Contiguous Memory Allocation

Question

This project will involve managing a contiguous region of memory of size MAX where addresses may range from 0 ... MAX− 1. Your program must respond to four different requests:

1. Request for a contiguous block of memory
2. Release of a contiguous block of memory
3. Compact unused holes of memory into one single block
4. Report the regions of free and allocated memory
5. Three ways to allocating memory

Answer

Request memory

Allocating memory

1. worst fit

用简单的冒泡搜索当前内存块的最大值，如果最大值大于request的内存，就返回该block，否则返回NULL，代码如下：

```
struct process * worst_fit (int memory) {
    struct process * ptr = head;
    struct process * worst_ptr = NULL;
    int vol, worst_vol;
    worst_vol = -1;

    while (ptr != tail) {
        vol = (ptr->next->base - (ptr->base + ptr->limit));
        if (vol > worst_vol) {
            worst_vol = vol;
            worst_ptr = ptr;
        }
        ptr = ptr->next;
    }

    if (memory <= worst_vol)
        return worst_ptr;
    else
        return NULL;
}
```

2. best fit

和worst类似，不同之处在于冒泡时需要增加大于request的条件，代码如下：

```
struct process * best_fit (int memory) {
    struct process * ptr = head;
    struct process * best_ptr = NULL;
    int vol, min_vol;
    min_vol = tail->base + 1; // unreachable in this case

    while (ptr != tail) {
        vol = (ptr->next->base - (ptr->base + ptr->limit));
        if (memory <= vol && vol < min_vol) {
            min_vol = vol;
            best_ptr = ptr;
        }
        ptr = ptr->next;
    }

    return best_ptr;
}
```

3. first fit

遍历，如果block大于request就return即可

```
struct process * first_fit (int memory) {
    struct process * ptr = head;
    int vol;
    while (ptr != tail) {
        vol = (ptr->next->base - (ptr->base + ptr->limit));
        if (memory > vol) {
        /* not fit */
            ptr = ptr->next;
        } else {
            return ptr;
        }
    }
    return NULL;
}
```

Release memory

遍历链表，将pid分配的内存free即可，代码如下

```
bool release_memory (char * pid) {
    struct process * ptr = head->next;
    struct process * prev = head;
    bool flag = false; // true if at least one found

    while (ptr != tail) {
        if (strcmp (pid, ptr->pid) == 0) {
        /* process found */
            prev->next = ptr->next; // remove partition
            free (ptr); // release allocated memory
            ptr = prev->next; // note prev need not be changed
```

```
            flag = true; // set flag because found
        } else {
        /* not this process, try next */
            prev = ptr;
            ptr = ptr->next;
        }
    }
    return flag;
}
```

STAT

同样是遍历链表，判断是否use，然后输出即可。代码如下：

```
void show_status (void) {
    struct process * ptr = head->next;
    int start = 0;
    while (ptr != NULL) {
        if (ptr->base != start) {
        /* not adjacent partitions, exist unused memory */
            printf ("Addresses [%d:%d] Unused\n", start, ptr->base - 1);
        }
        if (ptr->limit != 0) {
        /* ptr->limit != 0 means this is not TAIL process */
            start = ptr->base + ptr->limit - 1;
            printf ("Addresses [%d:%d] Process %s\n", ptr->base, start, ptr->pid);
            ++ start; // increment start, because [base : base+limit-1] is allocated to
this process, plus one enters the next partition
        }
        ptr = ptr->next;
    }
}
```

Compact

将每个链表标注的空间连接起来即可

代码如下：

```
void compaction (void) {
    /* a very convenient implementation because of the data structure I use */
    struct process * ptr = head;

    while (ptr->next != tail) {/* stop one process before TAIL */
    /* set the following process, set base = ptr->base + ptr->limit */
        ptr->next->base = ptr->base + ptr->limit;
        ptr = ptr->next;
    }
}
```

