HO CHI MINH UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTER SCIENCE & ENGINEERING

ASSIGNMENT

OPERATING SYSTEM

Lecturer: Lê Thanh Vân

Class: CC01

Student's name	Student ID
Trần Đinh Anh Hùng	2053066
Hoàng Đỗ Phương Nguyên	1952360
Ngô Chấn Phong	2053321

Ho Chi Minh City – 2022

1. Scheduler

1.1. Question

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

Answer:

- Easy to use.
- Processes with higher priority execute first which saves time.
- The importance of each process is precisely defined.
- Using time slot, creating equity of execution time between processes, avoiding CPU usage, indefinitely delay.
- Able to quickly access the highest priority item with the time complexity of just O(1).
- It is more flexible in scheduling process with the usage of 2 queues: ready_queue and run_queue.
- Short-processes will be quickly completed, giving execution time to other processes.
- A good algorithm for applications with fluctuating time and resource requirements.

1.2. Result

1.2.1. Run `sched_0`

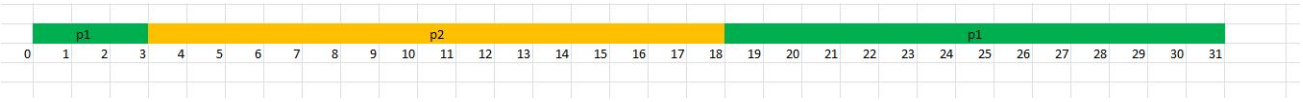


Figure 1. Gantt CPU executes processes - test sched 0

```
phong@MSI: ~/OS-ASSIGNMENT-HK221-main
./os sched_0
Time slot   0
        Loaded a process at input/proc/s0, PID: 1 PRIO: 4
Time slot   1
        Loaded a process at input/proc/s0, PID: 2 PRIO: 0
        CPU 0: Dispatched process  1
Time slot   2
Time slot   3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   4
Time slot   5
        CPU 0: Put process  2 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  2
Time slot  10
Time slot  11
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  12
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  13
Time slot  14
Time slot  15
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  16
Time slot  17
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
```

```
Time slot  18
        CPU 0: Processed  2 has finished
        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
Time slot  24
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  25
Time slot  26
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  27
Time slot  28
        CPU 0: Put process  1 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  1
Time slot  31
        CPU 0: Processed  1 has finished
        CPU 0 stopped
```

