ĐẠI HỌC BÁCH KHOA THÀNH PHỐ HỒ CHÍ MINH

KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

HỆ ĐIỀU HÀNH

Assignment 2

BÁO CÁO BÀI TẬP LỚN 2

Giáo viên hướng dẫn: Hoàng Lê Hải Thanh

Sinh viên: 1812660-Nguyễn Văn Khoa

1927001-Lê Nhựt Anh

1814226-Hứa Phước Thuận

1812962-Đoàn Tấn Lộc

PHÂN CHIA CÔNG VIỆC

Sinh viên	**Nhiệm vụ**
Lê Nhựt Anh	Hiện thực scheduler, memory management, put it all
Hứa Phước Thuận	Hiện thực put it all ,viết báo cáo
Nguyễn Văn Khoa	Thực hiện put it all ,viết báo cáo
Đoàn Tấn Lộc	Viết báo cáo ,memory management,

MỤC LỤC

IMPLEMENTATION

1. Scheduler

Hàm enqueue(): hàm thực thi việc đưa xếp process vào hàng đợi ưu tiên, nên ta chỉ cần dùng vòng lặp đưa vào hàng đợi.

Hàm enqueue() được hiện thực như bên dưới

```
9   void enqueue(struct queue_t * q, struct pcb_t * proc) {
10      /* TODO: put a new process to queue [q] */
11      if (q->size == MAX_QUEUE_SIZE) return;
12      else {
13          q->proc[q->size++] = proc;
14      }
15  }
```

Hàm dequeue(): thực thi nhiệm vụ trả về pcb có độ ưu tiên cao nhất trong hàng đợi và loại bỏ nó khỏi hàng đợi

Hàm dequeue() được hiện thực như hình bên dưới:

```
17  struct pcb_t * dequeue(struct queue_t * q) {
18      /* TODO: return a pcb whose prioprity is the highest
19       * in the queue [q] and remember to remove it from q
20       * */
21      if (q->size == 0 ) return NULL;
22      int idx = 0 , j = 1;
23      int max_priority= q->proc[0]->priority;
24      for (j = 1; j<q->size; j++){
25          if (q->proc[j]->priority > max_priority){
26              idx = j;
27              max_priority=q->proc[j]->priority;
28          }
29      }
30      struct pcb_t *out = q->proc[idx];
31      for (j = idx+1;j< q->size; j++){
32          q->proc[j-1] = q->proc[j];
33      }
34      q->size--;
35      return out;
36  }
```

Hàm get_proc(): thực thi nhiệm vụ lấy thông tin PCB của process đang nằm trong hàng đợi ready_queue. Nếu hàng đợi ready_queue trống thì chuyển PCB của process từ run_queue trở lại hàng đợi ready_queue trước khi lấy process từ hàng đợi ready_queue.

Hàm get_proc(): được hiện thực như bên dưới.

```
20  struct pcb_t * get_proc(void) {
21      struct pcb_t * proc = NULL;
22      /*TODO: get a process from [ready_queue]. If ready queue
23       * is empty, push all processes in [run_queue] back to
24       * [ready_queue] and return the highest priority one.
25       * Remember to use lock to protect the queue.
26       * */
27
28      pthread_mutex_lock(&queue_lock);
29      /*khi hàng chờ rỗng */
30      if (empty(&ready_queue)) {
31          /* xóa PCB của process chờ ở run_queue bỏ vô lại ready_queue*/
32          while (!empty(&run_queue)) {
33              enqueue(&ready_queue, dequeue(&run_queue));
34          }
35      }
36
37      if (!empty(&ready_queue)) {
38          proc = dequeue(&ready_queue);
39      }
40      pthread_mutex_unlock(&queue_lock);
41      return proc;
42  }
43
```

Scheduling: draw Gantt diagram describing how processes are executed by the CPU.

