操作系统 project-7 实验报告

* 姓名: 管仁阳　学号:519021911058　邮箱: guanrenyang@sjtu.edu.cn

1 实验名称

Contiguous Memory Allocation

2 实验任务

编写 C 程序来实现连续内存分配，需要支持以下方式：

1. First-Fit
2. Worst-Fit
3. Best-Fit

且支持以下功能

1. 用户请求分配连续的内存
2. 用户释放连续的内存块
3. 将为使用的内存块压缩成一整个
4. 显示使用的和未使用的的内存块

3 预备知识

3.1 连续内存分配：

主内存必须同时兼顾操作系统和各种用户进程。因此我们需要使用连续内存分配。内存通常分为两个分区：一个用于操作系统和一个用于用户进程的系统。每个进程都包含在单个内存单元中且与包含下一个进程的部分相邻。

3.2 First-Fit

分配足够大的第一个 hole。搜索可以在第一个 hole 或者它以前开始。一旦发现空的 hole 就可以立即停止搜索因为它已经足够大了。

3.3 Worst-Fit

分配最大的一个 hole。搜索必须遍历所有 hole 以找到最大的 hole。若未找到则拒绝请求。

3.4 Best-Fit

分配大小最合适的一个 hole。搜索必须遍历所有 hole 以找到最合适的 hole。若未找到则拒绝请求。

4 实验内容

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define MAX_INPUT 20

#define RQ 1
#define RL 2
#define C 3
#define STAT 4

#define Fail -1

short *memory;//0 empty, 1 full

int *memory_size;
struct processor
{
    char * name;
    int starting_address;
    int ending_address;
    struct processor * next;
};

struct processor *head;
// list function
void insert(char* name, int starting_address, int ending_address);
struct processor * search_by_address(int i);
struct processor * search_by_name(char * name);
struct processor * delete_by_name(char * name);

int Decode_Input(char * input);

int request(char * input);
int release(char * input);
void compact();
void status();

int first_fit(int size);
int worst_fit(int size);
int best_fit(int size);

int main(int argc,char * argv[])
{
    memory_size=(int*)malloc(sizeof(int));
    (*memory_size)=atoi(argv[1]);
    memory=(short*) malloc(sizeof(short)*(*memory_size));

```

```
49      fprintf(stdout,"allocator>");
50
51      char * input=malloc(MAX_INPUT*sizeof(char));
52      while (fgets(input,MAX_INPUT-1,stdin))
53      {
54
55          int command=Decode_Input(input);
56          switch (command)
57          {
58          case RQ:
59              request(input);
60              break;
61          case RL:
62              release(input);
63              break;
64          case C:
65              compact();
66              break;
67          case STAT:
68              status();
69          default:
70              break;
71          }
72          fprintf(stdout,"allocator>");
73      }
74
75  }
76  int Decode_Input(char * input)
77  {
78      if(input[0]=='C')
79          return C;
80      else if(input[0]=='S')
81          return STAT;
82      else if(input[1]=='Q')
83          return RQ;
84      else if(input[1]=='L')
85          return RL;
86  }
87  int request(char *input)
88  {
89      char * command=strsep(&input," ");
90      char * name=strsep(&input," ");
91      int size=atoi(strsep(&input," "));
92      char * strategy=strsep(&input," ");
93      //ok
94
95      int start;
96      if(strategy[0]=='F')
97          start=first_fit(size);
98      else if(strategy[0]=='W')
```

```
99          start=worst_fit(size);
100     else if(strategy[0]=='B')
101         start=best_fit(size);
102
103     for(int i=0;i<size;++i)
104     {
105         memory[start+i]=1;
106     }
107     insert(name,start,start+size-1);
108
109 }
110 int release(char * input)
111 {
112     char * command=strsep(&input," ");
113     char * name=strsep(&input,"\n");
114
115     struct processor * temp=delete_by_name(name);
116     for(int i=temp->starting_address;i<=temp->ending_address;++i)
117         memory[i]=0;
118     free(temp);
119 }
120
121 void compact()
122 {
123     struct processor *new_head=NULL;
124     for(int i=0;i<(*memory_size);++i)
125     {
126         while (head!=NULL)
127         {
128             int size=head->ending_address-head->starting_address+1;
129
130             for(int j=0;j<size;j++)
131                 memory[i+j]=1;
132
133             head->starting_address=i;
134             head->ending_address=i+size-1;
135             struct processor * temp=head;
136             head=head->next;
137
138             // all temp to the new head
139             if(new_head==NULL)
140             {
141                 new_head=temp;
142                 new_head->next=NULL;
143             }
144             else
145             {
146                 temp->next=new_head;
147                 new_head=temp;
148             }
```