1.2.2. Run sched_1

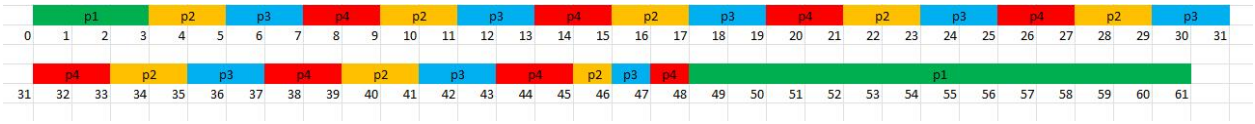
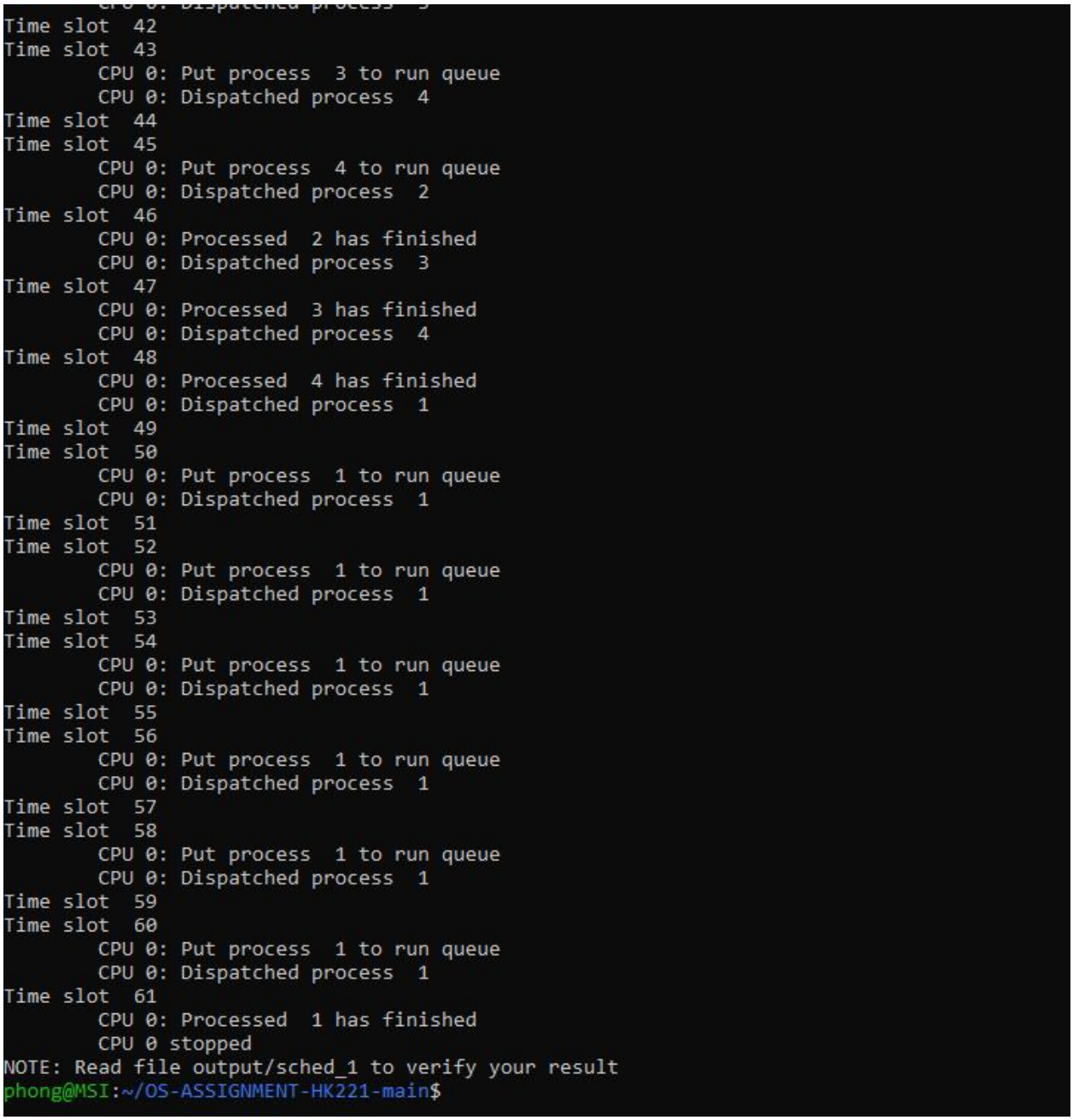


Figure 2. Gantt CPU executes processes - test sched 1

```
------ SCHEDULING TEST 1 --------------------------------------------
./os sched_1
Time slot   0
        Loaded a process at input/proc/s0, PID: 1 PRIO: 4
Time slot   1
        Loaded a process at input/proc/s0, PID: 2 PRIO: 0
        CPU 0: Dispatched process  1
Time slot   2
phong@MSI:~/OS-ASSIGNMENT-HK221-main$
Time slot   3
        Loaded a process at input/proc/s0, PID: 4 PRIO: 0
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   4
Time slot   5
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot   6
Time slot   7
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot   8
Time slot   9
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  10
Time slot  11
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  12
Time slot  13
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  14
Time slot  15
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  16
Time slot  17
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  18
Time slot  19
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  20
Time slot  21
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
```

```
Time slot  21
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  22
Time slot  23
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  24
Time slot  25
        CPU 0: Put process  3 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
Time slot  29
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  30
Time slot  31
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  32
Time slot  33
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  34
Time slot  35
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  36
Time slot  37
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  38
Time slot  39
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  40
Time slot  41
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  42
Time slot  43
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
```



