操作系统实验3

题目：动态内存分配器的实现

徐宇鸣，PB17111636

1、实验目的

使用显示空闲链表实现一个32位系统堆内存分配器。

2、实验主要步骤

1、思考设计实验中空闲块规格，各种函数实现的具体操作

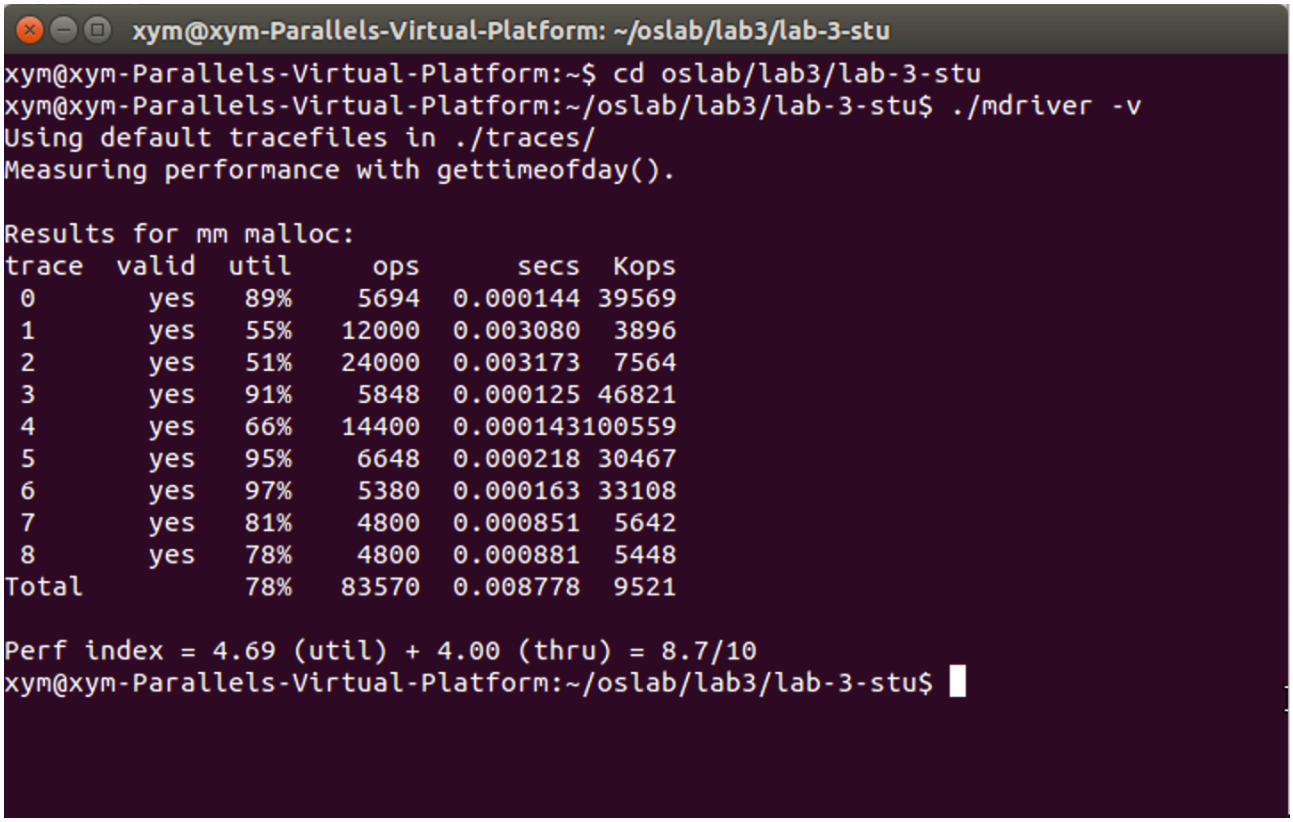
2、针对自己所设计的动态内存分配器进行代码实现

3、调试实现

3、实验运行结果截图

1: version1:

首次适配+LIFO机制



```
xym@xym-Parallels-Virtual-Platform: ~/oslab/lab3/lab-3-stu
xym@xym-Parallels-Virtual-Platform:~$ cd oslab/lab3/lab-3-stu
xym@xym-Parallels-Virtual-Platform:~/oslab/lab3/lab-3-stu$ ./mdriver -v
Using default tracefiles in ./traces/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   89%    5694  0.000144 39569
 1       yes   55%   12000  0.003080  3896
 2       yes   51%   24000  0.003173  7564
 3       yes   91%    5848  0.000125 46821
 4       yes   66%   14400  0.000143100559
 5       yes   95%    6648  0.000218 30467
 6       yes   97%    5380  0.000163 33108
 7       yes   81%    4800  0.000851  5642
 8       yes   78%    4800  0.000881  5448
Total          78%   83570  0.008778  9521

Perf index = 4.69 (util) + 4.00 (thru) = 8.7/10
xym@xym-Parallels-Virtual-Platform:~/oslab/lab3/lab-3-stu$
```

存在问题：存在几个样例空间分配不好，需要进行改动

原因：使用首次适配和LIFO使得外部碎片增多

改进思路：使用其他适配进行尝试

光从分数来看，助教给的测试评分很容易达到速度的4分满分，关键是空间利用率

2: version1.1:

将version1改为最佳适配，速度仍保持在4分（尽管用时已经变成了原来的15倍），但是空间利用率分数上升了0.3

也说明了如果只考虑分数来看本次实验分数中速度分数并不是瓶颈

但是需要注意到存在速度超过0.15s使得分数不为满分

```
xym@xym-Parallels-Virtual-Platform: ~/oslab/lab3/lab-3-stu
]
   if(GET(heap_listp)!=NULL)
                    ^
gcc -g -Wall -O2 -m32 -o mdriver mdriver.o mm.o memlib.o fsecs.o fcyc.o clock.o
ftimer.o
xym@xym-Parallels-Virtual-Platform:~/oslab/lab3/lab-3-stu$ ./mdriver -v
Using default tracefiles in ./traces/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.000856  6650
 1       yes   55%   12000  0.022061   544
 2       yes   51%   24000  0.084130   285
 3       yes   98%    5848  0.001398  4182
 4       yes   66%   14400  0.000144100348
 5       yes  100%    6648  0.001772  3751
 6       yes  100%    5380  0.001770  3039
 7       yes   92%    4800  0.006819   704
 8       yes   89%    4800  0.007895   608
Total          83%   83570  0.126845   659

Perf index = 5.00 (util) + 4.00 (thru) = 9.0/10
xym@xym-Parallels-Virtual-Platform:~/oslab/lab3/lab-3-stu$
```

3:version2

将LIFO机制改为按地址顺序维护机制，空间利用率相比version1上升了0.25，但是时间居然变成了原来的将近70倍

原因：访存开销太大，由于使用了大量的宏定义中存在过多的不必要的访存操作，比如对一个相同的数据进行了两次访存而不是储存值在临时变量中加以使用。（但是在经过改写之后发现并没有使得速度上升，说明在gcc进行-o2编译的时候已经将大部分都优化过了

```
xym@xym-Parallels-Virtual-Platform: ~/oslab/lab3/lab-3-stu
]
   if(GET(SUCC(bp))!=NULL)
                    ^
gcc -g -Wall -O2 -m32 -o mdriver mdriver.o mm.o memlib.o fsecs.o fcyc.o clock.o
ftimer.o
xym@xym-Parallels-Virtual-Platform:~/oslab/lab3/lab-3-stu$ ./mdriver -v
Using default tracefiles in ./traces/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.002052  2775
 1       yes   55%   12000  0.171760    70
 2       yes   51%   24000  0.346561    69
 3       yes   98%    5848  0.000849  6886
 4       yes   66%   14400  0.000142101695
 5       yes  100%    6648  0.001073  6194
 6       yes  100%    5380  0.000806  6672
 7       yes   88%    4800  0.003547  1353
 8       yes   87%    4800  0.003546  1354
Total          83%   83570  0.530337   158

Perf index = 4.96 (util) + 1.05 (thru) = 6.0/10
xym@xym-Parallels-Virtual-Platform:~/oslab/lab3/lab-3-stu$
```

