HCMC University Of Technology
Faculty of Computer Science & Engineering

Course: Operating Systems (CO2017) - HK211

Assignment

Simple Operating System

Lecturer: Le Thanh Van

Class: CC01

Group members	ID
Nguyen Tran Anh Quan	1952418
Tran Minh Trung	1953052
Doan Viet Tu	1952521

Ho Chi Minh City, December 2021

Contents

Page

I. SCHEDULER.. 1

1.1. Question... 1
1.2. Implementation.. 2

II. MEMORY MANAGEMENT.. 9

2.1. Question... 9
2.2. Implementation.. 10

III. PUT IT ALL TOGETHER 23

I. SCHEDULER

1.1. Question:

What is the advantage of using priority feedback queue in comparison with other scheduling algorithms you have learned such as FIFO, Round Robin? Explain clearly your answer .

Answer:

Priority feedback queue algorithm is based on the concept of the multilevel feedback queue and it can utilize several algorithms such as priority scheduling and round-robin .

This algorithm uses 2 priority queues : *ready_queue* and *run_queue*

- *ready_queue*: This queue contains processes that are ready to be executed; when CPU finishes the previous job and is ready to start a new one, it will pick a process from this queue.
- *run_queue*: This queue contains all the processes that could not finish after a time slice . These processes will wait until the *ready_queue* is empty again , they will be push back to *ready_queue* to finish their jobs .

→ **Advantages:**

- Determine the priority of a given process by using feedback , so the processes are executed consecutively.
- Executing processes using round-robin algorithm : each process can get an equal share of CPU resources, no process perpetually lacks necessary resources and avoid starvation.
- Using multilevel queue: a process can be switched between 2 queues until it is finished each process's duration of response is increased.

2. Implementation

1. Scheduler:

```
struct pcb_t * get_proc(void) {
        struct pcb_t * proc = NULL;
        /*TODO: get a process from [ready_queue]. If ready queue
         * is empty, push all processes in [run_queue] back to
         * [ready_queue] and return the highest priority one.
         * Remember to use lock to protect the queue.
         * */
        pthread_mutex_lock(&queue_lock);
        if (empty(&ready_queue)){
                while (!empty(&run_queue)){
                        enqueue(&ready_queue, dequeue(&run_queue));
                }
        }
        if (!empty(&ready_queue)){
                proc = dequeue(&ready_queue);
        }
        pthread_mutex_unlock(&queue_lock);
        return proc;
}
```

1.2.2. Priority queue:

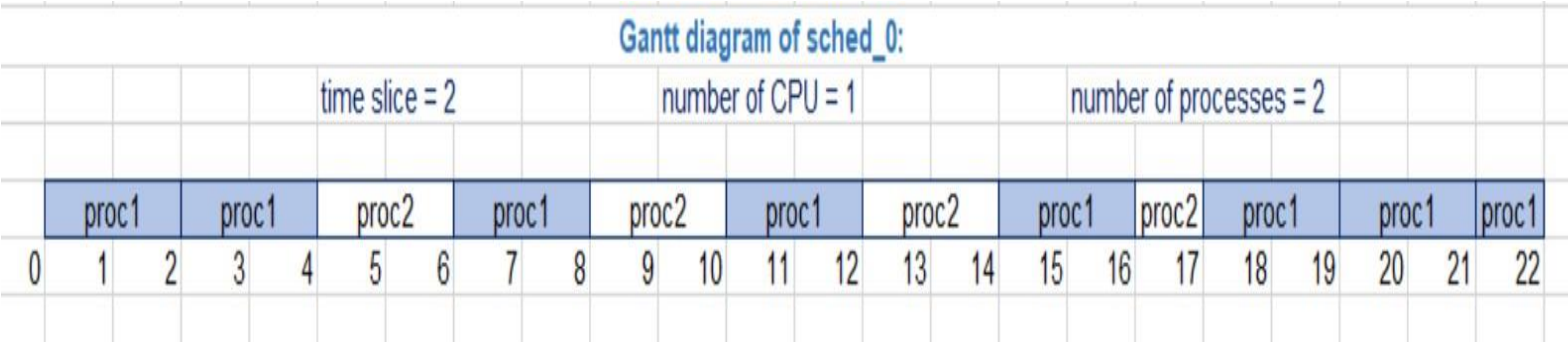
```
void enqueue(struct queue_t * q, struct pcb_t * proc) {
        /* TODO: put a new process to queue [q] */
        if (q->size >= MAX_QUEUE_SIZE) return;
        q->proc[q->size++] = proc;
}

struct pcb_t * dequeue(struct queue_t * q) {
        /* TODO: return a pcb whose prioprity is the highest
         * in the queue [q] and remember to remove it from q
         * */
        if (q->size >= 0){
                int k = 0;
                for (int n = 1; n < q->size; n++){
                        if (q->proc[n]->priority < q->proc[k]->priority)
                                k = n;
                }
                struct pcb_t * temp = q->proc[k];
                for (int m = k + 1; m < q->size; m++){
                        q->proc[m-1] = q->proc[m];
                }
                q->size--;
                return temp;
        } else
                return NULL;
}
```