1.3. Implementation

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

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 (!empty(q))
    {
        int max_i = 0;
        uint32_t max = q->proc[0]->priority;
        for (int i = 1; i < q->size; i++)
        {
            if (q->proc[i]->priority > max)
            {
                max = q->proc[i]->priority;
                max_i = i;
            }
        }
        struct pcb_t *maxPCB = (struct pcb_t *)malloc(sizeof(struct pcb_t));
        maxPCB = q->proc[max_i];

        for (int i = max_i; i < q->size - 1; i++)
        {
            q->proc[i] = q->proc[i + 1];
        }
        q->proc[q->size - 1] = NULL;
        q->size--;
        return maxPCB;
    }
    return NULL;
}

```

Specifically with the `enqueue()` function, we only add at the end of the queue if it is available (empty). With the `dequeue()` function, we search the process with the highest priority out, and at the same time update the queue's state when deleting an element. Below is the priority queue implementation for the scheduler.

```c
struct pcb_t *get_mlq_proc(void)
{
    struct pcb_t *proc = NULL;
    /*TODO: get a process from PRIORITY [ready_queue].
     * Remember to use lock to protect the queue.
     * */
    pthread_mutex_lock(&queue_lock);
    for (int prio = 0; prio < MAX_PRIO; prio++)
    {
        if (!empty(&mlq_ready_queue[prio]))
        {
            proc = dequeue(&mlq_ready_queue[prio]);
            break;
        }
    }
    pthread_mutex_unlock(&queue_lock);
    return proc;
}
```

Obtain a process from the [ready queue]. If the ready queue is empty, move all processes from the [run queue] to the [ready queue] and return the one with the greatest priority.

2. Memory management

2.1. Question

What is the advantage and disadvantage of segmentation with paging?

Answer:

	Segmentation	Paging
Advantages	- Simple to relocate segments than the entire address space. - The absence of internal fragmentation as external fragmentation has to be done. - The segment table is of lesser size compared with the page table in paging. - The average size of the segment is larger to the actual size of the page	- Frames do not have to be contiguous. - Easy to use memory management algorithm. - Swapping is easy between equal-sized pages and page frames.