Test 0

CPU 0		P1				P2				P1		P2	
Time	0	1	2	3	4	5	6	7	8	9	10	11	12
CPU 0	P1		P2	P1									
Time	13	14	15	16	17	18	19	20	21	22			

CPU xử lý 2 process với P1 và P2 trong 22 time slot

Test 1

CPU 0			P1				P2		P3		P4		P2
Time	0	1	2	3	4	5	6	7	8	9	10	11	12
CPU 0	P2	P3		P1		P4		P2		P3		P1	
Time	13	14	15	16	17	18	19	20	21	22	23	24	25
CPU 0	P4		P2	P3		P1		P4		P3		P1	
Time	26	27	28	29	30	31	32	33	34	35	36	37	38
CPU 0	P4		P3		P1		P4	P1					
Time	39	40	41	42	43	44	45	46					

CPU xử lý với 4 process P1,P2,P3,P4 theo lượt đồ trên

Question: What is the advantage of using priority feedback queue in comparison with other scheduling algorithms you have learned?

a. **Giải thuật priority feedback queue sử dụng 2 hàng đợi là ready_queue và run_queue:**

+ ready_queue: hàng đợi chứa các process sẵn sàng thực thi, process nào có độ ưu tiên cao hơn thì sẽ được CPU chọn xử lý trước, ngược lại process nào có độ ưu tiên thấp thì CPU sẽ xử lý sau.

+ run_queue: hàng đợi chứa các process đã được CPU xử lý, khi hàng đợi ready_queue rỗng thì các process nằm trong hàng đợi run_queue sẽ được nạp lại vào hàng đợi ready_queue để CPU tiến hành xử lý tiếp.

Cả 2 hàng đợi đều sử dụng hàng đợi ưu tiên, mức độ ưu tiên dựa vào chỉ số priority trong phần tử của hàng đợi.

b. **Giải thuật Priority Feedback Queue sử dụng kết hợp các giải pháp đã được đề ra trong các giải thuật khác như:**

+ First Come First Served (FCFS) - dùng khi hàng đợi ready_queue rỗng và ở thời điểm hiện tại chỉ 1 process sẵn sàng thực thi.

+ Priority Scheduling (PS) – dùng mức độ ưu tiên để quyết định process nào sẽ được CPU xử lý kế tiếp.

+ Multilevel Queue (MQ)/ Multilevel Feedback Queue Scheduling – dùng nhiều hàng đợi và CPU sẽ lấy process từ hàng đợi nào để xử lý. Các hàng đợi sẽ luân chuyển process cho nhau.

+ Round Robin (RR) – sử dụng 1 đơn vị thời gian quantum time để cho các process thực thi, hết thời gian CPU sẽ chuyển sang xử lý process tiếp theo trong hàng đợi ready_queue để không bị trì hoãn vô hạn định.

Ưu điểm của giải thuật PFQ:

+ Sử dụng đơn vị thời gian time slot tương tự như quantum time của giải thuật Round Robin tạo sự luân chuyển liên tục, thời gian xử lý bằng nhau giữa các process tránh tình trạng một process có độ ưu tiên cao liên tục chiếm thời gian sử dụng CPU làm cho các process khác bị trì hoãn vô hạn.

+ Sử dụng 2 hàng đợi, các process được luân chuyển giữa 2 hàng đợi cho đến khi hoàn tất, tăng thời gian đáp ứng cho các process (process có độ ưu tiên thấp vẫn có thể được thực thi trước các process có độ ưu tiên cao hơn sau khi các process đó đã thực hiện xong time slot của mình)

+ Tính công bằng của các process được đảm bảo, chỉ phụ thuộc vào độ ưu tiên của process và thời gian thực thi time slot. Vd khi ở thời điểm t1, process p1 đang thực thi thì các process khác không được chen ngang vào mà phải đợi khi time slot hết hạn thì process mới được thêm vào p2 ở thời điểm t2 mới được thực thi. Sau khi thực thi xong t2 mà hàng đợi ready_queue trống thì 2 process đang ở run_queue sẽ được nạp lại ở hàng chờ ready_queue và lúc này thì process nào có độ ưu tiên cao hơn sẽ được thực thi.