Using command *make sched* and then *make test_sched* , we have the result of *sched_0* as:

```
vtudn@vtudn-VirtualBox:~/Desktop/os_as/source_code$ make sched
gcc -Iinclude  -Wall -g obj/cpu.o obj/loader.o obj/mem.o obj/queue.o obj/os.o obj/sched.o obj/timer.o -o os -lpthrea
vtudn@vtudn-VirtualBox:~/Desktop/os_as/source_code$ make test_sched
------ SCHEDULING TEST 0 ------------------------------------------
./os sched_0
Time slot   0
        Loaded a process at input/proc/s0, PID: 1
        CPU 0: Dispatched process  1
Time slot   1
Time slot   2
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot   3
Time slot   4
        Loaded a process at input/proc/s1, PID: 2
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   5
Time slot   6
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  1
Time slot   7
Time slot   8
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   9
Time slot  10
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  1
Time slot  11
Time slot  12
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot  13
Time slot  14
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  1
Time slot  15
Time slot  16
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot  17
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  1
Time slot  18
```

```
        CPU 0: Dispatched process  1
Time slot  18
Time slot  19
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  20
Time slot  21
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  22
        CPU 0: Processed  1 has finished
        CPU 0 stopped
```

Using Gantt diagram to illustrates the process of *sched_0* we have:



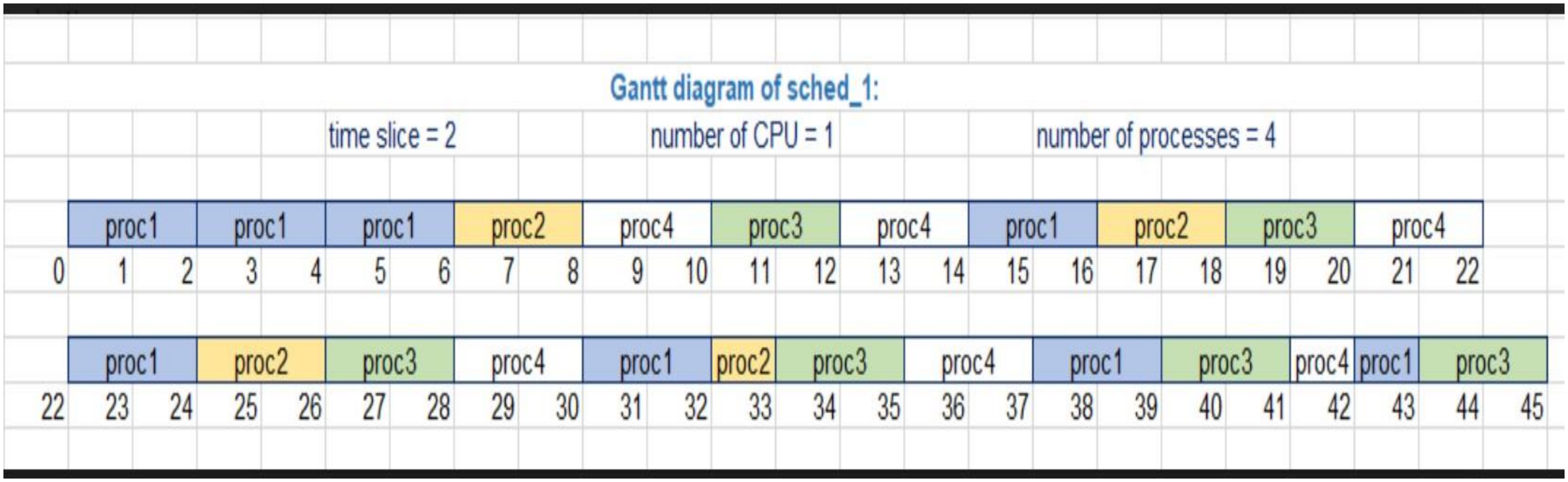
Using command *make sched* and then *test_sched* , we have the result of *sched_1* as:

```
MEMORY CONTENT:
NOTE: Read file output/sched_0 to verify your result
------ SCHEDULING TEST 1 ---------------------------------
./os sched_1
Time slot   0
        Loaded a process at input/proc/s0, PID: 1
        CPU 0: Dispatched process  1
Time slot   1
Time slot   2
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot   3
Time slot   4
        Loaded a process at input/proc/s1, PID: 2
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   5
Time slot   6
        Loaded a process at input/proc/s2, PID: 3
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot   7
        Loaded a process at input/proc/s3, PID: 4
Time slot   8
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot   9
Time slot  10
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
Time slot  11
Time slot  12
        CPU 0: Put process  4 to run queue
```

```
Time slot  12
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  1
Time slot  13
Time slot  14
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot  15
Time slot  16
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  17
Time slot  18
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  19
Time slot  20
        CPU 0: Put process  4 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  2
Time slot  23
Time slot  24
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  25
Time slot  26
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  27
Time slot  28
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  1
Time slot  29
Time slot  30
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot  31
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  3
```

```
Time slot  30
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot  31
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  3
Time slot  32
Time slot  33
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  34
Time slot  35
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  1
Time slot  36
Time slot  37
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  3
Time slot  38
Time slot  39
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  40
        CPU 0: Processed  4 has finished
        CPU 0: Dispatched process  1
Time slot  41
Time slot  42
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  3
Time slot  43
Time slot  44
        CPU 0: Processed  3 has finished
        CPU 0: Dispatched process  1
Time slot  45
        CPU 0: Processed  1 has finished
        CPU 0 stopped

MEMORY CONTENT:
NOTE: Read file output/sched_1 to verify your result
```