Disadvantages	- Unequal size of segments is not good in the case of swapping. - Hard to allocate contiguous memory to partition as irt is of its variable size. - This is costly memory management algorithm.	- Paging causes internal fragmentation on older systems. - Longer memory lookup times compared to segmentation - It may cause internal fragmentation - Page tables consume additional memory. - Multi-level paging may lead to memory reference overhead.

2.2. Result

The two results below show the RAM status after each allocation and deallocation function call.

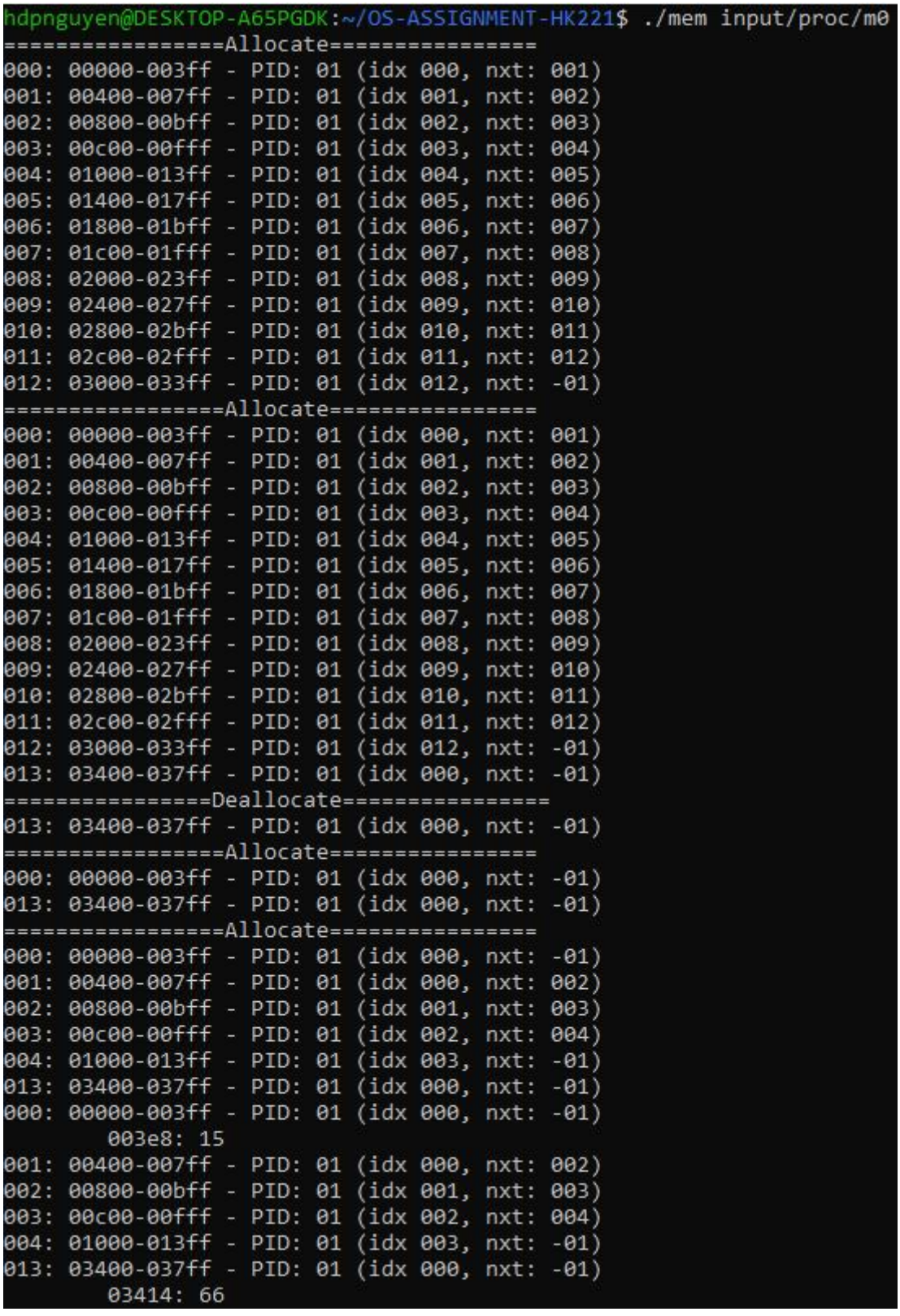


Figure 3. Test 0

```
hdpnguyen@DESKTOP-A65PGDK:~/OS-ASSIGNMENT-HK221$ ./mem input/proc/m1
=================Allocate================
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: -01)
=================Allocate================
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: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Deallocate================
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
=================Allocate================
000: 00000-003ff - PID: 01 (idx 000, nxt: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
=================Allocate================
000: 00000-003ff - PID: 01 (idx 000, nxt: -01)
001: 00400-007ff - PID: 01 (idx 000, nxt: 002)
002: 00800-00bff - PID: 01 (idx 001, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 002, nxt: 004)
004: 01000-013ff - PID: 01 (idx 003, nxt: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Deallocate================
001: 00400-007ff - PID: 01 (idx 000, nxt: 002)
002: 00800-00bff - PID: 01 (idx 001, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 002, nxt: 004)
004: 01000-013ff - PID: 01 (idx 003, nxt: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Deallocate================
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Deallocate================
hdpnguyen@DESKTOP-A65PGDK:~/OS-ASSIGNMENT-HK221$
```

Figure 4. Test 1

The figure below show the result of test 0 and 1 when run `make test_mem`