4:version2.1

在version2的基础之上将首次适配改为最佳适配，结果空间利用率不升反降，且速度又下降了一些。

```
xym@xym-Parallels-Virtual-Platform: ~/oslab/lab3/lab-3-stu
]
   if(GET(SUCC(bp))!=NULL)
                     ^
gcc -g -Wall -O2 -m32 -o mdriver mdriver.o mm.o memlib.o fsecs.o fcyc.o clock.o
ftimer.o
xym@xym-Parallels-Virtual-Platform:~/oslab/lab3/lab-3-stu$ ./mdriver -v
Using default tracefiles in ./traces/
Measuring performance with gettimeofday().

Results for mm malloc:
trace  valid  util     ops      secs  Kops
 0       yes   99%    5694  0.003018  1886
 1       yes   55%   12000  0.201425    60
 2       yes   51%   24000  0.360087    67
 3       yes   98%    5848  0.002906  2012
 4       yes   66%   14400  0.000163 88235
 5       yes  100%    6648  0.003836  1733
 6       yes  100%    5380  0.004011  1341
 7       yes   87%    4800  0.013973   344
 8       yes   84%    4800  0.011855   405
Total          82%   83570  0.601274   139

Perf index = 4.94 (util) + 0.93 (thru) = 5.9/10
xym@xym-Parallels-Virtual-Platform:~/oslab/lab3/lab-3-stu$
```