```
149                 i=i+size;
150             }
151             memory[i]=0;
152         }
153         head=new_head;
154 }
155 void status()
156 {
157
158         int start=0;
159         int end=start;
160         int prev=0;
161         for(int i=0;i<(*memory_size);++i)
162         {
163             if(prev==0&&memory[i]==1&&i!=0)
164             {
165                 fprintf(stdout,"Addresses [%d:%d] Unuesd\n",start,end);
166             }
167             else if(prev==1&&memory[i]==0)
168             {
169                 start=end=i;
170                 prev=0;
171                 if(i==(*memory_size)-1)
172                     fprintf(stdout,"Addresses [%d:%d] Unuesd\n",start,end);
173             }
174             else if(prev==0&&memory[i]==0)
175             {
176                 end=i;
177                 if(i==(*memory_size)-1)
178                     fprintf(stdout,"Addresses [%d:%d] Unuesd\n",start,end);
179             }
180             if(memory[i]==1)
181             {
182                 struct processor* proc=search_by_address(i);
183                 fprintf(stdout,"Addresses [%d:%d] Process %s\n",proc->starting_ad
184                 i=proc->ending_address;
185                 prev=1;
186             }
187         }
188
189 }
190
191 int first_fit(int size)
192 {
193         int start=0;
194         int end=start;
195         int prev=1;
196         int sum=0;
197
198         for(int i=0;i<(*memory_size);i++)
```

```
199     {
200         if(prev==1&&memory[i]==0)
201         {
202             start=i;
203             end=start;
204             prev=0;
205             sum++;
206         }
207         else if(prev==0&&memory[i]==0)
208         {
209             sum++;
210             end=i;
211         }
212         else if(prev==0&&memory[i]==1)
213         {
214             prev=1;
215             sum=0;
216         }
217 
218         if(sum>=size)
219         {
220             return start;
221         }
222     }
223     return -1;
224 }
225 int worst_fit(int size)
226 {
227     int start=0;
228     int end=start;
229     int prev=1;
230     int sum=0;
231 
232     int max=-1;
233     int max_start=-1;
234     for(int i=0;i<(*memory_size);i++)
235     {
236         if(prev==1&&memory[i]==0)//start of contious empty memory
237         {
238             start=i;
239             prev=0;
240             sum++;
241         }
242         else if(prev==0&&memory[i]==0&&i!=((*memory_size)-1))
243         {
244             sum++;
245         }
246         else if(prev==0&&memory[i]==1)//end of contious empty memory
247         {
248             prev=1;
```

```
249                 sum=0;
250             }
251
252             if((i==((*memory_size)-1)&&memory[i]==0)||(memory[i]==0&&memory[i+1]=
253             {
254                 if(sum>max)
255                 {
256                     max=sum;
257                     max_start=start;
258                 }
259             }
260
261         }
262         if(max_start!=-1)
263             return max_start;
264         else
265             return -1;
266 }
267 int best_fit(int size)
268 {
269         int start=0;
270         int end=start;
271         int prev=1;
272         int sum=0;
273
274         int min=(*memory_size)+1;
275         int min_start=-1;
276         for(int i=0;i<(*memory_size);i++)
277         {
278             if(prev==1&&memory[i]==0)//start of contious empty memory
279             {
280                 start=i;
281                 prev=0;
282                 sum++;
283             }
284             else if(prev==0&&memory[i]==0&&i!=((*memory_size)-1))
285             {
286                 sum++;
287             }
288             else if(prev==0&&memory[i]==1)//end of contious empty memory
289             {
290                 prev=1;
291                 sum=0;
292             }
293
294             if((i==((*memory_size)-1)&&memory[i]==0)||(memory[i]==0&&memory[i+1]=
295             {
296                 if(sum<min&&sum>=size)
297                 {
298                     min=sum;
```

```
                min_start=start;
            }
        }
    }
    if(min_start!=-1)
        return min_start;
    else
        return -1;
}

void insert(char* name, int starting_address, int ending_address)
{
    if(head==NULL)
    {
        head=(struct processor*) malloc(sizeof(struct processor));
        head->name=strdup(name);
        head->starting_address=starting_address;
        head->ending_address=ending_address;
        head->next=NULL;
    }
    else
    {
        struct processor * temp=(struct processor*) malloc(sizeof(struct proc
        temp->name=strdup(name);
        temp->starting_address=starting_address;
        temp->ending_address=ending_address;

        temp->next=head->next;
        head->next=temp;
    }
}
struct processor * search_by_address(int i)
{
    if(head==NULL)
        return NULL;

    struct processor *temp=head;
    struct processor *prev;
    while (temp!=NULL)
    {
        if(temp->starting_address==i)
            return temp;
        prev=temp;
        temp=temp->next;
    }
    return NULL;
}
struct processor * search_by_name(char * name)
{
    if(head==NULL)
```

