ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

Course: OPERATING SYSTEM

Assignment #2 - Simple Operating System Report

GVHD: Huỳnh Nam

TP. HỒ CHÍ MINH, THÁNG 4/2019

Danh sách thành viên

1. Nguyễn Tiến Phát - 1712572
2. Nguyễn Lục Sâm Bảo - 1710598

Mục lục

1 Scheduling **3**

2 Memory Management **8**
2.1 Result - State of RAM 8
2.2 Implementation 10
2.2.1 Get page table - Lấy bảng phân trang 10
2.2.2 Mapping from virtual address to physical address 11
2.2.3 Checking amount memory to allocted for process 12
2.2.4 Alloc memory 13
2.2.5 Free memory 14
2.3 Question:What is the advantage and disadvantage of segmentation with paging 16

3 Overall - Synchronization **17**

4 Conclusion **19**

1 Scheduling

Đây là phần hiện thực Scheduler, Scheduler trong OS của ta có nhiệm vụ sắp xếp các process. Do hệ thống áp dụng feedback queue priority nên OS có 2 queue là ready queue và run queue. Trong đó ready queue là nơi các process sẽ đợi để được quyền sử dụng CPU. Sau khi hết thời gian, Process sẽ được chuyển về run queue.

Nhiệm vụ của ta là sẽ hoàn thiện enqueue() và dequeue trong file queue.c. Enqueue() có chức năng thêm process mới vào queue, còn dequeue() sẽ trả về process có priority cao nhất.

```
void enqueue(struct queue_t * q, struct pcb_t * proc) {
      q->proc[q->size]=proc;
      q->size ++;
}

struct pcb_t * dequeue(struct queue_t * q) {
      struct pcb_t * temp;
      for (int i = 0; i<=(q->size -2); i++){
            if(q->proc[i]->priority > q->proc[i+1]->priority){
                  temp = q->proc[i];
                  q->proc[i] = q-> proc[i+1];
                  q->proc[i+1] = temp;
            }
      }
      temp = q->proc[q->size-1];
      q->size --;
      return temp;
}
```

Và hiện thực get_proc trong sched.c. Get_proc có nhiệm vụ lấy process từ ready queue. Và khi ready queue trống sẽ chuyển tất cả process từ run queue về ready queue.

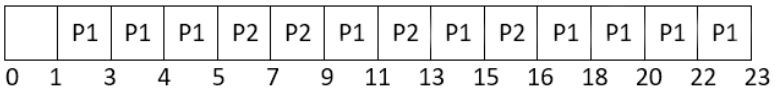
```
struct pcb_t * get_proc(void) {
      struct pcb_t * proc;
      if(queue_empty()){
            return 0;
      }
      if(empty(&ready_queue)){
            int m = run_queue.size;
            for(int i = 0; i <= (m -1); i++){

                  pthread_mutex_lock(&queue_lock);
                  proc = dequeue(&run_queue);
                  pthread_mutex_unlock(&queue_lock);

                  pthread_mutex_lock(&queue_lock);
                  enqueue(&ready_queue, proc);
                  pthread_mutex_unlock(&queue_lock);
            }
            get_proc();

      }else{
            pthread_mutex_lock(&queue_lock);
            proc = dequeue(&ready_queue);
            pthread_mutex_unlock(&queue_lock);
            return proc;
      }
}
```

Sau khi hoàn thiện file queue.c và sched.c, ta dùng lệnh Make sched rồi Make test_sched để kiểm tra kết quả

Kết quả test run sched_0:

```
File  Edit  View  Search  Terminal  Help
------ SCHEDULING TEST 0 ------------------------------------------
./os sched_0
Time slot   0
        Loaded a process at input/proc/s0, PID: 1
Time slot   1
        CPU 0: Dispatched process  1
Time slot   2
Time slot   3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot   4
        Loaded a process at input/proc/s1, PID: 2
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   6
Time slot   7
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   8
Time slot   9
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  1
Time slot  10
Time slot  11
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot  12
Time slot  13
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  1
Time slot  14
Time slot  15
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot  16
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  1
Time slot  17
Time slot  18
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  19
Time slot  20
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  21
Time slot  22
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  23
        CPU 0: Processed  1 has finished
        CPU 0 stopped
```

Biểu đồ Gantt Sched_0:



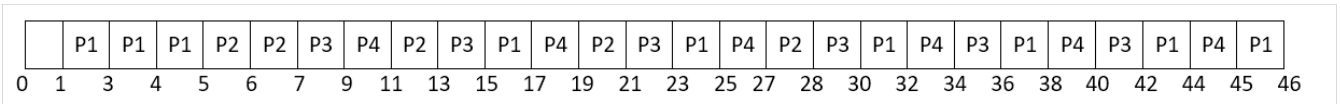
Process	Time arrived	Priority
P1	0	12
P2	4	20

- Time 0, process 1 được nạp vào ready queue.
- Time 1, process 1 được đưa vào CPU để thực thi với quantum = 2.
- Time 3, process 1 hết thời gian. Do vẫn chưa xong nên process 1 đuọc đưa vào run queue. Hiện tại trong ready queue trống, vì thế, đưa process 1 từ run queue quay về ready queue để chạy tiếp.
- Time 4, process 2 được nạp vào ready queue.
- Time 5, process 1 hết giờ, được chuyển sang run queue. Trong ready queue đang có process 2 -> đưa process 2 vào CPU.
- Time 7, process 2 hết giờ, được chuyển sang run queue. Ready queue bây giờ trống, ta sẽ chuyển tất cả process trong run queue sang ready queue. Process 2 có priority cao hơn nên được dùng CPU trước.
- Tương tự cho tới Time 16.
- Time 16, Process 2 đã hoàn thành. Process 1 tiếp tục sử dụng CPU,
- Time 23, Process 1 hoàn thành. Test kết thúc.

Kết quả test run sched_1:

```
File  Edit  View  Search  Terminal  Help
------ SCHEDULING TEST 1 ----------------------------------------
./os sched_1
Time slot   0
        Loaded a process at input/proc/s0, PID: 1
Time slot   1
        CPU 0: Dispatched process  1
Time slot   2
Time slot   3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot   4
        Loaded a process at input/proc/s1, PID: 2
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   6
        Loaded a process at input/proc/s2, PID: 3
Time slot   7
        Loaded a process at input/proc/s3, PID: 4
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot   8
Time slot   9
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  10
Time slot  11
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  12
Time slot  13
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  14
Time slot  15
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
Time slot  16
Time slot  17
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  4
Time slot  18
Time slot  19
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  20
Time slot  21
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  22
Time slot  23
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
Time slot  24
Time slot  25
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  4
Time slot  26
Time slot  27
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  28
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  3
Time slot  29
Time slot  30
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
File  Edit  View  Search  Terminal  Help
        CPU 0: Dispatched process  1
Time slot  31
Time slot  32
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  4
Time slot  33
Time slot  34
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  3
Time slot  35
Time slot  36
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  1
Time slot  37
Time slot  38
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  4
Time slot  39
Time slot  40
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  3
Time slot  41
Time slot  42
        CPU 0: Processed  3 has finished
        CPU 0: Dispatched process  1
Time slot  43
Time slot  44
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  4
Time slot  45
        CPU 0: Processed  4 has finished
        CPU 0: Dispatched process  1
Time slot  46
        CPU 0: Processed  1 has finished
        CPU 0 stopped

MEMORY CONTENT:
```

Biểu đồ Gantt Sched_1:



Process	Time arrived	Priority
P1	0	12
P2	4	20
P3	6	20
P4	7	22

- Time 0, process 1 được nạp vào ready queue.
- Time 1, process 1 được đưa vào CPU để thực thi với quantum = 2.
- Time 3, process 1 hết thời gian. Do vẫn chưa xong nên process 1 đuọc đưa vào run queue. Hiện tại trong ready queue trống, vì thế, đưa process 1 từ run queue quay về ready queue để chạy tiếp.
- Time 4, process 2 được nạp vào ready queue.
- Time 5, process 1 hết giờ, được chuyển sang run queue. Trong ready queue đang có process 2 -> đưa process 2 vào CPU.
- Time 6, process 3 được nạp vào ready queue.
- Time 7, process 4 được nạp vào ready queue. Process 2 hết thời gian, được đưa vào run queue. Trong ready queue đang có process 3 và process 4, process 3 có priority cao hơn nên được đi trước.
- Time 9, process 3 hết thời gian về run queue. Process 4 được phép dùng CPU.
- Time 11, process 4 hết thời gian về run queue. Ready queue trống -> tất cả process trong run queue về ready queue. Process 2 và 3 cùng có priority cao nhất , mà process 2 đến (arrive) trước nên được sử dụng CPU đầu tiên.
- Time 13, process 2 hết thời gian, về run queue. Process 3 tiếp tục dùng CPU.
- Time 15, process 3 về run queue. Process 1 được ưu tiên vì priority cao hơn.
- Time 17, Process 1 về run queue. Process 4 tiếp tục dùng CPU.
- Time 19, process 4 hêt thời gian, về run queue.
- Ready queue trống. Tiếp tục tựa các bước trên.
- Time 28, process 2 kêt thúc. Process 3 tiếp tục dùng CPU.
- Time 42, process 3 kêt thúc. Process 1 tiếp tục.
- Time 45, process 4 kết thúc. Process 1 tiếp tục.
- Time 46, process 1 kết thúc. Test kết thúc.

Question: Đâu là lợi thế của việc sử dụng *priority feedback queue* so với các thuật toán scheduling mà ta đã được học?

Các process trong ready queue gồm nhiều loại khác nhau như foreground process, background process, v.v.. và mỗi loại process như thế cần có thuật toán scheduling phù hợp. Cho nên ta có thuật toán priority feedback queue sẽ chia ready queue thành nhiều queue cho từng loại process. Mỗi queue sẽ áp dụng thuật toán scheduling thích hợp và có priority khác nhau.

Ngoài ra priority feedback queue cho phép process di chuyển giữa các queue giúp linh hoạt hơn. Trong khi priority queue, các process chỉ cố định chạy trong 1 queue.

2 Memory Management

2.1 Result - State of RAM

REQUIMENT: Show the status of RAM after each memory allocation and deallocation function call.
Test mem_0

```
------ MEMORY MANAGEMENT TEST 0 -----------------------
./mem input/proc/m0
-----------==Allocation==-----------
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
-----------==Allocation==-----------
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
-----------==Deallocation==-----------
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
-----------==Allocation==-----------
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
```

```
------------==Allocation==------------
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
        003e8: 15
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
        03814: 66
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
NOTE: Read file output/m0 to verify your result
```

Test mem_1

```
------ MEMORY MANAGEMENT TEST 1 ------------------------
./mem input/proc/m1
------------==Allocation==------------
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
------------==Allocation==------------
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: 002)
002: 00800-00bff - PID: 01 (idx 002, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 003, nxt: 004)
004: 01000-013ff - PID: 01 (idx 004, nxt: 005)
005: 01400-017ff - PID: 01 (idx 005, nxt: 006)
006: 01800-01bff - PID: 01 (idx 006, nxt: 007)
007: 01c00-01fff - PID: 01 (idx 007, nxt: 008)
008: 02000-023ff - PID: 01 (idx 008, nxt: 009)
009: 02400-027ff - PID: 01 (idx 009, nxt: 010)
010: 02800-02bff - PID: 01 (idx 010, nxt: 011)
```

```
011: 02c00-02fff - PID: 01 (idx 011, nxt: 012)
012: 03000-033ff - PID: 01 (idx 012, nxt: 013)
013: 03400-037ff - PID: 01 (idx 013, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
-----------==Deallocation==-----------
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
-----------==Allocation==------------
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
-----------==Allocation==------------
000: 00000-003ff - PID: 01 (idx 000, nxt: 001)
001: 00400-007ff - PID: 01 (idx 001, nxt: -01)
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
-----------==Deallocation==-----------
002: 00800-00bff - PID: 01 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 001, nxt: 004)
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
-----------==Deallocation==-----------
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
-----------==Deallocation==-----------
NOTE: Read file output/m1 to verify your result (your ir
```