2.Memory Management:

Hàm get_page_table() có tác dụng kiếm bảng phân trang tại địa chỉ segment của process

Hàm get_page_table() được hiện thực như sau:

```
45  /* Search for page table table from the a segment table */
46  static struct page_table_t *get_page_table(addr_t index, // Segment level index
47                                             struct seg_table_t *seg_table)
48  { // first level table
49
50      /*
51       * TODO: Given the Segment index [index], you must go through each
52       * row of the segment table [seg_table] and check if the v_index
53       * field of the row is equal to the index
54       *
55       * */
56
57      int i;
58      for (i = 0; i < seg_table->size; i++)
59      {
60          // Enter your code here
61          if (seg_table->table[i].v_index == index)
62          {
63              return seg_table->table[i].pages;
64          }
65      }
66      return NULL;
67  }
```

Hàm translate() dùng để chuyển từ địa chỉ ảo sang địa chỉ vật lý

Hàm translate() được hiện thực như sau

```
/* Translate virtual address to physical address. If [virtual_addr] is valid,
 * return 1 and write its physical counterpart to [physical_addr].
 * Otherwise, return 0 */
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]  */
            *physical_addr = page_table->table[i].p_index << OFFSET_LEN | offset;
            return 1;
        }
    }
    return 0;
}
```

Hàm alloc_mem() được hiện thực như sau:

```
addr_t alloc_mem(uint32_t size, struct pcb_t *proc)
{
    pthread_mutex_lock(&mem_lock);
    addr_t ret_mem = 0;
    /* TODO: Allocate [size] byte in the memory for the
    * process [proc] and save the address of the first
    * byte in the allocated memory region to [ret_mem].
    * */

    uint32_t num_pages = ((size % PAGE_SIZE) == 0)
                        ? size / PAGE_SIZE
                        : size / PAGE_SIZE + 1; // Number of pages we will use
    int mem_avail = 0;                          // We could allocate new memory region or not?
    /* First we must check if the amount of free memory in
    * virtual address space and physical address space is
    * large enough to represent the amount of required
    * memory. If so, set 1 to [mem_avail].
    * Hint: check [proc] bit in each page of _mem_stat
    * to know whether this page has been used by a process.
    * For virtual memory space, check bp (break pointer).
    * */

    uint32_t cr_num_pages = 0;
    for (int i = 0; i < NUM_PAGES; i++)
    {
        if (_mem_stat[i].proc == 0)
        {
            cr_num_pages++;
        }
    }
    if (cr_num_pages >= num_pages && proc->bp + num_pages * PAGE_SIZE < NUM_PAGES * PAGE_SIZE + PAGE_SIZE)
        mem_avail = 1;
    else
        ret_mem = 0;
    if (mem_avail)
    {
        /* We could allocate new memory region to the process */
        ret_mem = proc->bp;
        proc->bp += num_pages * PAGE_SIZE;
        /* Update status of physical pages which will be allocated
         * to [proc] in _mem_stat. Tasks to do:
         *  - Update [proc], [index], and [next] field
         *  - Add entries to segment table page tables of [proc]
         *    to ensure accesses to allocated memory slot is
         *    valid. */
```

```
153         uint32_t num_pages_use = 0;