```
hdpnguyen@DESKTOP-A65PGDK:~/OS-ASSIGNMENT-HK221$ make mem
gcc -Iinclude  -Wall -c -g src/mem.c -o obj/mem.o
src/mem.c: In function ‘alloc_mem’:
src/mem.c:181:74: warning: assignment to ‘struct trans_table_t *’ from incompatible pointer type ‘struct page_table_t *’ [-Wincompatible-pointer-types
  181 |                                 proc->seg_table->table[first_lv].next_lv = (struct page_table_t *)malloc(sizeof(struct page_table_t));
      |                                                                           ^
gcc -Iinclude  -Wall -g obj/paging.o obj/mem.o obj/cpu.o obj/loader.o -o mem -lpthread
hdpnguyen@DESKTOP-A65PGDK:~/OS-ASSIGNMENT-HK221$ make test_mem
------ MEMORY MANAGEMENT TEST 0 ----------------------------------
./mem input/proc/m0
000: 00000-003ff - PID: 01 (idx 000, nxt: -01)
       003e8: 15
001: 00400-007ff - PID: 01 (idx 000, nxt: 002)
002: 00800-00bff - PID: 01 (idx 001, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 002, nxt: 004)
004: 01000-013ff - PID: 01 (idx 003, nxt: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
       03414: 66
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 print nothing)
```

2.3. Implementation

2.3.1. Find the paging table from the segment

By default, each address is represented by 20 bits, with the first 5 bits for segment index, the next 5 bits for page index and the last 10 bits for offset. This function accepts 5 bits segment index and 5 bits page_table segment table. Because the `page_table` table of segment is a structured list of `v_index` and `page_table_t`, where `v_index` is the 5-bit segment and `page_table_t` is the corresponding page table of that segment. Hence, we have to traverse this table, any `v_index` is equal to the index we need to find, we return to the corresponding `page_table`.

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

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

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

        return NULL;
}
```

2.3.2. Translate virtual address to physical address

Since each address contains 20 bits and the first 10 bits used for segment and page and the last 10 bits used for offset, the first 10 bits are taken to connect to the last 10 bits. Each `trans_table_t` stores elements with `p_index` of the first 10 bits. In order to create a physical address, the first 10 bits will be shifted to the left by 10-bit offset.

```
/* 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 trans_table_t *trans_table = NULL;
        trans_table = get_trans_table(first_lv, proc->seg_table);
        if (trans_table == NULL)
        {
                return 0;
        }

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

2.3.3. Allocate memory

2.3.3.1. Check the availabilty of the memory

We will check whether our memory is available on the physical memory or not. First, we traverse the segment table by shifting left to check if there are enough slots. Then, an array is created to store the index of available frames in physical memory. If there is a free page, the counter variable called `num_physical_page_avail` is increased by 1 and when we have enough physical memory, `mem_avail` is set to 1.

```c
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 : size / PAGE_SIZE + 1; // Number of pages we will use
        int mem_avail = 0;

        /* 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).
         * */

        /* Checking if there are enough slots available*/
        proc->seg_table->size = 1 << SEGMENT_LEN;
        int num_physical_page_avail = 0;

        /*array to store the index of available frame in physical memory*/
        int *avail_physical_index = (int *)malloc(sizeof(int) * (num_pages + 1));
        for (int i = 0; i < NUM_PAGES; i++)
        {
                if (_mem_stat[i].proc == 0)
                { // this page is free
                        avail_physical_index[num_physical_page_avail] = i;
                        num_physical_page_avail++;
                }
                if (num_physical_page_avail == num_pages)
                {
                        mem_avail = 1; // we have enough physiccal memory
                        break;
                }
        }
        if (proc->bp + num_pages * PAGE_SIZE >= RAM_SIZE)
                mem_avail = 0; // not logical memory
        /* end checking */
```

2.3.3.2. Allocate memory

After traversing on physical memory, finding free page and assigning this page to be used by the process, we traverse the page table. In this step, the `_mem_stat` is continually updated, when a page has not been initialized yet, we create it and lastly, the page table is updated for further access.

