中山大學

20336014

D403

2022 6 4

1.

.

2.

best-fit, first-fit .

3.

FIFO, LRU .

4.

“ ” , , .

1.

(1) Bitmap

, , , Bitmap. Bitmap , , `char` .
Bitmap , (best-fit). :

```c
typedef struct {
    int length;
    char *bitmap;
    IntervalHeap heap;
    QueueInt q;
} Bitmap;

void bm_init(Bitmap *bm, char *bitmap, int length);

Bool bm_get(Bitmap *bm, int index);

void bm_set(Bitmap *bm, int index, Bool status);

int bm_size(Bitmap *bm);

int bm_allocate(Bitmap *bm, int count);

void bm_release(Bitmap *bm, int index, int count);
```

1: Bitmap

```c
void bm_init(Bitmap *bm, char *bitmap, int length) {
    bm->bitmap = bitmap;
    bm->length = length;

    int bytes = ceil(length, 8);
    memset(bitmap, 0, bytes);

    // ih_init(&bm->heap);
    // qi_init(&bm->q);
}

Bool bm_get(Bitmap *bm, int index) {
    int pos = index / 8;
    int offset = index % 8;

    return (bm->bitmap[pos] & (1 << offset));
}
```

2: Bitmap get

```c
void bm_set(Bitmap *bm, int index, Bool status) {
    int pos = index / 8;
    int offset = index % 8;

    bm->bitmap[pos] = bm->bitmap[pos] & (~(1 << offset));

    if (status) {
        bm->bitmap[pos] = bm->bitmap[pos] | (1 << offset);
    }
}
```

3: set

```c
int bm_allocate(Bitmap *bm, int count) {
    if (count == 0) {
        return -1;
    }

    int index, empty, start;

    index = 0;
    while (index < bm->length) {
        while (index < bm->length && bm_get(bm, index)) {
            ++index;
        }

        if (index == bm->length) {
            return -1;
        }

        empty = 0;
        start = index;

        while ((index < bm->length) && (!bm_get(bm, index)) && (empty < count)) {
            ++empty;
            ++index;
        }

        if (empty == count) {
            for (int i = 0; i < count; ++i) {
                bm_set(bm, start + i, true);
            }

            return start;
        }
    }
    return -1;
}
```

4:

(2)

Bitmap PCB , , , Bitmap , .
Bitmap , . Bitmap . :

```c
typedef struct {
    Bitmap resources;
    int start_addr;
} AddressPool;
```

5:

```c
void ap_init(AddressPool *ap, char* bitmap, int length, int start_addr) {
    bm_init(&ap->resources, bitmap, length);
    ap->start_addr = start_addr;
}

int ap_allocate(AddressPool *ap, int count) {
    uint32 start = bm_allocate_with_fifo(&ap->resources, count);
    return (start == -1) ? -1 : (start * PAGE_SIZE + ap->start_addr);
}

void ap_release(AddressPool *ap, int addr, int amount) {
    bm_release(&ap->resources, (addr - ap->start_addr) / PAGE_SIZE, amount);
}
```

6:

(3)

, .

, . . , , mbr , (

), .

(4)

, . x86 , , 4KiB . 32 , .

, :

1. , .
2. 1MiB (0x00000000 ~ 0x000fffff), , “ ”.
3. 3GiB . 768 , , .
4. cr3 , cr0 PG 1, .

:

```
enum AddressPoolType {
    USER,
    KERNEL
};

typedef struct {
    int total_memory;
    AddressPool kernel_phy;
    AddressPool user_phy;
    AddressPool kernel_vir;
    QueueInt fifo_pages;
} MemoryManeger;
```

7:

```
int mm_allocate_physical_pages(MemoryManeger *mm, enum AddressPoolType type, int count) {
    int start = -1;

    if (type == KERNEL) {
        start = ap_allocate(&mm->kernel_phy, count);
    } else if (type == USER) {
        start = ap_allocate(&mm->user_phy, count);
    }

    return (start == -1) ? 0:start;
}

void mm_release_physical_pages(MemoryManeger *mm, enum AddressPoolType type, int paddr, int count)
    if (type == KERNEL) {
        ap_release(&mm->kernel_phy, paddr, count);
    } else if (type == USER) {
        ap_release(&mm->user_phy, paddr, count);
    }
}
```

8:

```c
void mm_open_page_mechanism(MemoryManeger *mm) {
    int *directory = (int *)PAGE_DIRECTORY;
    int *page = (int *)(PAGE_DIRECTORY + PAGE_SIZE);

    memset(directory, 0, PAGE_SIZE);
    memset(page, 0, PAGE_SIZE);

    int addr = 0;
    for (int i = 0; i < 256; ++i) {
        page[i] = addr | 0x7;
        addr += PAGE_SIZE;
    }

    directory[0] = ((int)page) | 0x07;
    directory[768] = directory[0];
    directory[1023] = ((int) directory) | 0x7;

    asm_init_page_reg(directory);

    printf("page mechanism enabled\n");
}
```

9:

, .

