Project 7

秦啸涵　**521021910604**

实现连续内存分配器

由于内存被划分为等大小的页，总内存为 `1MB`，一页为 `4kB`，因此共有256页。我初始采用大小为256的数组来进行内存的模拟(后续更改为双链表)， `main()` 函数主体如下

```
1  int main(int argc, char *argv[])
2  {
3      int shouldrun = 1;
4      char* input;
5      input = (char*) malloc(MAX_LINE * sizeof(char));
6      while(shouldrun)
7      {
8          printf("\033[1;35mallocator>\033[0m");
9          fflush(stdout);
10         fgets(input, 100, stdin);
11         input[strlen(input) - 1] = '\0';
12         if (strcmp(input, "exit") == 0) {
13             shouldrun = 0;
14             continue;
15         }
16         char* args[10];
17         for (int i=0; i<10; i++)
18         {
19             args[i] = (char*) malloc(10 * sizeof(char));
20         }
21         int arg_num = parse(input, args);                    //传
   递参数
22         if (arg_num == 1 && strcmp(args[0], "STAT") == 0)
23         {
24             printstatics();                                  //支
   持STAT指令
25         }
26         if (arg_num == 4 && strcmp(args[0], "RQ") == 0)
```

```
27          {
28              if (strcmp(args[3], "F") == 0)
29                  first_fit(args[1], atoi(args[2]));
30              if (strcmp(args[3], "B") == 0)
31                  best_fit(args[1], atoi(args[2]));
32              if (strcmp(args[3], "W") == 0)
33                  worst_fit(args[1], atoi(args[2]));          //三
    种内存分配方式
34          }
35          if (arg_num == 2 && strcmp(args[0], "RL") == 0)
36          {
37              RL(args[1]);                                    //支
    持内存的释放
38          }
39      }
40      return 0;
41  }
```

实现结果如下：

```
jianke@ubuntu:~/Desktop/final-src-osc10e/lab7$ ./allocator
allocator>RQ P0 1234 W
Allocated P0 from page 0 to page 0
allocator>RQ P1 214324 F
Allocated P1 from page 1 to page 53
allocator>RQ P2 32432 B
Allocated P2 from page 54 to page 61
allocator>RL P1
Process P1, 53 pages, from page 1 to page 53 is removed from memory.
allocator>STAT
Addresses [0:4095] Process P0
Addresses [4096:221183] Unused
Addresses [221184:253951] Process P2
Addresses [253952:1048575] Unused
There are 2 process(es) in total
allocator>RQ P3 1234 W
Allocated P3 from page 62 to page 62
allocator>RQ PR 1234 F
Allocated PR from page 1 to page 1
allocator>RQ P5 1234 B
Allocated P5 from page 2 to page 2
allocator>STAT
Addresses [0:8191] Process P0
Addresses [8192:12287] Process P5
Addresses [12288:221183] Unused
Addresses [221184:253951] Process P2
Addresses [253952:258047] Process P3
Addresses [258048:1048575] Unused
There are 4 process(es) in total
allocator>exit
```

实现三种分配策略

first-fit()

```
void first_fit(char* name, int size)
{
    int i = 0;
    char* tmp = (char*) malloc(10 * sizeof(char));
    strcpy(tmp, name+1);
    int pid = atoi(tmp);
    int page = size/4096 + (size%4096 == 0 ? 0 : 1);
    while (i < 256)
    {
        if (memory[i] == 0)
        {
            int j = i;
            while (j < 256 && memory[j] == 0)
            {
                j++;
            }
            if (j - i ⩾ page)
            {
                for (int k=i; k<i+page; k++)
                {
                    memory[k] = pid+1;
                }
                printf("Allocated %s from page %d to page
%d\n", name, i, i+page-1);
                return;
            }
            else
            {
                i = j;
            }
        }
        else
        {
            i++;
        }
```

```
    }
    printf("Cannot allocate %s, %d\n", name, size);
}
```

best-fit()

```
void best_fit(char* name, int size)
{
    int i = 0;
    char* tmp = (char*) malloc(10 * sizeof(char));
    strcpy(tmp, name+1);
    int pid = atoi(tmp);
    int page = size/4096 + (size%4096 == 0 ? 0 : 1);
    int min = 256;
    int min_index = -1;
    while (i < 256)
    {
        if (memory[i] == 0)
        {
            int j = i;
            while (j < 256 && memory[j] == 0)
            {
                j++;
            }
            if (j - i >= page && j - i < min)
            {
                min = j - i;
                min_index = i;
            }
            i = j;
        }
        else
        {
            i++;
        }
    }
    if (min_index == -1)
    {
        printf("Cannot allocate %s, %d\n", name, size);
```

```
34          return;
35      }
36      for (int k=min_index; k<min_index+page; k++)
37      {
38          memory[k] = pid+1;
39      }
40      printf("Allocated %s from page %d to page %d\n", name,
    min_index, min_index+page-1);
41  }
```