2.2 Implementation

2.2.1 Get page table - Lấy bảng phân trang

Trong simulation này, mỗi địa chỉ được biểu diễn 20 bits, trong đó 5 bit đầu là segment, 5 bit tiếp theo là page và 10 bit cuối là phần offset
Cấu trúc phân đoạn(struct segmet_table_t) bao gồm size của segment_table và một bảng segment. Bảng segment có cấu trúc gồm một biến v_index dùng để lưu 5 bit đầu của địa chỉ là phần segment, và một con trỏ chỉ đến cấu trúc phân trang - bảng phân trang. Ở task này ta sẽ lấy bảng phân trang.
Để lấy được bảng trang, đầu lấy 5 bit đầu của địa chỉ truyền vào, sau đó so sánh với v_index trong seg_table. Nếu tìm được thì return seg_table->table[i].pages, nếu không thì return NULL.
Sau đây là hiện thực task này:

```c
/* Search for page table table from the a segment table */
static struct page_table_t * get_page_table(
        addr_t index,   // Segment level index
        struct seg_table_t * seg_table) { // first level table

    /*
    * TODO: Given the Segment index [index], you must go through each
    * row of the segment table [seg_table] and check if the v_index
    * field of the row is equal to the index
    *
    * */

    int i;
    for (i = 0; i < seg_table->size; i++) {
        // Enter your code here
        if(seg_table->table[i].v_index == index){
            return seg_table->table[i].pages;
        }
    }
    return NULL;
}
```

2.2.2 Mapping from virtual address to physical address

Ở task này chúng ta thực hiện việc ánh xạ địa chỉ bộ nhớ ảo sang bộ nhớ thực. Để hiện thực task này ta thực hiện như sau.Đầu tiên ta lấy địa chỉ page_table->table[i].p_index sau đó dịch trái 10 bit(OFFSET_LEN) rồi or lại với phần offset.
Hiện thực như dưới đây:

```
static int translate(
        addr_t virtual_addr,    // Given virtual address
        addr_t * physical_addr, // Physical address to be returned
        struct pcb_t * proc) {  // Process uses given virtual address

        /* Offset of the virtual address */
    addr_t offset = get_offset(virtual_addr);
    /* The first layer index */
    addr_t first_lv = get_first_lv(virtual_addr);
    /* The second layer index */
    addr_t second_lv = get_second_lv(virtual_addr);

    /* Search in the first level */
    struct page_table_t * page_table = NULL;
    page_table = get_page_table(first_lv, proc->seg_table);
    if (page_table == NULL) {
        return 0;
    }

    int i;
    for (i = 0; i < page_table->size; i++) {
        if (page_table->table[i].v_index == second_lv) {
            /* TODO: Concatenate the offset of the virtual addess
            * to [p_index] field of page_table->table[i] to
            * produce the correct physical address and save it to
            * [*physical_addr]  */
            addr_t p_index = page_table->table[i].p_index;
            *physical_addr = (p_index << OFFSET_LEN) | offset;
            return 1;
        }
    }
    return 0;
}
```

2.2.3 Checking amount memory to allocted for process

Ở task này thực hiện kiểm tra xem bộ nhớ vật lý và bộ nhớ ảo có còn đủ để cung cấp cho process hay không?. Tại vùng nhớ vật lý, ta kiểm duyệt số lượng page còn trống, chưa được process sử dụng, nếu đủ với lượng yêu cầu cần cấp phát thì vùng nhớ vật lý sẵn sàng. Để tối ưu việc quản lý vùng nhớ, ta xây dựng một_mem_stat[NUM_PAGES] để quản lý kích thước, số lượng vùng nhớ trống,.. để tối ưu việc truy xuất vùng nhớ.
Sau đây là hiện thực chức năng:

```
//Checking amount memory to allocted for process
int check_amount_memmory(struct pcb_t * proc , uint32_t num_pages){
    uint32_t number_mem_free = 0;

    //check physical memory
    for(int i = 0 ; i < NUM_PAGES ; i++){
        if(_mem_stat[i].proc == 0){
            if(++number_mem_free == num_pages)
                break;
        }
    }
    if(number_mem_free < num_pages) return 0;
    //check virtual memory
    if(proc->bp + num_pages*PAGE_SIZE >= RAM_SIZE) return 0;
    return 1;
}
```

2.2.4 Alloc memory

Ở task này ta hiện thực các bước sau:

- Duyện vùng nhớ vật lý, tìm các trang chưa được sử dụng, gán các trang này cho process sử dụng
- Cập nhật giá trị của next, last next bằng -1
- Trên vùng nhớ ảo, dựa trên địa chỉ cấp phát, tình từ địa chỉ bắt đầu, ta tìm được các segment, page của nó. Từ đó cập nhật các segment_table, page_table tương ứng.