Using Gantt diagram to illustrates the process of *sched_1* we have:



II. MEMORY MANAGEMENT:

2.1. Question: In which system is segmentation with paging used (give an example of at least one system)? Explain clearly the advantage and disadvantage of segmentation with paging.

Answer:

Some modern computers use segmentation with paging . Main memory is divided into variably-sized segments, which are then divided into smaller fixed-size pages on disk. Each segment contains a page table, and there are multiple page tables per process. Each of the tables contains information on every segment page, while the segment table has information about every segment. Segment tables are mapped to page tables, and page tables are mapped to individual pages within a segment.

The advantage and disadvantage of segmentation with paging:

- *Advantages*:
  + Reduce memory usage and simplify memory allocation .
  + Avoid external fragmentation .
  + Page table size is limited by the segment size .
  + Segment table has only one entry corresponding to one actual segment.
- *Disadvantages*:
  + It may cause internal fragmentation.
  + The complexity level will be much higher than other types of paging.
  + Page Tables need to be contiguously stored in the memory.

2. Implementation

We implement some functions in *mem.c*:

1. Find the page table given a segment index of a process.

```
/* 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++) {
             if (seg_table->table[i].v_index == index)
                return seg_table->table[i].pages;
        }
        return NULL;

}
```

2.2.2. Uses get_page_table() function to translate a virtual address to physical address.

```
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;
}
```