154         for (int i = 0, j = 0, k = 0; i < NUM_PAGES; i++)
155         {
156             if (_mem_stat[i].proc == 0)
157             {
158                 _mem_stat[i].proc = proc->pid;
159                 _mem_stat[i].index = j;
160                 if (j != 0)
161                     _mem_stat[k].next = i;
162                 addr_t physical_addr = i << OFFSET_LEN;
163                 addr_t first_lv = get_first_lv(ret_mem + j * PAGE_SIZE);
164                 addr_t second_lv = get_second_lv(ret_mem + j * PAGE_SIZE);
165                 int booler = 0;
166                 /*Neu da co first index trong page table*/
167                 for (int n = 0; n < proc->seg_table->size; n++)
168                 {
169                     if (proc->seg_table->table[n].v_index == first_lv)
170                     {
171                         proc->seg_table->table[n].pages->table[proc->
172                             seg_table->table[n].pages->size].v_index = second_lv;
173                         proc->seg_table->table[n].pages->table[proc->
174                             seg_table->table[n].pages->size].p_index = physical_addr >> OFFSET_LEN;
175                         proc->seg_table->table[n].pages->size++;
176                         booler = 1;
177                         break;
178                     }
178                     }
179                 }
180                 /*Neu chua co first index trong seg table tao page table moi*/
181                 if (booler == 0)
182                 {
183                     int n = proc->seg_table->size;
184                     proc->seg_table->size++;
185                     proc->seg_table->table[n].pages =
186                         (struct page_table_t *)malloc(sizeof(struct page_table_t));
187                     proc->seg_table->table[n].pages->size++;
188                     proc->seg_table->table[n].v_index = first_lv;
189                     proc->seg_table->table[n].pages->table[0].v_index = second_lv;
190                     proc->seg_table->table[n].pages->table[0].p_index = physical_addr >> OFFSET_LEN;
191                 }
192                 /*------------------------*/
193                 k = i;
194                 j++;
195                 num_pages_use++;
196                 if (num_pages_use == num_pages)
197                 {
198                     _mem_stat[k].next = -1;
199                     break;
200                 }
201             }
202         }
203     }
```

```
    printf("Cap phat cho process: %d %d trang\n",proc->pid,num_pages );
    /*Code kiem tra bang phan trang va phan doan*/
    printf("Cap phat vung nho cho process:%d\n", proc->pid);
     for (int i = 0; i < proc->seg_table->size; i++)
     {
         printf("seg table v index :%05x\n", proc->seg_table->table[i].v_index);
         for (int j = 0; j < proc->seg_table->table[i].pages->size; j++)
         {
             printf("page table v index: %05x page: %05x\n",
                 proc->seg_table->table[i].pages->table[j].v_index,
                    proc->seg_table->table[i].pages->table[j].p_index);
         }
     }
     dump();
    /*Kiem tra _mem_stat*/
     printf("=========alocccccccccccc=========\n");
     dump();
    pthread_mutex_unlock(&mem_lock);
    return ret_mem;
}