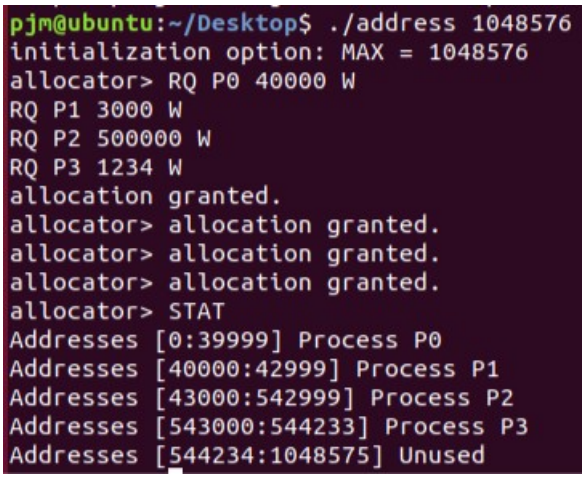
Clean memory

把每段process free掉即可，代码如下

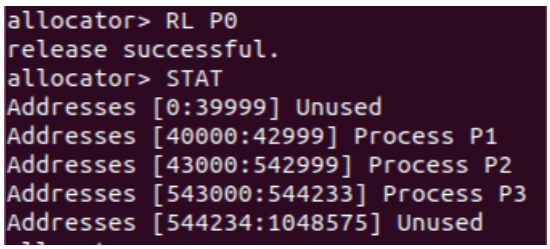
```
void clear_memory (void) {
    struct process * ptr = head;
    struct process * tmp;
    while (ptr != NULL) {
        tmp = ptr;
        ptr = ptr->next;
        free (tmp);
    }
}
```

运行结果

Request



Release



Worst fit

```
allocator> STAT
Addresses [0:39999] Unused
Addresses [40000:42999] Process P1
Addresses [43000:542999] Process P2
Addresses [543000:544233] Process P3
Addresses [544234:584232] Process P4
Addresses [584233:1048575] Unused
allocator> RQ P5 123 W
allocation granted.
allocator> STAT
Addresses [0:39999] Unused
Addresses [40000:42999] Process P1
Addresses [43000:542999] Process P2
Addresses [543000:544233] Process P3
Addresses [544234:584232] Process P4
Addresses [584233:584355] Process P5
Addresses [584356:1048575] Unused
```

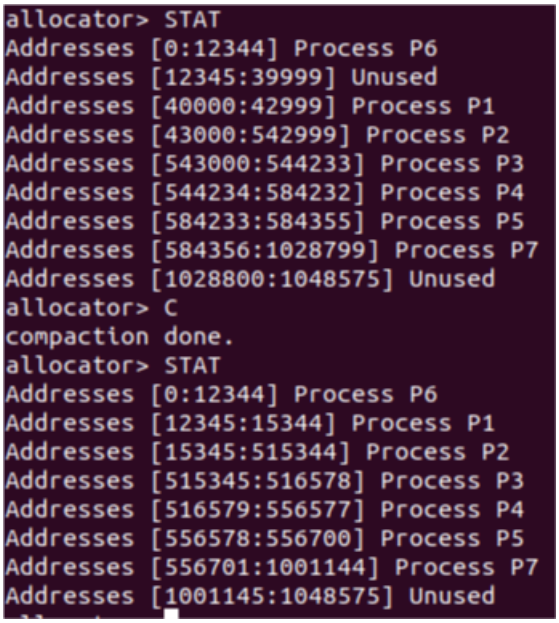
First fit

```
allocator> STAT
Addresses [0:39999] Unused
Addresses [40000:42999] Process P1
Addresses [43000:542999] Process P2
Addresses [543000:544233] Process P3
Addresses [544234:584232] Process P4
Addresses [584233:584355] Process P5
Addresses [584356:1048575] Unused
allocator> RQ P6 12345 F
allocation granted.
allocator> STAT
Addresses [0:12344] Process P6
Addresses [12345:39999] Unused
Addresses [40000:42999] Process P1
Addresses [43000:542999] Process P2
Addresses [543000:544233] Process P3
Addresses [544234:584232] Process P4
Addresses [584233:584355] Process P5
Addresses [584356:1048575] Unused
```

Best fit

```
allocator> STAT
Addresses [0:12344] Process P6
Addresses [12345:39999] Unused
Addresses [40000:42999] Process P1
Addresses [43000:542999] Process P2
Addresses [543000:544233] Process P3
Addresses [544234:584232] Process P4
Addresses [584233:584355] Process P5
Addresses [584356:1048575] Unused
allocator> RQ P7 444444 B
allocation granted.
allocator> STAT
Addresses [0:12344] Process P6
Addresses [12345:39999] Unused
Addresses [40000:42999] Process P1
Addresses [43000:542999] Process P2
Addresses [543000:544233] Process P3
Addresses [544234:584232] Process P4
Addresses [584233:584355] Process P5
Addresses [584356:1028799] Process P7
Addresses [1028800:1048575] Unused
```

Compact


allocator> STAT
Addresses [0:12344] Process P6
Addresses [12345:39999] Unused
Addresses [40000:42999] Process P1
Addresses [43000:542999] Process P2
Addresses [543000:544233] Process P3
Addresses [544234:584232] Process P4
Addresses [584233:584355] Process P5
Addresses [584356:1028799] Process P7
Addresses [1028800:1048575] Unused
allocator> C
compaction done.
allocator> STAT
Addresses [0:12344] Process P6
Addresses [12345:15344] Process P1
Addresses [15345:515344] Process P2
Addresses [515345:516578] Process P3
Addresses [516579:556577] Process P4
Addresses [556578:556700] Process P5
Addresses [556701:1001144] Process P7
Addresses [1001145:1048575] Unused