Hiện thực như sau:

```
void alloc_ret_mem(addr_t ret_mem, uint32_t num_pages, struct pcb_t * proc){
    int count_pages = 0;    //count pages
    int last_alloc_pages = -1;          //get [next] = -1 at last page
    for(int i = 0 ; i< NUM_PAGES ; i++)
    {
        if(_mem_stat[i].proc != 0) continue; //page have used
        _mem_stat[i].proc = proc->pid;          // the page is used by process
        _mem_stat[i].index = count_pages;       //index list of allocated pages
        if(last_alloc_pages > -1){
            _mem_stat[last_alloc_pages].next = i;       //update last page
        }
        last_alloc_pages = i;           //update last page
        //find and creat virtual page_table
        addr_t virtual_address = ret_mem + count_pages*PAGE_SIZE;
        addr_t segment_addr = get_first_lv(virtual_address);
        addr_t page_addr = get_second_lv(virtual_address);
        struct page_table_t* v_page_table = get_page_table(segment_addr, proc->seg_table);
        if(v_page_table == NULL){
            int indx = proc->seg_table->size++;
            proc->seg_table->table[indx].v_index = segment_addr;
            v_page_table
            = proc->seg_table->table[indx].pages
            = (struct page_table_t*)malloc(sizeof(struct page_table_t));
        }
        int idx = v_page_table->size++;
        v_page_table->table[idx].v_index = page_addr;
        v_page_table->table[idx].p_index = i;
        if(++count_pages == num_pages){
            _mem_stat[i].next = -1; break;
        }

    }
}
```

2.2.5 Free memory

Ở task này chúng ta thu hồi vùng nhớ đã cấp phát cho process. Thực hiện các bước như sau:

- Thu hồi địa chỉ vật lý: Chuyển địa chỉ luận lý từ process thành địa chỉ vật lý, sau đó cấp nhật lại _mem_stat[i].proc = 0 dựa trên giá trị tương ứng.
- Tìm tất cả các trang đã cấp phát cho process,sau đó xóa toàn bộ chúng
- Cập nhật lại giá trị break point

Hiện thực như sau:

```
int free_mem(addr_t address, struct pcb_t * proc) {
    pthread_mutex_lock(&mem_lock);
    addr_t v_address = address; // virtual address to free in process
    addr_t p_address = 0;       // physical address to free in memory
    // Find physical page in memory
    if (!translate(v_address, &p_address, proc)) return 1;
    // Clear physical page in memory
    addr_t p_segment_page_index = p_address >> OFFSET_LEN;
    int num_pages = 0; // number of pages in list
    int i;
    for (i=p_segment_page_index; i!=-1; i=_mem_stat[i].next) {
        num_pages++;
        _mem_stat[i].proc = 0; // clear physical memory
    }
    // Clear virtual page in process
    for (i = 0; i < num_pages; i++) {
        addr_t v_addr = v_address + i * PAGE_SIZE;
        addr_t v_segment = get_first_lv(v_addr);
        addr_t v_page = get_second_lv(v_addr);
        struct page_table_t * page_table = get_page_table(v_segment, proc->seg_table);
        if (page_table == NULL) {
            puts("============= Error ===============");
            continue;
        }
        int j;
        for (j = 0; j < page_table->size; j++) {
            if (page_table->table[j].v_index == v_page) {
                int last = --page_table->size;
                page_table->table[j] = page_table->table[last];
                break;
            }
        }
        if (page_table->size == 0) {
            remove_page_table(v_segment, proc->seg_table);
        }
    }
    // Update break pointer
    proc->bp = proc->bp - num_pages*PAGE_SIZE;
    if (1) {
        puts("============  Deallocation  ============");
        dump();
    }
    pthread_mutex_unlock(&mem_lock);
    return 0;
}
```

2.3 Question:What is the advantage and disadvantage of segmentation with paging

- **Advantage**:
  - Tiết kiệm bộ nhớ, sử dụng bộ nhớ hiệu quả
  - Mang các ưu điểm của giải thuật phân trang: Đơn giản việc cấp phát và khắc phục được phân mảnh ngoài.
- **Disadvantage:**
  - Vẫn còn hiện tượng phân mảnh nội