```

Hàm free_mem() được hiện thực như sau:

```
int free_mem(addr_t address, struct pcb_t *proc)
{

    /*TODO: Release memory region allocated by [proc]. The first byte of
    * this region is indicated by [address]. Task to do:
    *     - Set flag [proc] of physical page use by the memory block
    *       back to zero to indicate that it is free.
    *     - Remove unused entries in segment table and page tables of
    *       the process [proc].
    *     - Remember to use lock to protect the memory from other
    *       processes.  */
     pthread_mutex_lock(&mem_lock);
     addr_t physical_addr;
     if (translate(address, &physical_addr, proc))
     {
          int number1=0,first_num=0;
         int next = -2;
         int i = 0, j = 0;
         /*Xoa tim ra vi tri ung voi address*/
         for (; i < NUM_PAGES; i++)
         {
             if (physical_addr == i << OFFSET_LEN)
             {
                 // first_num=i;
                 break;
```

```
250             }
251         }
252         /*Ung voi vi tri vua tim duoc ta xoa nhung cai tiep theo*/
253         next = i;
254         while (next != -1)
255         {
256             number1++;
257             _mem_stat[next].proc = 0;
258             next = _mem_stat[next].next;
259             /*----------------*/
260             /*Xoa seg table va page table*/
261             addr_t first_lv = get_first_lv(address + j * PAGE_SIZE);
262             addr_t second_lv = get_second_lv(address + j * PAGE_SIZE);
263             for (int n = 0; n < proc->seg_table->size; n++)
264             {
265                 if (proc->seg_table->table[n].v_index == first_lv)
266                 {
267                     for (int m = 0; m < proc->seg_table->table[n].pages->size; m++)
268                     {
269                         if (proc->seg_table->table[n].pages->table[m].v_index == second_lv)
270                         {
271                             /*----------------*/
272                             /*Don page table lai, xoa phan tu cuoi cung*/
273                             int k = 0;
274                             for (k = m; k < proc->seg_table->table[n].pages->size - 1; k++)
275                             {
275                                 {
276                                     proc->seg_table->table[n].pages->table[k].v_index
277                                         = proc->seg_table->table[n].pages->table[k + 1].v_index;
278                                     proc->seg_table->table[n].pages->table[k].p_index
279                                     = proc->seg_table->table[n].pages->table[k + 1].p_index;
280                                 }
281                                 proc->seg_table->table[n].pages->table[k].v_index = 0;
282                                 proc->seg_table->table[n].pages->table[k].p_index = 0;
283                                 proc->seg_table->table[n].pages->size--;
284                                 /*----------------*/
285                                 break;
286                             }
287                         }
288                         if (proc->seg_table->table[n].pages->size == 0)
289                         {
290                             /* Neu page-table do rong thi xoa di */
291                             free(proc->seg_table->table[n].pages);
292                             int m = 0;
293                             for (m = n; m < proc->seg_table->size - 1; m++)
294                             {
295                                 proc->seg_table->table[m].v_index = proc->seg_table->table[m + 1].v_index;
296                                 proc->seg_table->table[m].pages = proc->seg_table->table[m + 1].pages;
297                             }
298                             proc->seg_table->table[m].v_index = 0;
```

```
298                     proc->seg_table->table[m].v_index = 0;
299                     proc->seg_table->table[m].pages = NULL;
300                     proc->seg_table->size--;
301                 }
302                 break;
303             }
304         }
305         j++;
306     }
307     printf("Giai phong %d trang nho bat dau tu trang %d\n",number1 ,first_num );
308 }
309
310 /*Code kiem tra bang phan trang va phan doan*/
311  printf("Giai phong vung nho cho process:%d\n", proc->pid);
312  for (int i = 0; i < proc->seg_table->size; i++)
313  {
314      printf("seg table v index :%05x\n", proc->seg_table->table[i].v_index);
315     for (int j = 0; j < proc->seg_table->table[i].pages->size; j++)
316      {
317          printf("page table v index: %05x page: %05x\n",
318             proc->seg_table->table[i].pages->table[j].v_index,
319                 proc->seg_table->table[i].pages->table[j].p_index);
320      }
321  }
322  dump();

323 /*Code kiem tra _mem_stat*/
324  printf("=========free========\n");
325 dump();
326 pthread_mutex_unlock(&mem_lock);
327 return 0;
328 }
```

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

Ưu điểm giải thuật:

- Tiết kiệm bộ nhớ, sử dụng bộ nhớ hiệu quả.
- Mang các ưu điểm của giải thuật phân trang:
  + Đơn giản việc cấp phát vùng nhớ.
  + Khắc phục được phân mảnh ngoại.
- Giải quyết vấn đề phân mảnh ngoại của giải thuật phân đoạn bằng cách phân trang trong mỗi đoạn.

Nhược điểm giải thuật:

- Phân mảnh nội của giải thuật phân trang vẫn còn.