```
        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. */
                int loop = 0;

                for (; loop < num_pages; loop++)
                {
                        /* update the _mem_stat */
                        _mem_stat[avail_physical_index[loop]].proc = proc->pid;
                        _mem_stat[avail_physical_index[loop]].index = loop;
                        _mem_stat[avail_physical_index[loop]].next = avail_physical_index[loop + 1];
                        /* The first layer index */
                        addr_t first_lv = get_first_lv(proc->bp);
                        /* The second layer index */
                        addr_t second_lv = get_second_lv(proc->bp);
                        /* if the page haven't been initialized, then create it */
                        if (!proc->seg_table->table[first_lv].next_lv)
                        {
                                proc->seg_table->table[first_lv].next_lv = (struct page_table_t *)malloc(sizeof(struct page_table_t));
                                proc->seg_table->table[first_lv].next_lv->size = 1 << PAGE_LEN;
                        }
                        /* Update the page table fỏ further accesses */
                        proc->seg_table->table[first_lv].v_index = first_lv;
                        proc->seg_table->table[first_lv].next_lv->table[second_lv].v_index = second_lv;
                        proc->seg_table->table[first_lv].next_lv->table[second_lv].p_index = avail_physical_index[loop];
                        proc->bp += PAGE_SIZE;
                }
                _mem_stat[avail_physical_index[loop - 1]].next = -1; // the last element's next field is -1
        }
        if (1)
        {
                puts("================Allocate===============");
                dump();
        }
        // if you want to show memory actions
        pthread_mutex_unlock(&mem_lock);
        return ret_mem;
}
```

2.3.4. Free memory

At first, we get the first and second layer index of the page table and set `allow_to_free` as a flag to notify when the memory is free, the value 1 represents that the memory is allowed to free, otherwise 0. We recognize there are three cases with which the memory cannot be free, first is that when the address does not point to the first address of the allocated block, second is that other processes' memory is not also allowed to free and the last one is that the space which has not been allocated yet.

```c
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);
        /* The first layer index */
        addr_t first_lv = get_first_lv(address);
        /* The second layer index */
        addr_t second_lv = get_second_lv(address);
        /* First frame to free */
        addr_t frame_index = proc->seg_table->table[first_lv].next_lv->table[second_lv].p_index;

        int allow_to_free = 1;
        /* If the address not pointing to the first address of the allocated block*/
        if (_mem_stat[frame_index].index != 0)
                allow_to_free = 0;
        /* We also not allow to free other processes' memory */
        if (_mem_stat[frame_index].proc != proc->pid)
                allow_to_free = 0;
        /* Not to free the space that have not been allocated yet */
        if (_mem_stat[frame_index].proc == 0)
                allow_to_free = 0;

        while (allow_to_free)
        {
                _mem_stat[frame_index].proc = 0;
                frame_index = _mem_stat[frame_index].next;
                if (frame_index == -1)
                        break;
        }
        if (1)
        {
                puts("================Deallocate================");
                dump();
        }
        // if you want to show memory actions
        pthread_mutex_unlock(&mem_lock);
        return allow_to_free;
}
```

3. Put it all

After combining both scheduler and memory, we run `make all` to compile all the test