2.2.3. Memory allocation.

```
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) ?
                size / PAGE_SIZE + 1 : size / PAGE_SIZE; // 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).
         * */
        int count_num_pages = 0;
        for (int i = 0; i < NUM_PAGES; i++)
                if (_mem_stat[i].proc == 0)
                  count_num_pages++;
        if (count_num_pages >= num_pages && NUM_PAGES * PAGE_SIZE - proc->bp >= num_pages * PAGE_SIZE)
                mem_avail = 1;

        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
```

```
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. */
uint32_t num_pages_use = 0;
int n = 0, id = 0, prev = -1;
while (num_pages_use != num_pages) {
        if (_mem_stat[n].proc == 0) {
          _mem_stat[n].proc = proc->pid;
          _mem_stat[n].index = id;
          _mem_stat[n].next = -1;
             if (prev != -1) _mem_stat[prev].next = n;
                prev = n;
                addr_t first_lv = get_first_lv(ret_mem + id * PAGE_SIZE);
                addr_t second_lv = get_second_lv(ret_mem + id * PAGE_SIZE);
                int exist = 0;
                for (int j = 0; j < proc->seg_table->size; j++)
                        if (proc->seg_table->table[n].v_index == first_lv){
                          proc->seg_table->table[j].pages->table[proc->seg_table->table[j].pages->size].v_index = second_lv;
                          proc->seg_table->table[j].pages->table[proc->seg_table->table[j].pages->size].p_index = n;
                          proc->seg_table->table[j].pages->size++;
                          exist = 1;
                                break;
                        }

                if (!exist) {
                 int k = proc->seg_table->size++;
                        proc->seg_table->table[k].pages = (struct page_table_t *)malloc(sizeof(struct page_table_t));
                        proc->seg_table->table[k].pages->size++;
                        proc->seg_table->table[k].v_index = first_lv;
                        proc->seg_table->table[k].pages->table[0].v_index = second_lv;
                        proc->seg_table->table[k].pages->table[0].p_index = n;
                }
                num_pages_use++;
                id++;
        }
        n++;
```

```
                "'',
            }
        }

        printf("====================Allocation====================\n");
printf("Process %d. Size: %d. New break point: 0x%05x. Return memory: 0x%05x\n",proc->pid, size, proc->bp, ret_mem);
dump();
        pthread_mutex_unlock(&mem_lock);
        return ret_mem;
}

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)) {
                pthread_mutex_unlock(&mem_lock);
                return 1;
        }

        int num_pages = 0;
        for (int i = physical_addr>> OFFSET_LEN; i != -1; i = _mem_stat[i].next) {
             _mem_stat[i].proc = 0;
              proc->bp -= PAGE_SIZE;
              num_pages++;
}

        for (int i = 0; i < num_pages; i++) {
                addr_t ad = address + i*PAGE_SIZE;
                addr_t first_lv = get_first_lv(ad);
                addr_t second_lv = get_second_lv(ad);
                for (int x = 0; x < proc->seg_table->size; x++)
                        if (proc->seg_table->table[x].v_index == first_lv) {
                                for (int z = 0; z < proc->seg_table->table[x].pages->size; z++)
                                  if (proc->seg_table->table[x].pages->table[z].v_index == second_lv) {
                                                proc->seg_table->table[x].pages->table[z] = proc->seg_table->table[x].pages->table[--proc->seg_table->table[x].pages->size];
                                        break;
                        }

                                if (proc->seg_table->table[x].pages->size == 0) {
                                  free(proc->seg_table->table[x].pages);
```

```
		for (int x = 0; x < proc->seg_table->size; x++)
			if (proc->seg_table->table[x].v_index == first_lv) {
				for (int z = 0; z < proc->seg_table->table[x].pages->size; z++)
				  if (proc->seg_table->table[x].pages->table[z].v_index == second_lv) {
						proc->seg_table->table[x].pages->table[z] = proc->seg_table->table[x].pages->table[--proc->seg_table->table[x].pages->size];
					break;
			}

				if (proc->seg_table->table[x].pages->size == 0) {
				  free(proc->seg_table->table[x].pages);
				  proc->seg_table->table[x] = proc->seg_table->table[--proc->seg_table->size];
		}
			break;
			}
	}

	printf("==================Deallocation===================\n");
	printf("Process %d. Address: %d\n",proc->pid, address);
	dump();
	pthread_mutex_unlock(&mem_lock);
	return 0;
}

int read_mem(addr_t address, struct pcb_t * proc, BYTE * data) {
	addr_t physical_addr;
	if (translate(address, &physical_addr, proc)) {
		pthread_mutex_lock(&ram_lock);
		*data = _ram[physical_addr];
		pthread_mutex_unlock(&ram_lock);
		return 0;
	}else{
		return 1;
	}
}

int write_mem(addr_t address, struct pcb_t * proc, BYTE data) {
	addr_t physical_addr;
	if (translate(address, &physical_addr, proc)) {
		pthread_mutex_lock(&ram_lock);
		_ram[physical_addr] = data;
		pthread_mutex_unlock(&ram_lock);
		return 0;
	}else{
		return 1;
	}
}
```

Implementation:

About the memory management in this assignment, we are going to implement the file mem.c:

In get_page_table() function, go through each row of segment table and check if v_index of this row is equal to index. If it equals, return the page table located in this row.