Memory: Show the status of RAM after each memory allocation and deallocation function call.

```
root@AnhLe-PC:/mnt/g/He dieu hanh/TH/btl2/Assignment 2/source_code# make mem
gcc -Iinclude  -Wall -c -g src/mem.c -o obj/mem.o
gcc -Iinclude  -Wall -g obj/paging.o obj/mem.o obj/cpu.o obj/loader.o -o mem -lpthread
root@AnhLe-PC:/mnt/g/He dieu hanh/TH/btl2/Assignment 2/source_code# make test_mem
------ MEMORY MANAGEMENT TEST 0 -----------------------------------
./mem input/proc/m0
Cap phat cho process: 1 14 trang
Cap phat vung nho cho process:1
seg table v index :00000
page table v index: 00001 page: 00000
page table v index: 00002 page: 00001
page table v index: 00003 page: 00002
page table v index: 00004 page: 00003
page table v index: 00005 page: 00004
page table v index: 00006 page: 00005
page table v index: 00007 page: 00006
page table v index: 00008 page: 00007
page table v index: 00009 page: 00008
page table v index: 0000a page: 00009
page table v index: 0000b page: 0000a
page table v index: 0000c page: 0000b
page table v index: 0000d page: 0000c
page table v index: 0000e page: 0000d

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)
=========alloccccccccccc=========
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)
```

```
Cap phat cho process: 1 2 trang
Cap phat vung nho cho process:1
seg table v index :00000
page table v index: 00001 page: 00000
page table v index: 00002 page: 00001
page table v index: 00003 page: 00002
page table v index: 00004 page: 00003
page table v index: 00005 page: 00004
page table v index: 00006 page: 00005
page table v index: 00007 page: 00006
page table v index: 00008 page: 00007
page table v index: 00009 page: 00008
page table v index: 0000a page: 00009
page table v index: 0000b page: 0000a
page table v index: 0000c page: 0000b
page table v index: 0000d page: 0000c
page table v index: 0000e page: 0000d
page table v index: 0000f page: 0000e
page table v index: 00010 page: 0000f
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)
=========alloccccccccccc=========
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)
Giai phong 14 trang nho bat dau tu trang 0
Giai phong vung nho cho process:1
seg table v index :00000
page table v index: 0000f page: 0000e
page table v index: 00010 page: 0000f
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
```

```
=========free========
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
Cap phat cho process: 1 2 trang
Cap phat vung nho cho process:1
seg table v index :00000
page table v index: 0000f page: 0000e
page table v index: 00010 page: 0000f
page table v index: 00011 page: 00000
page table v index: 00012 page: 00001
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)
=========alloccccccccccc=========
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)
Cap phat cho process: 1 5 trang
Cap phat vung nho cho process:1
seg table v index :00000
page table v index: 0000f page: 0000e
page table v index: 00010 page: 0000f
page table v index: 00011 page: 00000
page table v index: 00012 page: 00001
page table v index: 00013 page: 00002
page table v index: 00014 page: 00003
page table v index: 00015 page: 00004
page table v index: 00016 page: 00005
page table v index: 00017 page: 00006

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)
=========alloccccccccccc=========
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
------ MEMORY MANAGEMENT TEST 1 --------------------
./mem input/proc/m1
Cap phat cho process: 1 14 trang
Cap phat vung nho cho process:1
seg table v index :00000
page table v index: 00001 page: 00000
page table v index: 00002 page: 00001
page table v index: 00003 page: 00002
page table v index: 00004 page: 00003
page table v index: 00005 page: 00004
page table v index: 00006 page: 00005
page table v index: 00007 page: 00006
page table v index: 00008 page: 00007
page table v index: 00009 page: 00008
page table v index: 0000a page: 00009
page table v index: 0000b page: 0000a
page table v index: 0000c page: 0000b
page table v index: 0000d page: 0000c
page table v index: 0000e page: 0000d
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)
=========alloccccccccccc=========
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)
Cap phat cho process: 1 2 trang
Cap phat vung nho cho process:1
seg table v index :00000
page table v index: 00001 page: 00000
page table v index: 00002 page: 00001
page table v index: 00003 page: 00002
page table v index: 00004 page: 00003
page table v index: 00005 page: 00004
page table v index: 00006 page: 00005
```

```
page table v index: 00007 page: 00006
page table v index: 00008 page: 00007
page table v index: 00009 page: 00008
page table v index: 0000a page: 00009
page table v index: 0000b page: 0000a
page table v index: 0000c page: 0000b
page table v index: 0000d page: 0000c
page table v index: 0000e page: 0000d
page table v index: 0000f page: 0000e
page table v index: 00010 page: 0000f
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)
=========allocccccccccc=========
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)
Giai phong 14 trang nho bat dau tu trang 0
Giai phong vung nho cho process:1
seg table v index :00000
page table v index: 0000f page: 0000e
page table v index: 00010 page: 0000f
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
=========free=========
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
Cap phat cho process: 1 2 trang
Cap phat vung nho cho process:1
seg table v index :00000
page table v index: 0000f page: 0000e
page table v index: 00010 page: 0000f
page table v index: 00011 page: 00000
page table v index: 00012 page: 00001
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)
=========allocccccccccc=========
```

```
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)
Cap phat cho process: 1 5 trang
Cap phat vung nho cho process:1
seg table v index :00000
page table v index: 0000f page: 0000e
page table v index: 00010 page: 0000f
page table v index: 00011 page: 00000
page table v index: 00012 page: 00001
page table v index: 00013 page: 00002
page table v index: 00014 page: 00003
page table v index: 00015 page: 00004
page table v index: 00016 page: 00005
page table v index: 00017 page: 00006
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)
=========alloccccccccccc=========
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)
```