3.1、实验代码

version1 :

```
char *heap_listp;
static void *coalesce(void *bp)
{
        size_t prev_alloc = GET(HDRP(bp))&0x2;
        size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
        size_t size = GET_SIZE(HDRP(bp));
        if(prev_alloc && next_alloc)//插入空闲链表头部
        {
                PUT(PRED(bp), heap_listp);//前驱为heap_listp
                PUT(SUCC(bp), GET(heap_listp));
                if(GET(heap_listp)!=NULL)
                {
                        PUT(PRED(GET(heap_listp)), bp);
                }
                PUT(heap_listp, (unsigned int)bp);
        }
        else if(prev_alloc && !next_alloc)
        {
                size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
                PUT(PRED(bp), GET(PRED(NEXT_BLKP(bp))));
```

```
        PUT(SUCC(bp), GET(SUCC(NEXT_BLKP(bp))));
        if(GET(PRED(NEXT_BLKP(bp)))!=heap_listp)
        {
                PUT(SUCC(GET(PRED(NEXT_BLKP(bp)))), GET(SUCC(NEXT_BLKP(bp))));
        }
        else
        {
                PUT(heap_listp, GET(SUCC(NEXT_BLKP(bp))));
        }
        if(GET(SUCC(NEXT_BLKP(bp)))!=NULL)
        {
                PUT(PRED(GET(SUCC(NEXT_BLKP(bp)))), GET(PRED(NEXT_BLKP(bp))));
        }
        PUT(HDRP(bp), PACK(size, 2));
        PUT(FTRP(bp), PACK(size, 2));
        PUT(PRED(bp), heap_listp);//前驱为heap_listp
        PUT(SUCC(bp), GET(heap_listp));
        if(GET(heap_listp)!=NULL)
        {
                PUT(PRED(GET(heap_listp)), bp);
        }
        PUT(heap_listp, (unsigned int)bp);
}
else if(!prev_alloc && next_alloc)
{
        size += GET_SIZE(HDRP(PREV_BLKP(bp)));
        if(GET(PRED(PREV_BLKP(bp)))!=heap_listp)
        {
                PUT(SUCC(GET(PRED(PREV_BLKP(bp)))), GET(SUCC(PREV_BLKP(bp))));
        }
        else
        {
                PUT(heap_listp, GET(SUCC(PREV_BLKP(bp))));
        }
        if(GET(SUCC(PREV_BLKP(bp)))!=NULL)
        {
                PUT(PRED(GET(SUCC(PREV_BLKP(bp)))), GET(PRED(PREV_BLKP(bp))));
        }
        PUT(FTRP(bp), PACK(size, GET(HDRP(PREV_BLKP(bp)))&0x7));
        PUT(HDRP(PREV_BLKP(bp)), GET(FTRP(bp)));
        bp = PREV_BLKP(bp);
        PUT(PRED(bp), heap_listp);//前驱为heap_listp
        PUT(SUCC(bp), GET(heap_listp));
        if(GET(heap_listp)!=NULL)
        {
                PUT(PRED(GET(heap_listp)), bp);
        }
        PUT(heap_listp, (unsigned int)bp);
```

```
	}
	else
	{
		size += GET_SIZE(HDRP(PREV_BLKP(bp))) + GET_SIZE(FTRP(NEXT_BLKP(bp)));
		if(GET(PRED(PREV_BLKP(bp)))!=heap_listp)
		{
			PUT(SUCC(GET(PRED(PREV_BLKP(bp)))), GET(SUCC(PREV_BLKP(bp))));
		}
		else
		{
			PUT(heap_listp, GET(SUCC(PREV_BLKP(bp))));
		}
		if(GET(SUCC(PREV_BLKP(bp)))!=NULL)
		{
			PUT(PRED(GET(SUCC(PREV_BLKP(bp)))), GET(PRED(PREV_BLKP(bp))));
		}
		if(GET(PRED(NEXT_BLKP(bp)))!=heap_listp)
		{
			PUT(SUCC(GET(PRED(NEXT_BLKP(bp)))), GET(SUCC(NEXT_BLKP(bp))));
		}
		else
		{
			PUT(heap_listp, GET(SUCC(NEXT_BLKP(bp))));
		}
		if(GET(SUCC(NEXT_BLKP(bp)))!=NULL)
		{
			PUT(PRED(GET(SUCC(NEXT_BLKP(bp)))), GET(PRED(NEXT_BLKP(bp))));
		}
		PUT(HDRP(PREV_BLKP(bp)), PACK(size, GET(HDRP(PREV_BLKP(bp)))&0x7));
		PUT(FTRP(NEXT_BLKP(bp)), GET(HDRP(PREV_BLKP(bp))));
		bp = PREV_BLKP(bp);