In translate() function, we get offset, first and second layer index from given virtual address. We also get page table that include *p_index* too. If there is no page, return 0 to report that can not translate. Otherwise, continue to determine *p_index* by the way that go through each row of this page table and check if *v_index* is equal to second layer index. If it equals, translate by concatenating offset of virtual address to *p_index* and then assign to *physical_addr*. Else , return 0.

Implementation:

In alloc_mem() function, firstly, we count the free pages by *count_num_pages* after checking these pages is have no process in it. Then we use it to check the virtual and physical space that it has enough space or not? If it has enough, the *mem_avail* will be set as 1 and we can allocate memory region to the process.

Then we go through each page and check if this page is used. If it is used already, pass this page and continue to next one. Otherwise, we can determine that can use this page. The last page will have the next index equals -1.

After that we will get the virtual address of each page (include first layer index and second layer index). Now it has 2 cases, in case 1, the first layer index is already exist and I just get the page table. Else, we have to create new one.

Implementation:

In the free_mem() function, to deallocate the memory region that we have allocated :

Firstly, we will find the physical page in memory and from that go to clear the physical memory. And I also count how many that I have clear too.

After that, base on the number of physical pages I have deleted , we are going to find the virtual address to get the virtual segment and virtual pages. Then we will update the page table. While the page table is empty is the time we will free it.

Using command *make test_mem* , we have the result of *memory management test 0* as:

```
./mem input/proc/m0
====================Allocation====================
Process 1. Size: 13535. New break point: 0x03c00. Return memory: 0x00400
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====================
Process 1. Size: 1568. New break point: 0x04400. Return memory: 0x03c00
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)
```

```
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====================
Process 1. Address: 1024
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
====================Allocation====================
Process 1. Size: 1386. New break point: 0x01400. Return memory: 0x00c00
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====================
Process 1. Size: 4564. New break point: 0x02800. Return memory: 0x01400
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)
```

```
====================Allocation====================
Process 1. Size: 1386. New break point: 0x01400. Return memory: 0x00c00
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====================
Process 1. Size: 4564. New break point: 0x02800. Return memory: 0x01400
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 0 ------------------------------------
./mem input/proc/m0
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
NOTE: Read file output/m1 to verify your result (your implementation should pri
nt nothing)
quankk123456@quankk123456-VirtualBox:~/OS/source_code$ make
```

Using command *make test_mem* , we have the result of *memory management test 1* as:

```
------ MEMORY MANAGEMENT TEST 1 -----------------------------------
./mem input/proc/m1
=====================Allocation=====================
Process 1. Size: 13535. New break point: 0x03c00. Return memory: 0x00400
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=====================
Process 1. Size: 1568. New break point: 0x04400. Return memory: 0x03c00
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)
```

```
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=====================
Process 1. Address: 1024
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
=====================Allocation=====================
Process 1. Size: 1386. New break point: 0x01400. Return memory: 0x00c00
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=====================
Process 1. Size: 4564. New break point: 0x02800. Return memory: 0x01400
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)
```

```
====================Allocation====================
Process 1. Size: 4564. New break point: 0x02800. Return memory: 0x01400
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====================
Process 1. Address: 3072
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====================
Process 1. Address: 5120
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
==================Deallocation====================
Process 1. Address: 15360
NOTE: Read file output/m1 to verify your result (your implementation should pri
nt nothing)
```

III. PUT IT ALL TOGETHER

Using command *./os* , we have *os_0* as:

```
./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  3
Time slot   8
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  4
Time slot   9
Time slot  10
```

```
Time slot  10
Time slot  11
Time slot  12
Time slot  13
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  1
Time slot  14
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  15
Time slot  16
Time slot  17
        CPU 1: Processed  1 has finished
        CPU 1: Dispatched process  3
Time slot  18
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  4
Time slot  19
Time slot  20
Time slot  21
        CPU 1: Processed  3 has finished
        CPU 1 stopped
```