worst-fit()

```
1   void worst_fit(char* name, int size)
2   {
3       int i = 0;
4       char* tmp = (char*) malloc(10 * sizeof(char));
5       strcpy(tmp, name+1);
6       int pid = atoi(tmp);
7       int page = size/4096 + (size%4096 == 0 ? 0 : 1);
8       int max = 0;
9       int max_index = -1;
10      while (i < 256)
11      {
12          if (memory[i] == 0)
13          {
14              int j = i;
15              while (j < 256 && memory[j] == 0)
16              {
17                  j++;
18              }
19              if (j - i >= page && j - i > max)
20              {
21                  max = j - i;
22                  max_index = i;
23              }
24              i = j;
25          }
26          else
27          {
```

```
            i++;
        }
    }
    if (max_index == -1)
    {
        printf("Cannot allocate %s, %d\n", name, size);
        return;
    }
    for (int k=max_index; k<max_index+page; k++)
    {
        memory[k] = pid+1;
    }
    printf("Allocated %s from page %d to page %d\n", name, max_index, max_index+page-1);
}
```

实现结果上图已展示

bonus:使用有序双链表加速best-fit()和worst-fit()

更改内存结构为双链表，并更改对应的插入、删除逻辑

```
typedef struct Block{
    int used;
    int start;
    int end;
    char name[5];
}block;

typedef struct Node{
   block *hole;
   struct Node* next;
   struct Node* pre;
}node;
```

更改后的`bestFit()`，只需要按链表顺序查找至首个满足要求的`hole`即可

```
int bestFit(int process_size, char* name){
    int min_fit_hole_size = 1e9;
    node*tmp = head -> next;
    while(tmp != tail){
       if(tmp -> hole -> used){
           tmp = tmp -> next;
           continue;
       }
       int hole_size = tmp -> hole -> end - tmp -> hole -> start + 1;
       if(hole_size < process_size){
           tmp = tmp -> next;
           continue;
       }
       else{
           min_fit_hole_size = (hole_size < min_fit_hole_size) ? hole_size : min_fit_hole_size;
           tmp = tmp -> next;
       }
    }

    tmp = head -> next;
    if(min_fit_hole_size == -1) return -1;
    while(1){
         if(tmp -> hole -> used){
           tmp = tmp -> next;
         }else{
           int hole_size = tmp -> hole -> end - tmp -> hole -> start + 1;
           if(hole_size == min_fit_hole_size){
               insert_process(tmp, process_size, name);
               break;
           }
           tmp = tmp -> next;
       }
    }
    return 0;
}
```

更改后的`worstFit()`，直接顺着链表找最大的`hole`，判断是否满足要求即可。(注意这里面涉及到一些特判，避免出现段错误)

```
int worstFit(int process_size, char* name){
    // find the largest hole;
    int max_hole_size = -1;
    node*tmp = head -> next;
    while(tmp != tail){
       if(tmp -> hole -> used){
           tmp = tmp -> next;
       }else{
           int hole_size = tmp -> hole -> end - tmp -> hole -> start;
           max_hole_size = (hole_size > max_hole_size) ? hole_size : max_hole_size;
           tmp = tmp -> next;
       }
    }
    tmp = head -> next;
    //printf("%d", process_size);
    if(max_hole_size + 1 < process_size) return -1;
    while(1){
         if(tmp -> hole -> used){
           tmp = tmp -> next;
         }else{
           int hole_size = tmp -> hole -> end - tmp -> hole -> start;
           if(hole_size == max_hole_size){
              // printf("insert");
               insert_process(tmp, process_size, name);
               break;
           }
           tmp = tmp -> next;
       }
    }
    return 0;

}
```

实现结果：

```
jianke@ubuntu:~/Desktop/final-src-osc10e/lab7$ ./allocator-list
Mem[0]-[1048575], unused
allocator>RQ P0 1234 W
allocator>RQ P1 213243 F
allocator>RQ P2 12344 B
allocator>RL P1
allocator>STAT
Mem[0]-[1233], [P0]
Mem[1234]-[214476], unused
Mem[214477]-[226820], [P2]
Mem[226821]-[1048575], unused
allocator>RQ P3 34324 W
allocator>RQ P4 12324 F
allocator>STAT
Mem[0]-[1233], [P0]
Mem[1234]-[13557], [P4]
Mem[13558]-[214476], unused
Mem[214477]-[226820], [P2]
Mem[226821]-[261144], [P3]
Mem[261145]-[1048575], unused
allocator>exit
Bye!
```