		PUT(PRED(bp), heap_listp);//前驱为heap_listp
		PUT(SUCC(bp), GET(heap_listp));
		if(GET(heap_listp)!=NULL)
		{
			PUT(PRED(GET(heap_listp)), bp);
		}
		PUT(heap_listp, (unsigned int)bp);
	}
	return bp;
}
static void *extend_heap(size_t words)//合并部分才考虑插入
{
	char *bp;
	size_t size;
	size = (words %2) ? (words+1) * WSIZE :words * WSIZE;
	if((long)(bp = mem_sbrk(size))== -1)
		return NULL;
```

```
	PUT(HDRP(bp), PACK(size,(GET(HDRP(bp))&0x2)));
	PUT(FTRP(bp), GET(HDRP(bp)));
	PUT(HDRP(NEXT_BLKP(bp)), PACK(0,1));
	if(GET(heap_listp)==bp)
	{
		PUT(bp, bp);
		PUT(bp+WSIZE, bp);
		return bp;
	}
	return coalesce(bp);
}
int mm_init(void)//空闲链表最开始指向NULL
{
	if((heap_listp = mem_sbrk(4*WSIZE)) == (void *)-1)
		return -1;
	PUT(heap_listp, 0);
	PUT(heap_listp + (1*WSIZE), PACK(DSIZE, 1));
	PUT(heap_listp + (2*WSIZE), PACK(DSIZE, 1));
	PUT(heap_listp + (3*WSIZE), PACK(0,3));
	if (extend_heap(CHUNKSIZE/WSIZE) == NULL)
		return -1;
	return 0;
}
/*
 * mm_malloc - Allocate a block by incrementing the brk pointer.
 *     Always allocate a block whose size is a multiple of the alignment.
 */
void *find_fit(size_t size)
{
	char* temp;
	temp = GET(heap_listp);
	while(temp!=NULL)
	{
		if(size<=GET_SIZE(HDRP(temp)))
			return temp;
		temp=GET(SUCC(temp));
	}
	return NULL;
}

void place(void *bp,size_t size)
{
	size_t temp = GET_SIZE(HDRP(bp)) - size;
	if(GET(PRED(bp))!=heap_listp)
	{
		PUT(SUCC(GET(PRED(bp))), GET(SUCC(bp)));
	}
	else
```

```
        {
                PUT(heap_listp, GET(SUCC(bp)));
        }
        if(GET(SUCC(bp))!=NULL)
        {
                PUT(PRED(GET(SUCC(bp))), GET(PRED(bp)));
        }
        if(temp<4*WSIZE)
        {
                PUT(HDRP(bp),GET(HDRP(bp))|0x1);
                PUT(HDRP(NEXT_BLKP(bp)),GET(HDRP(NEXT_BLKP(bp)))|0x2);
        }
        else
        {
                PUT(HDRP(bp),((GET(HDRP(bp))|0x1)&0x7)|size);
                PUT(FTRP(bp),GET(HDRP(bp)));
                bp = NEXT_BLKP(bp);
                PUT(HDRP(bp),PACK(temp, 0x2));
                PUT(FTRP(bp),PACK(temp, 0x2));
                PUT(PRED(bp), heap_listp);//前驱为heap_listp
                PUT(SUCC(bp), GET(heap_listp));
                if(GET(heap_listp)!=NULL)
                {
                        PUT(PRED(GET(heap_listp)), bp);
                }
                PUT(heap_listp, (unsigned int)bp);
        }
}

void *mm_malloc(size_t size)
{
        size_t asize;
        size_t extendsize;
        char *bp;
        if(size == 0)
                return NULL;
        if(size <=DSIZE)
                asize = 2*DSIZE;
        else
                asize = DSIZE *((size + (DSIZE)+(DSIZE - 1))/DSIZE);
        if((bp = find_fit(asize))!= NULL)
        {
                place(bp, asize);
                return bp;
        }
        extendsize = MAX(asize,CHUNKSIZE);
        if((bp= extend_heap(extendsize/WSIZE))==NULL)
                return NULL;
```

```
	place(bp, asize);
	return bp;
}

/*
 * mm_free - Freeing a block does nothing.
 */
void mm_free(void *bp)
{
	size_t size = GET_SIZE(HDRP(bp));
	PUT(HDRP(bp), PACK(size, GET(HDRP(bp))&0x7));
	PUT(FTRP(bp), GET(HDRP(bp)));
	if(GET_SIZE(HDRP(NEXT_BLKP(bp)))==0)
		GET(HDRP(NEXT_BLKP(bp))) -=2;
	coalesce(bp);
}
```

