=%p \n", p3);
        Mem_free(p3);
        Mem_print();

        printf("fourth free 76 - 1 p=%p\n", p4);
        Mem_free(p4);
        Mem_print();

        printf("fifth free of 60 - 1 p=%p \n", p3);
        Mem_free(p5);
        Mem_print();

        printf("sixth free of 76 - 1 p=%p\n", p4);
        Mem_free(p6);

        printf("unit driver 4 has returned all memory to free list\n");

        Mem_print();
        Mem_stats();
        printf("\n----- End unit test driver 4 -----\n");
    }
    exit(0);
}
```

They are shown in the lab4.c file.