```
349         return NULL;
350
351     struct processor *temp=head;
352     struct processor *prev;
353     while (temp!=NULL)
354     {
355         if(strcmp(name,temp->name)==0)
356             return temp;
357         prev=temp;
358         temp=temp->next;
359     }
360     return NULL;
361 }
362 struct processor * delete_by_name(char * name)
363 {
364     if(head==NULL)
365         return NULL;
366
367     struct processor *temp=head;
368     struct processor *prev;
369     while (temp!=NULL)
370     {
371         if(strcmp(name,temp->name)==0)
372         {
373             if(temp==head)
374                 head=temp->next;
375             else
376                 prev->next=temp->next;
377             return temp;
378         }
379
380         prev=temp;
381         temp=temp->next;
382     }
383     return NULL;
384 }
```

5 实验结果

实验结果如图 1。可见 RQ 中 F、W、B 三个功能，RL、STAT 与 C 都很好地完成。

6 总结与思考

通过此次实验我实现了 First-Fit、Best-Fit 与 Worst-Fit 三种内存分配算法。且实现了内存碎片压缩。在此过程中我对硬件的理解与调度算法理解地更加深入了，并且我更好地理解了连续内存分配的效率高的原因。
同时我也知道了这种直接内存分配的限制所在，促使我更好地理解了为什么要使用虚拟内存与物理内存分离的做法。

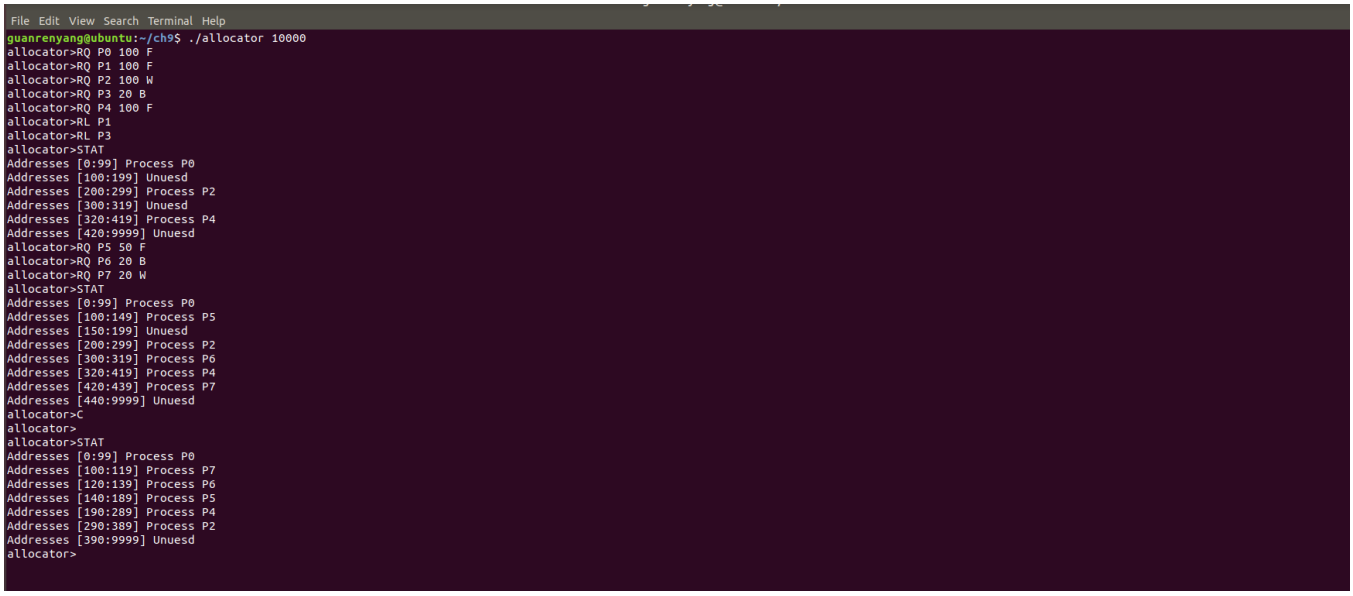
File Edit View Search Terminal Help
guanrenyang@ubuntu:~/ch9$ ./allocator 10000
allocator>RQ P0 100 F
allocator>RQ P1 100 F
allocator>RQ P2 100 W
allocator>RQ P3 20 B
allocator>RQ P4 100 F
allocator>RL P1
allocator>RL P3
allocator>STAT
Addresses [0:99] Process P0
Addresses [100:199] Unuesd
Addresses [200:299] Process P2
Addresses [300:319] Unuesd
Addresses [320:419] Process P4
Addresses [420:9999] Unuesd
allocator>RQ P5 50 F
allocator>RQ P6 20 B
allocator>RQ P7 20 W
allocator>STAT
Addresses [0:99] Process P0
Addresses [100:149] Process P5
Addresses [150:199] Unuesd
Addresses [200:299] Process P2
Addresses [300:319] Process P6
Addresses [320:419] Process P4
Addresses [420:439] Process P7
Addresses [440:9999] Unuesd
allocator>C
allocator>
allocator>STAT
Addresses [0:99] Process P0
Addresses [100:119] Process P7
Addresses [120:139] Process P6
Addresses [140:189] Process P5
Addresses [190:289] Process P4
Addresses [290:389] Process P2
Addresses [390:9999] Unuesd
allocator>

图 1: Result