version1.1:

只写出了有别于version1的地方

```
void *find_fit(size_t size)
{
	char* temp;
	temp = GET(heap_listp);
	while(temp!=NULL)
	{
		if(GET_SIZE(HDRP(temp))<=size)
			return temp;
		temp = GET(SUCC(temp));
	}
	return NULL;
}
```

version2:

```
static void *coalesce(void *bp)
{
	size_t prev_alloc = GET(HDRP(bp))&0x2;
	size_t next_alloc = GET_ALLOC(HDRP(NEXT_BLKP(bp)));
	size_t size = GET_SIZE(HDRP(bp));
	char *temp;
	if(prev_alloc && next_alloc)
	{
		if(GET(heap_listp)==NULL)
		{
			PUT(heap_listp, (unsigned int)bp);
			PUT(SUCC(bp), NULL);
			PUT(bp, heap_listp);
		}
		else if(GET(heap_listp)>bp)
		{
			PUT(PRED(bp), heap_listp);
```

```
                PUT(SUCC(bp), GET(heap_listp));
                PUT(PRED(GET(heap_listp)), (unsigned int)bp);
                PUT(heap_listp, (unsigned int)bp);
        }
        else
        {
                temp=GET(heap_listp);
                while(temp<bp)
                {
                        if(GET(SUCC(temp))==NULL)
                                break;
                        temp=GET(SUCC(temp));
                }
                if(temp<bp)
                {
                        PUT(SUCC(temp), (unsigned int)bp);
                        PUT(PRED(bp), (unsigned int)temp);
                        PUT(SUCC(bp), NULL);
                }
                else
                {
                        PUT(SUCC(GET(PRED(temp))), (unsigned int)bp);
                        PUT(PRED(bp), GET(PRED(temp)));
                        PUT(SUCC(bp), (unsigned int)temp);
                        PUT(PRED(temp), (unsigned int)bp);
                }
        }
}
else if(prev_alloc && !next_alloc)
{
        size += GET_SIZE(HDRP(NEXT_BLKP(bp)));
        PUT(PRED(bp), GET(PRED(NEXT_BLKP(bp))));
        PUT(SUCC(bp), GET(SUCC(NEXT_BLKP(bp))));
        if(GET(PRED(bp))!=heap_listp)
        {
                PUT(SUCC(GET(PRED(bp))), (unsigned int)bp);
        }
        else
        {
                PUT(heap_listp, (unsigned int)bp);
        }
        if(GET(SUCC(bp))!=NULL)
        {
                PUT(PRED(GET(SUCC(bp))), (unsigned int)bp);
        }
        PUT(HDRP(bp), PACK(size, 2));
        PUT(FTRP(bp), PACK(size, 2));
}
```

```
	else if(!prev_alloc && next_alloc)
	{
		size += GET_SIZE(HDRP(PREV_BLKP(bp)));
		PUT(FTRP(bp), PACK(size, GET(HDRP(PREV_BLKP(bp)))&0x7));
		PUT(HDRP(PREV_BLKP(bp)), GET(FTRP(bp)));
		bp = PREV_BLKP(bp);
	}
	else
	{
		size += GET_SIZE(HDRP(PREV_BLKP(bp))) + GET_SIZE(FTRP(NEXT_BLKP(bp)));
		PUT(SUCC(PREV_BLKP(bp)), GET(SUCC(NEXT_BLKP(bp))));
		if(GET(SUCC(NEXT_BLKP(bp)))!=NULL)
			PUT(PRED(GET(SUCC(NEXT_BLKP(bp)))), PREV_BLKP(bp));
		PUT(HDRP(PREV_BLKP(bp)), PACK(size, GET(HDRP(PREV_BLKP(bp)))&0x7));
		PUT(FTRP(NEXT_BLKP(bp)), GET(HDRP(PREV_BLKP(bp))));
		bp = PREV_BLKP(bp);
	}
	return bp;
}
void *find_fit(size_t size)
{
	char* temp;
	temp = GET(heap_listp);
	while(temp!=NULL)
	{
		if(GET_SIZE(HDRP(temp))<=size)
			return temp;
		temp = GET(SUCC(temp));
	}
	return NULL;
}

void place(void *bp,size_t size)
{
	size_t temp = GET_SIZE(HDRP(bp)) - size;
	if(temp<4*WSIZE)
	{
		if(GET(PRED(bp))!=heap_listp)
			PUT(SUCC(GET(PRED(bp))), GET(SUCC(bp)));
		else
			PUT(heap_listp, GET(SUCC(bp)));
		if(GET(SUCC(bp))!=NULL)
			PUT(PRED(GET(SUCC(bp))), GET(PRED(bp)));
		PUT(HDRP(bp),GET(HDRP(bp))|0x1);
		PUT(HDRP(NEXT_BLKP(bp)),GET(HDRP(NEXT_BLKP(bp)))|0x2);
	}
	else
	{
```

```
		PUT(HDRP(bp),((GET(HDRP(bp))|0x1)&0x7)|size);
		PUT(FTRP(bp),GET(HDRP(bp)));//为了获得这个块的pred和succ
		bp = NEXT_BLKP(bp);
		PUT(HDRP(bp),PACK(temp, 0x2));
		PUT(FTRP(bp),PACK(temp, 0x2));
		PUT(PRED(bp), GET(PRED(PREV_BLKP(bp))));
		PUT(SUCC(bp), GET(SUCC(PREV_BLKP(bp))));
		if(GET(PRED(bp))!=heap_listp)
			PUT(SUCC(GET(PRED(bp))), (unsigned int)bp);
		else
			PUT(heap_listp, (unsigned int)bp);
		if(GET(SUCC(bp))!=NULL)
			PUT(PRED(GET(SUCC(bp))), (unsigned int)bp);
	}
}
```

version2.1:

```
void *find_fit(size_t size)
{
	char* temp;
	temp = GET(heap_listp);
	size_t min=0;
	char* tempp=NULL;
	while(temp!=NULL)
	{
		if((!min&&(size<=GET_SIZE(HDRP(temp))))||(min&&(GET_SIZE(HDRP(temp))-size<min)))
		{
			min=GET_SIZE(HDRP(temp));
			tempp=temp;
		}
		temp=GET(SUCC(temp));
	}
	return tempp;
}
```

4、实验过程中遇到的问题及解决方法

问题：实验中碰到了很多的段错误

解决方法：通过在网上学习了gdb的使用，以及托了《调试九法》这本书给予的建议，整个调试过程是相对顺利且愉快的

问题：很多地方如空闲块的脚部，下一个块的头部脚步要如何设置存在疑惑

解决方法：通过一些试编写，以及通过阅读csapp后得到了答案

5、实验总结

通过本次练习我熟悉了gdb的使用方法，也按照一些调试规则进行了尝试，整个实验个人是做的很开心的，希望能多有一些这样类型的实验。