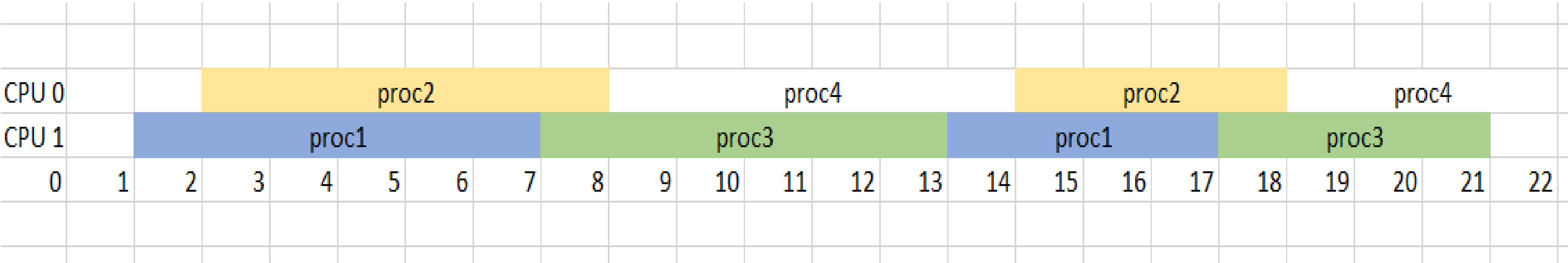
```
        CPU 1 stopped
Time slot  22
        CPU 0: Processed  4 has finished
        CPU 0 stopped
```

Memory content of *os_0:*

```
MEMORY CONTENT:
000: 00000-003ff - PID: 01 (idx 000, nxt: -01)
001: 00400-007ff - PID: 03 (idx 000, nxt: 002)
002: 00800-00bff - PID: 03 (idx 001, nxt: 003)
003: 00c00-00fff - PID: 03 (idx 002, nxt: 004)
004: 01000-013ff - PID: 03 (idx 003, nxt: -01)
005: 01400-017ff - PID: 04 (idx 000, nxt: 006)
        01414: 64
006: 01800-01bff - PID: 04 (idx 001, nxt: 012)
007: 01c00-01fff - PID: 02 (idx 000, nxt: 008)
008: 02000-023ff - PID: 02 (idx 001, nxt: 009)
009: 02400-027ff - PID: 02 (idx 002, nxt: 010)
        025e7: 0a
010: 02800-02bff - PID: 02 (idx 003, nxt: 011)
011: 02c00-02fff - PID: 02 (idx 004, nxt: -01)
012: 03000-033ff - PID: 04 (idx 002, nxt: 013)
013: 03400-037ff - PID: 04 (idx 003, nxt: -01)
015: 03c00-03fff - PID: 03 (idx 000, nxt: 016)
016: 04000-043ff - PID: 03 (idx 001, nxt: 017)
017: 04400-047ff - PID: 03 (idx 002, nxt: 018)
        045e7: 0a
018: 04800-04bff - PID: 03 (idx 003, nxt: 019)
```

```
018: 04800-04bff - PID: 03 (idx 003, nxt: 019)
019: 04c00-04fff - PID: 03 (idx 004, nxt: -01)
020: 05000-053ff - PID: 04 (idx 000, nxt: 021)
021: 05400-057ff - PID: 04 (idx 001, nxt: 022)
022: 05800-05bff - PID: 04 (idx 002, nxt: 023)
023: 05c00-05fff - PID: 04 (idx 003, nxt: -01)
024: 06000-063ff - PID: 02 (idx 000, nxt: 025)
025: 06400-067ff - PID: 02 (idx 001, nxt: 026)
026: 06800-06bff - PID: 02 (idx 002, nxt: 027)
027: 06c00-06fff - PID: 02 (idx 003, nxt: -01)
057: 0e400-0e7ff - PID: 04 (idx 000, nxt: 058)
058: 0e800-0ebff - PID: 04 (idx 001, nxt: 059)
059: 0ec00-0efff - PID: 04 (idx 002, nxt: 060)
        0ede7: 0a
060: 0f000-0f3ff - PID: 04 (idx 003, nxt: 061)
061: 0f400-0f7ff - PID: 04 (idx 004, nxt: -01)
062: 0f800-0fbff - PID: 03 (idx 000, nxt: 063)
063: 0fc00-0ffff - PID: 03 (idx 001, nxt: 064)
064: 10000-103ff - PID: 03 (idx 002, nxt: 065)
065: 10400-107ff - PID: 03 (idx 003, nxt: -01)
NOTE: Read file output/os_0 to verify your result
```

Using Gantt diagram to illustrates the process of os_*0* we have:



Using command *./os* , we have *os_1* as:

```
./os os_1
Time slot   0
Time slot   1
        Loaded a process at input/proc/p0, PID: 1
        CPU 0: Dispatched process  1
Time slot   2
        Loaded a process at input/proc/s3, PID: 2
        CPU 2: Dispatched process  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/m1, PID: 3
        CPU 2: Put process  2 to run queue
        CPU 1: Dispatched process  3
        CPU 2: Dispatched process  2
Time slot   5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot   6
        Loaded a process at input/proc/s2, PID: 4
        CPU 2: Put process  2 to run queue
        CPU 1: Put process  3 to run queue
```

```
Time slot   7
        CPU 3: Dispatched process  2
        Loaded a process at input/proc/m0, PID: 5
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  5
Time slot   8
        CPU 2: Put process  4 to run queue
        CPU 1: Put process  3 to run queue
        CPU 1: Dispatched process  4
        CPU 2: Dispatched process  1
Time slot   9
        CPU 3: Put process  2 to run queue
        CPU 3: Dispatched process  3
        Loaded a process at input/proc/p1, PID: 6
        CPU 0: Put process  5 to run queue
        CPU 0: Dispatched process  6
Time slot  10
        CPU 1: Put process  4 to run queue
        CPU 2: Put process  1 to run queue
        CPU 2: Dispatched process  5
        CPU 1: Dispatched process  2
```

```
        CPU 1: Dispatched process  2
Time slot  11
        CPU 3: Put process  3 to run queue
        CPU 3: Dispatched process  1
        Loaded a process at input/proc/s0, PID: 7
        CPU 0: Put process  6 to run queue
        CPU 0: Dispatched process  7
Time slot  12
        CPU 1: Put process  2 to run queue
        CPU 1: Dispatched process  4
        CPU 2: Put process  5 to run queue
        CPU 2: Dispatched process  3
Time slot  13
        CPU 3: Processed  1 has finished
        CPU 3: Dispatched process  6
        CPU 0: Put process  7 to run queue
        CPU 0: Dispatched process  5
Time slot  14
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  2
        CPU 2: Processed  3 has finished
        CPU 2: Dispatched process  7
Time slot  15
```

```
        Loaded a process at input/proc/s1, PID: 8
Time slot  16
        CPU 1: Put process  2 to run queue
        CPU 1: Dispatched process  5
        CPU 2: Put process  7 to run queue
        CPU 2: Dispatched process  8
Time slot  17
        CPU 3: Put process  4 to run queue
        CPU 3: Dispatched process  2
        CPU 1: Processed  5 has finished
        CPU 1: Dispatched process  7
        CPU 0: Put process  6 to run queue
        CPU 0: Dispatched process  4
Time slot  18
        CPU 3: Processed  2 has finished
        CPU 3: Dispatched process  6
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
Time slot  19
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  4
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  20
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
        CPU 3: Put process  6 to run queue
```

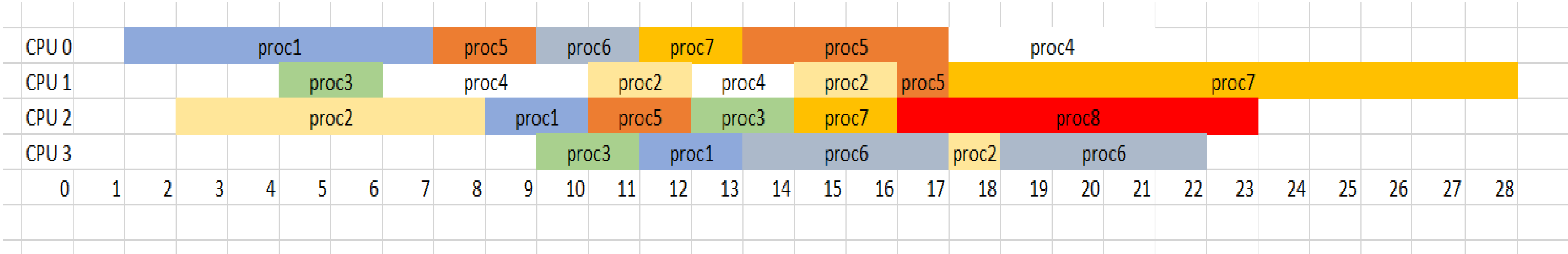
```
Time slot  20
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
        CPU 3: Put process  6 to run queue
        CPU 3: Dispatched process  6
Time slot  21
        CPU 0: Processed  4 has finished
        CPU 0 stopped
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  22
        CPU 3: Processed  6 has finished
        CPU 3 stopped
        CPU 2: Put process  8 to run queue
        CPU 2: Dispatched process  8
Time slot  23
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
        CPU 2: Processed  8 has finished
        CPU 2 stopped
Time slot  24
Time slot  25
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  26
Time slot  27
```

```
Time slot  26
Time slot  27
        CPU 1: Put process  7 to run queue
        CPU 1: Dispatched process  7
Time slot  28
        CPU 1: Processed  7 has finished
        CPU 1 stopped
```