2.

worst-fit .

(1)

worst-fit , . , . , .

(2)

. , . :

```c
//内存空隙的起始位置和长度
typedef struct {
    int size;
    int start_addr;
} Interval;

// 优先队列
typedef struct {
    int count;
    Interval intervals[MAX_INTERVAL];
} IntervalHeap;

// 初始化
void ih_init(IntervalHeap* ih);
// 极大堆化
void ih_adjust(IntervalHeap* ih);
// 弹出
Interval ih_pop(IntervalHeap* ih);
// 压入
void ih_push(IntervalHeap *ih, Interval x);
// 输出
void ih_print(IntervalHeap *ih);
```

10: ( )

(3)

, :

1. Bitmap , .
2. , .

3. Bitmap .

:

```
// 从bitmap获取空隙的信息
void bm_get_intervals(Bitmap *bm) {
    Bool is_interval = false;
    Interval temp = {0, 0};
    for (int i = 0; i < bm->length; ++i) {
        Bool is_allocated = bm_get(bm, i);
        if (is_allocated && is_interval) {
            is_interval = false;
            ih_push(&bm->heap, temp);
        } else if (!is_allocated && is_interval) {
            temp.size += 1;
        } else if (!is_allocated && !is_interval) {
            is_interval = true;
            temp.start_addr = i;
            temp.size = 1;
        }
    }

    if (is_interval) {
        ih_push(&bm->heap, temp);
    }
}
```

11:

```c
int bm_worse_fit(Bitmap *bm, int count) {
    ih_init(&bm->heap);
    bm_get_intervals(bm);
    // ih_print(&bm->heap);

    Interval worse = ih_pop(&bm->heap);

    if (count > worse.size) {
        return -1;
    } else {
        for (int i = 0; i < count; ++i) {
            bm_set(bm, worse.start_addr + i, true);
        }

        return worse.start_addr;
    }
}
```

12: worst-fit

, worst-fit .

3.

FIFO . . .

(1)

, :

```c
typedef struct {
    int data[MAX_QUEUE];
    int count;
} QueueInt;

void qi_init(QueueInt* qi);

void qi_push(QueueInt* qi, int x);

int qi_pop(QueueInt *qi);

Bool qi_empty(QueueInt *qi);

void qi_print(QueueInt *qi);
```

13:

(2) FIFO

:

1. , , .
2. , .

:

```c
int bm_allocate_with_fifo(Bitmap *bm, int count) {
    int result = bm_allocate(bm, count);

    while (result == -1) {
        if (!qi_empty(&bm->q)) {
            int poped = qi_pop(&bm->q);
            bm_release(bm, poped, 1);
            result = bm_allocate(bm, count);
        } else {
            return -1;
        }
    }

    for (int i = 0; i < count; ++i) {
        qi_push(&bm->q, result + i);
    }

    return result;
}
```

14: FIFO

4.

.

(1)

, , 0xc0100000:

(2)

:

1. n .
2. n ( ).
3. .

:

```
Bool mm_connect_phy_vir_pages(MemoryManeger *mm, int vaddr, int paddr) {
    int *pde = (int *)to_pde(vaddr);
    int *pte = (int *)to_pte(vaddr);

    if (!(*pde & 0x00000001)) {
        int page = mm_allocate_physical_pages(mm, KERNEL, 1);

        if (!page) {
            return false;
        }

        *pde = page | 0x7;

        char *page_ptr = (char *)(((int)pte) & 0xfffff000);
        memset(page_ptr, 0, PAGE_SIZE);
    }
    *pte = paddr | 0x7;

    return true;
}
```

15:

(3)

:

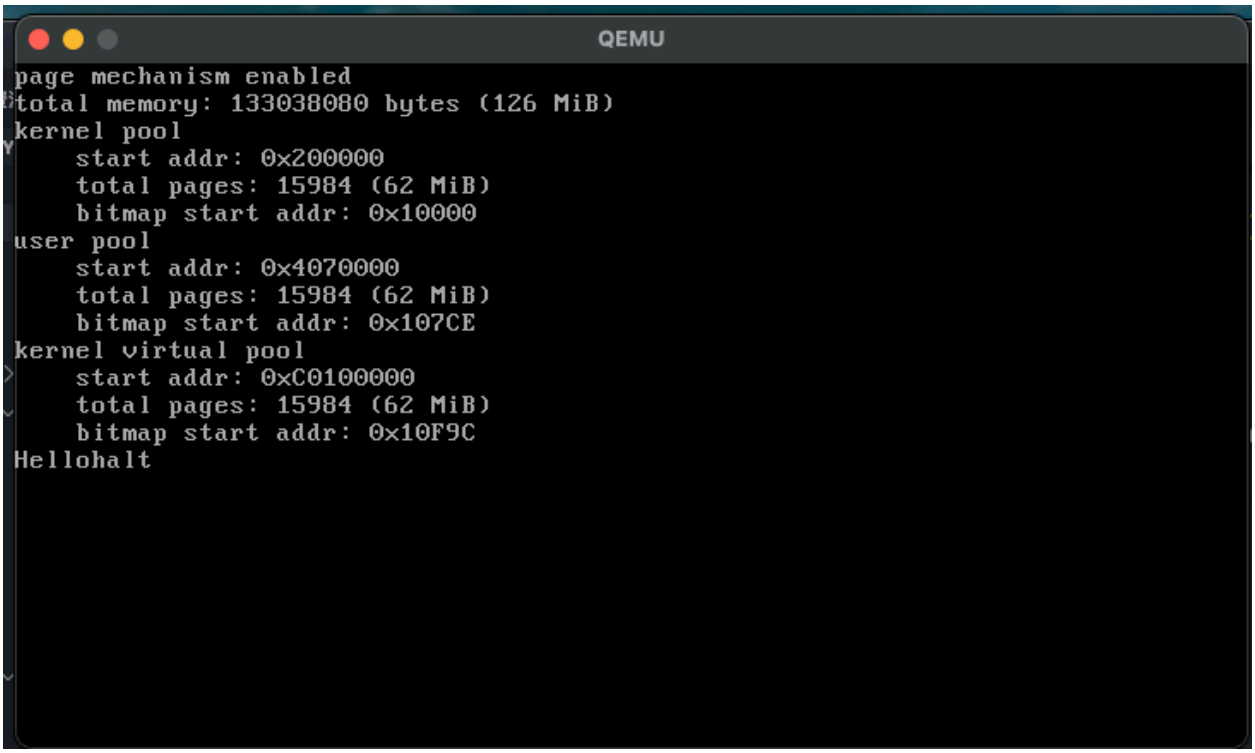
1. .
2. .

```
int vaddr2paddr(int vaddr) {
    int *pte = (int *)to_pte(vaddr);
    int page = (*pte) & 0xfffff000;
    int offset = vaddr & 0xfff;
    return (page + offset);
}
```

16:

, .

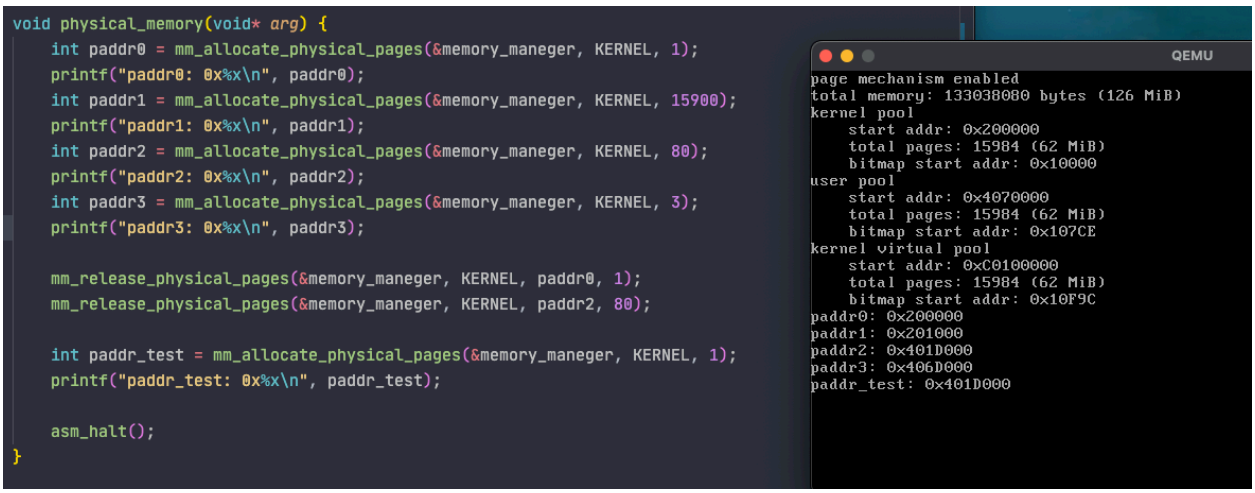
:



17:

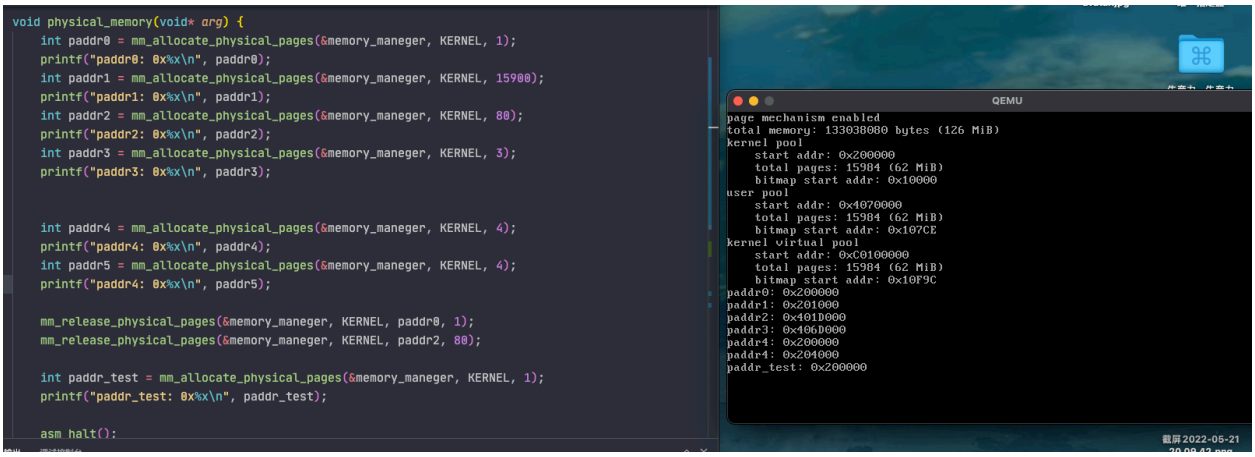
.

worst-fit , : 1 80 , , 80 .



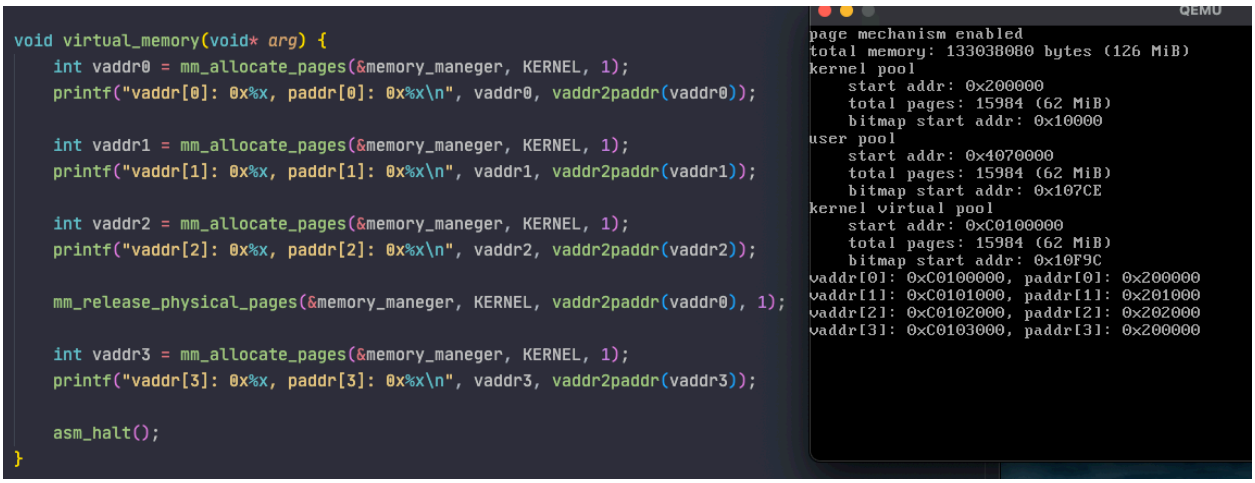
18: worst-fit

, , .
FIFO , : , 4 . 0-7 .



19: FIFO

, .
, : , , , . , .



20:

.