```
hdpnguyen@DESKTOP-A65PGDK:~/OS-ASSIGNMENT-HK221$ make clean
rm -f obj/*.o os sched mem
hdpnguyen@DESKTOP-A65PGDK:~/OS-ASSIGNMENT-HK221$ make sched
gcc -Iinclude  -Wall -c -g src/cpu.c -o obj/cpu.o
gcc -Iinclude  -Wall -c -g src/loader.c -o obj/loader.o
gcc -Iinclude  -Wall -c -g src/mem.c -o obj/mem.o
src/mem.c: In function ‘alloc_mem’:
src/mem.c:181:74: warning: assignment to ‘struct trans_table_t *’ from incompatible pointer type ‘struct page_table_t *’ [-Wincompatible-pointer-types]
  181 |                         proc->seg_table->table[first_lv].next_lv = (struct page_table_t *)malloc(sizeof(struct page_table_t));
      |                                                                          ^
gcc -Iinclude  -Wall -c -g src/queue.c -o obj/queue.o
gcc -Iinclude  -Wall -c -g src/os.c -o obj/os.o
gcc -Iinclude  -Wall -c -g src/sched.c -o obj/sched.o
gcc -Iinclude  -Wall -c -g src/timer.c -o obj/timer.o
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 -lpthread
hdpnguyen@DESKTOP-A65PGDK:~/OS-ASSIGNMENT-HK221$ make mem
gcc -Iinclude  -Wall -c -g src/paging.c -o obj/paging.o
gcc -Iinclude  -Wall -g obj/paging.o obj/mem.o obj/cpu.o obj/loader.o -o mem -lpthread
hdpnguyen@DESKTOP-A65PGDK:~/OS-ASSIGNMENT-HK221$ make all
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 -lpthread
```

Then, we run `make test_all` to get the result

```
ndpnguyen@DESKTOP-A65PGDK:~/OS-ASSIGNMENT-HK221$ make test_all
------ MEMORY MANAGEMENT TEST 0 ------------------------------
./mem input/proc/m0
================Allocate===============
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: -01)
================Allocate===============
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: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Deallocate==============
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Allocate===============
000: 00000-003ff - PID: 01 (idx 000, nxt: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Allocate===============
000: 00000-003ff - PID: 01 (idx 000, nxt: -01)
001: 00400-007ff - PID: 01 (idx 000, nxt: 002)
002: 00800-00bff - PID: 01 (idx 001, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 002, nxt: 004)
004: 01000-013ff - PID: 01 (idx 003, nxt: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
000: 00000-003ff - PID: 01 (idx 000, nxt: -01)
       003e8: 15
001: 00400-007ff - PID: 01 (idx 000, nxt: 002)
002: 00800-00bff - PID: 01 (idx 001, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 002, nxt: 004)
004: 01000-013ff - PID: 01 (idx 003, nxt: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
       03414: 66
NOTE: Read file output/m0 to verify your result
```

```
------ MEMORY MANAGEMENT TEST 1 ------------------------------
./mem input/proc/m1
================Allocate===============
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: -01)
================Allocate===============
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: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Deallocate==============
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Allocate===============
000: 00000-003ff - PID: 01 (idx 000, nxt: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Allocate===============
000: 00000-003ff - PID: 01 (idx 000, nxt: -01)
001: 00400-007ff - PID: 01 (idx 000, nxt: 002)
002: 00800-00bff - PID: 01 (idx 001, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 002, nxt: 004)
004: 01000-013ff - PID: 01 (idx 003, nxt: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Deallocate==============
001: 00400-007ff - PID: 01 (idx 000, nxt: 002)
002: 00800-00bff - PID: 01 (idx 001, nxt: 003)
003: 00c00-00fff - PID: 01 (idx 002, nxt: 004)
004: 01000-013ff - PID: 01 (idx 003, nxt: -01)
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Deallocate==============
013: 03400-037ff - PID: 01 (idx 000, nxt: -01)
================Deallocate==============
NOTE: Read file output/m1 to verify your result (your imple
```