Memory content of *os_1:*

```
MEMORY CONTENT:
000: 00000-003ff - PID: 06 (idx 000, nxt: 001)
001: 00400-007ff - PID: 06 (idx 001, nxt: 031)
006: 01800-01bff - PID: 06 (idx 000, nxt: 009)
007: 01c00-01fff - PID: 05 (idx 000, nxt: 008)
        01fe8: 15
008: 02000-023ff - PID: 05 (idx 001, nxt: -01)
009: 02400-027ff - PID: 06 (idx 001, nxt: 010)
010: 02800-02bff - PID: 06 (idx 002, nxt: 011)
011: 02c00-02fff - PID: 06 (idx 003, nxt: -01)
019: 04c00-04fff - PID: 05 (idx 000, nxt: 020)
020: 05000-053ff - PID: 05 (idx 001, nxt: 049)
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 002, nxt: 032)
        07de7: 0a
032: 08000-083ff - PID: 06 (idx 003, nxt: 033)
033: 08400-087ff - PID: 06 (idx 004, nxt: -01)
049: 0c400-0c7ff - PID: 05 (idx 002, nxt: 050)
050: 0c800-0cbff - PID: 05 (idx 003, nxt: 051)
051: 0cc00-0cfff - PID: 05 (idx 004, nxt: -01)
NOTE: Read file output/os_1 to verify your result
```

Using Gantt diagram to illustrates the process of os_*1* we have:



Question: What will be happen if the synchronization is not handled in your system? Illustrate the problem by example if you have any.

Answer:

When the synchronization is not handled in the system, there will occur some issues including a race condition. The race condition means the results depend on the timing execution of the code. Everytime we run the command, we may get a different result, which says that the result is indeterminate and we do not expect that.

The reason could be multiple threads running concurrently, but as we want some nice deterministic computation, which exactly is what the computers are made to do, we should use *count_mutex* to lock and unlock the critical sections, so that user can control the timing execution.

An example of lacking synchronization:
- In this assignment, if the synchronization is not used carefully then the time slice of the proccesses displayed on the console will change across runs.
- The 2 pictures below show the memory content of *os_0*, with the right code being sync-removed.
- It is clear that not only the time slices on the right console are different, but are also the order of page tables. This explains the situation of race condition.

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

```
MEMORY CONTENT:
000: 00000-003ff - PID: 01 (idx 000, nxt: -01)
001: 00400-007ff - PID: 03 (idx 000, nxt: 002)
002: 00800-00bff - PID: 03 (idx 001, nxt: 003)
003: 00c00-00fff - PID: 03 (idx 002, nxt: 004)
004: 01000-013ff - PID: 03 (idx 003, nxt: -01)
005: 01400-017ff - PID: 04 (idx 000, nxt: 006)
        01414: 64
006: 01800-01bff - PID: 04 (idx 001, nxt: 012)
007: 01c00-01fff - PID: 02 (idx 000, nxt: 008)
008: 02000-023ff - PID: 02 (idx 001, nxt: 009)
009: 02400-027ff - PID: 02 (idx 002, nxt: 010)
        025e7: 0a
010: 02800-02bff - PID: 02 (idx 003, nxt: 011)
011: 02c00-02fff - PID: 02 (idx 004, nxt: -01)
012: 03000-033ff - PID: 04 (idx 002, nxt: 013)
013: 03400-037ff - PID: 04 (idx 003, nxt: -01)
015: 03c00-03fff - PID: 03 (idx 000, nxt: 016)
016: 04000-043ff - PID: 03 (idx 001, nxt: 017)
017: 04400-047ff - PID: 03 (idx 002, nxt: 018)
        045e7: 0a
018: 04800-04bff - PID: 03 (idx 003, nxt: 019)
```