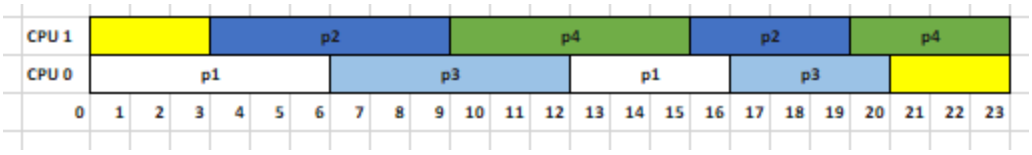
```
004: 01000-013ff - PID: 01 (idx 002, nxt: 005)
005: 01400-017ff - PID: 01 (idx 003, nxt: 006)
006: 01800-01bff - PID: 01 (idx 004, nxt: -01)
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
Giai phong 2 trang nho bat dau tu trang 0
Giai phong vung nho cho process:1
seg table v index :00000
page table v index: 0000f page: 0000e
page table v index: 00010 page: 0000f
page table v index: 00013 page: 00002
page table v index: 00014 page: 00003
page table v index: 00015 page: 00004
page table v index: 00016 page: 00005
page table v index: 00017 page: 00006
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)
=========free=========
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)
```

```
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
Giai phong 5 trang nho bat dau tu trang 0
Giai phong vung nho cho process:1
seg table v index :00000
page table v index: 0000f page: 0000e
page table v index: 00010 page: 0000f
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
=========free========
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
Giai phong 2 trang nho bat dau tu trang 0
Giai phong vung nho cho process:1
=========free========
NOTE: Read file output/m1 to verify your result (your implementation should print nothing)
root@AnhLe-PC:/mnt/g/He dieu hanh/TH/btl2/Assignment 2/source_code#
```

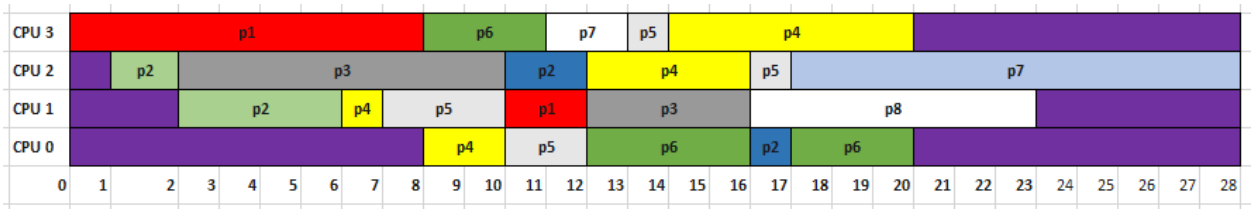
3. Put it together

Ta có sơ đồ gant như sau

Os test 0



Os test 1



Phần này ta thực hiện câu lệnh make all để thực hiện đồng bộ hóa (synchonization) khi thực hiện thao tac định thời (scheduling) và xử lý bộ nhớ (vitual memory).

Các CPU chia sẻ công việc thực hiện để hoàn tất công việc. Để tránh hiện tiên race condition, ta sử dụng cơ chế mutex lock để bảo vệ tài nguyên được chia sẻ.
Các đoạn code trong hàm alloc_mem() và free_mem() , put_proc(), add_proc sẽ được dặt trong cặp lệnh pthread_mutex_lock() và pthread_mutex_unlock() nhằm đảm bảo rằng tại mọi thời điểm trên 1 CPU chỉ có 1 tiến trình duy nhất thực thi một trong các hàm trên.

-=HẾT=-