3 Overall - Synchronization

Tiếp theo, điều chúng ta sẽ làm là tổng hợp phần scheduling và memory để hiện thực một OS đơn giản. Để bắt đầu, ta sẽ mô tả môi trường ảo mà ta sẽ chạy giả lập. Tất cả được mô tả trong configure file os_0, os_1.

```
[time slice] [N = Number of CPU] [M = Number of Processes to be run]
[time 0] [path 0]
[time 1] [path 1]
...
[time M-1] [path M-1]
```

Trong file configure sẽ mô tả hạ thống gồm bao nhiêu CPU, process, quantumm.
Chạy lệnh Make all để compile tất cả và kiểm tra kết quả.
Đây là os_0:

```
File  Edit  View  Search  Terminal  Help
./os os_0
Time slot   0
        Loaded a process at input/proc/p0, PID: 1
Time slot   1
        CPU 1: Dispatched process  1
Time slot   2
        Loaded a process at input/proc/p1, PID: 2
        CPU 0: Dispatched process  2
Time slot   3
        Loaded a process at input/proc/p1, PID: 3
Time slot   4
        Loaded a process at input/proc/p1, PID: 4
Time slot   5
Time slot   6
Time slot   7
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  4
Time slot   8
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot   9
Time slot  10
Time slot  11
Time slot  12
Time slot  13
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  1
Time slot  14
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  2
Time slot  15
Time slot  16
```

```
File  Edit  View  Search  Terminal  Help
Time slot  17
        CPU 1: Processed  1 has finished
        CPU 1: Dispatched process  4
Time slot  18
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  3
Time slot  19
Time slot  20
Time slot  21
        CPU 1: Processed  4 has finished
        CPU 1 stopped
Time slot  22
        CPU 0: Processed  3 has finished
        CPU 0 stopped

MEMORY CONTENT:
000: 00000-003ff - PID: 04 (idx 000, nxt: 001)
001: 00400-007ff - PID: 04 (idx 001, nxt: 007)
002: 00800-00bff - PID: 02 (idx 000, nxt: 003)
003: 00c00-00fff - PID: 02 (idx 001, nxt: 004)
004: 01000-013ff - PID: 02 (idx 002, nxt: 005)
        011e7: 0a
005: 01400-017ff - PID: 02 (idx 003, nxt: 006)
006: 01800-01bff - PID: 02 (idx 004, nxt: -01)
007: 01c00-01fff - PID: 04 (idx 002, nxt: 008)
008: 02000-023ff - PID: 04 (idx 003, nxt: -01)
009: 02400-027ff - PID: 03 (idx 000, nxt: 010)
010: 02800-02bff - PID: 03 (idx 001, nxt: 011)
        02814: 64
011: 02c00-02fff - PID: 03 (idx 002, nxt: 012)
012: 03000-033ff - PID: 03 (idx 003, nxt: -01)
014: 03800-03bff - PID: 04 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 04 (idx 001, nxt: 016)
016: 04000-043ff - PID: 04 (idx 002, nxt: 017)
        041e7: 0a
017: 04400-047ff - PID: 04 (idx 003, nxt: 018)
018: 04800-04bff - PID: 04 (idx 004, nxt: -01)
```