```
------ SCHEDULING TEST 0 ----------------------------------------
./os sched_0
Time slot   0
        Loaded a process at input/proc/s0, PID: 1 PRIO: 4
Time slot   1
        CPU 0: Dispatched process  1
        Loaded a process at input/proc/s0, PID: 2 PRIO: 0
Time slot   2
Time slot   3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
Time slot   4
Time slot   5
        CPU 0: Put process  2 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  2
Time slot  10
Time slot  11
        CPU 0: Put process  2 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  2
Time slot  14
Time slot  15
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  16
Time slot  17
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  2
Time slot  18
        CPU 0: Processed  2 has finished
        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
Time slot  24
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  25
Time slot  26
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  27
Time slot  28
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
        CPU 0 stopped
NOTE: Read file output/sched_0 to verify your result
```

```
------ SCHEDULING TEST 1 ----------------------------------------
./os sched_1
Time slot   0
Time slot   1
        Loaded a process at input/proc/s0, PID: 2 PRIO: 0
        CPU 0: Dispatched process  1
Time slot   2
        Loaded a process at input/proc/s0, PID: 3 PRIO: 0
Time slot   3
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  2
        Loaded a process at input/proc/s0, PID: 4 PRIO: 0
Time slot   4
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot   6
Time slot   7
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot   8
Time slot   9
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  10
Time slot  11
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  12
Time slot  13
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  14
Time slot  15
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  16
Time slot  17
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  18
Time slot  19
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  20
Time slot  21
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  22
Time slot  23
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  24
Time slot  25
        CPU 0: Put process  3 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
Time slot  29
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  30
Time slot  31
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  32
Time slot  33
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  34
Time slot  35
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  36
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  38
Time slot  39
        CPU 0: Put process  4 to run queue
        CPU 0: Dispatched process  2
Time slot  40
Time slot  41
        CPU 0: Put process  2 to run queue
        CPU 0: Dispatched process  3
Time slot  42
Time slot  43
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  4
Time slot  44
Time slot  45
        CPU 0: Put process  4 to run queue
Time slot  46
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  3
Time slot  47
        CPU 0: Processed  3 has finished
        CPU 0: Dispatched process  4
Time slot  48
        CPU 0: Processed  4 has finished
        CPU 0: Dispatched process  1
Time slot  49
Time slot  50
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  53
Time slot  54
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  55
Time slot  56
        CPU 0: Put process  1 to run queue
Time slot  57
Time slot  58
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  59
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  1
Time slot  61
        CPU 0: Processed  1 has finished
        CPU 0 stopped
NOTE: Read file output/sched_1 to verify your result
```

```
[TDAHung: ./os os_0
Time slot   0
        Loaded a process at input/proc/p0, PID: 1 PRIO: 2
Time slot   1
        CPU 0: Dispatched process  1
        Loaded a process at input/proc/p0, PID: 2 PRIO: 0
        CPU 1: Dispatched process  2
Time slot   2
        Loaded a process at input/proc/p0, PID: 3 PRIO: 0
Time slot   3
        Loaded a process at input/proc/p0, PID: 4 PRIO: 0
Time slot   4
Time slot   5
Time slot   6
Time slot   7
        CPU 0: Put process  1 to run queue
        CPU 0: Dispatched process  3
Time slot   8
        CPU 1: Put process  2 to run queue
        CPU 1: Dispatched process  4
Time slot   9
Time slot  10
Time slot  11
Time slot  12
Time slot  13
        CPU 0: Put process  3 to run queue
        CPU 0: Dispatched process  2
Time slot  14
        CPU 1: Put process  4 to run queue
        CPU 1: Dispatched process  3
Time slot  15
Time slot  16
Time slot  17
        CPU 0: Processed  2 has finished
        CPU 0: Dispatched process  4
Time slot  18
        CPU 1: Processed  3 has finished
        CPU 1: Dispatched process  1
Time slot  19
Time slot  20
Time slot  21
        CPU 0: Processed  4 has finished
        CPU 0 stopped
Time slot  22
        CPU 1: Processed  1 has finished
        CPU 1 stopped
TDAHung:
```