Đây là os_1:

```
File  Edit  View  Search  Terminal  Help
NOTE: Read file output/os_0 to verify your result
----- OS TEST 1 -----------------------------------------------------
./os os_1
Time slot   0
Time slot   1
        Loaded a process at input/proc/p0, PID: 1
        CPU 1: Dispatched process  1
Time slot   2
        Loaded a process at input/proc/s3, PID: 2
        CPU 0: Dispatched process  2
Time slot   3
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  1
Time slot   4
        Loaded a process at input/proc/m1, PID: 3
        CPU 2: Dispatched process  3
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot   5
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  1
Time slot   6
        Loaded a process at input/proc/s2, PID: 4
        CPU 3: Dispatched process  4
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
        CPU 2: Put process  3 to run queue
        CPU 2: Dispatched process  3
Time slot   7
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  1
        Loaded a process at input/proc/m0, PID: 5
Time slot   8
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  5
        CPU 3: Put process  4 to run queue
        CPU 3: Dispatched process  4
        CPU 2: Put process  3 to run queue
        CPU 2: Dispatched process  2
Time slot   9
        Loaded a process at input/proc/p1, PID: 6
        CPU 1: Put process  1 to run queue
        CPU 1: Dispatched process  6
Time slot  10
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  2
        CPU 3: Put process  4 to run queue
        CPU 3: Dispatched process  3
        CPU 0: Put process  5 to run queue
        CPU 0: Dispatched process  1
Time slot  11
        Loaded a process at input/proc/s0, PID: 7
        CPU 1: Put process  6 to run queue
        CPU 1: Dispatched process  7
Time slot  12
        CPU 2: Put process  2 to run queue
        CPU 2: Dispatched process  4
        CPU 0: Processed  1 has finished
        CPU 0: Dispatched process  2
        CPU 3: Put process  3 to run queue
        CPU 3: Dispatched process  5
Time slot  13
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  6
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  14
        CPU 2: Put process  4 to run queue
        CPU 2: Dispatched process  3
        CPU 3: Put process  5 to run queue
        CPU 3: Dispatched process  4
Time slot  15
        CPU 0: Put process  6 to run queue
```

```
File  Edit  View  Search  Terminal  Help
        CPU 2: Dispatched process  3
        CPU 3: Put process  5 to run queue
        CPU 3: Dispatched process  4
Time slot  15
        CPU 0: Put process  6 to run queue
        CPU 0: Dispatched process  5
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  16
        Loaded a process at input/proc/s1, PID: 8
        CPU 3: Put process  4 to run queue
        CPU 3: Dispatched process  8
        CPU 2: Processed  3 has finished
        CPU 2: Dispatched process  6
Time slot  17
        CPU 0: Put process  5 to run queue
        CPU 0: Dispatched process  4
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  5
        CPU 3: Put process  8 to run queue
        CPU 3: Dispatched process  8
Time slot  18
        CPU 2: Put process  6 to run queue
        CPU 2: Dispatched process  7
        CPU 1: Processed  5 has finished
        CPU 1: Dispatched process  6
Time slot  19
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
Time slot  20
        CPU 3: Put process  8 to run queue
        CPU 3: Dispatched process  8
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
        CPU 1: Put process  6 to run queue
File  Edit  View  Search  Terminal  Help
        CPU 3: Processed  8 has finished
        CPU 3 stopped
Time slot  24
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
Time slot  25
Time slot  26
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  7
Time slot  27
        CPU 2: Processed  7 has finished
        CPU 2 stopped

MEMORY CONTENT:
002: 00800-00bff - PID: 05 (idx 000, nxt: 003)
        00be8: 15
003: 00c00-00fff - PID: 05 (idx 001, nxt: -01)
004: 01000-013ff - PID: 05 (idx 000, nxt: 005)
005: 01400-017ff - PID: 05 (idx 001, nxt: 006)
        01414: 64
006: 01800-01bff - PID: 05 (idx 002, nxt: 007)
007: 01c00-01fff - PID: 05 (idx 003, nxt: 008)
008: 02000-023ff - PID: 05 (idx 004, nxt: -01)
011: 02c00-02fff - PID: 06 (idx 000, nxt: 012)
012: 03000-033ff - PID: 06 (idx 001, nxt: 013)
013: 03400-037ff - PID: 06 (idx 002, nxt: 014)
014: 03800-03bff - PID: 06 (idx 003, nxt: -01)
021: 05400-057ff - PID: 01 (idx 000, nxt: -01)
024: 06000-063ff - PID: 05 (idx 000, nxt: 025)
        06014: 66
025: 06400-067ff - PID: 05 (idx 001, nxt: -01)
031: 07c00-07fff - PID: 06 (idx 000, nxt: 032)
032: 08000-083ff - PID: 06 (idx 001, nxt: 033)
033: 08400-087ff - PID: 06 (idx 002, nxt: 034)
        085e7: 0a
034: 08800-08bff - PID: 06 (idx 003, nxt: 035)
```

4 Conclusion

Bảng phân chia công việc

Thành viên	Công việc	Tỉ lệ đóng góp
Nguyễn Tiến Phát	Memory Management, Sync, viết Report	50%
Nguyễn Lục Sâm Bảo	Scheduling, Sync, viết Report	50%

Reference

https://www.gnu.org/prep/standards/html node/Writing-C.html