VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

OPERATING SYSTEM (CO2018)

Assignment

SIMPLE OPERATING SYSTEM

Advisor: Nguyen Quang Hung
Students: Le Thong Minh Triet - 2053521.
Luc Gia Hung - 2053071.
Dang Nguyen Khanh - 2053109.

HO CHI MINH CITY, APRIL 2022

Contents

1 Member list & Workload

No.	Fullname	Student ID	Problems	Percentage of work
1	Le Thong Minh Triet	2053521	- Code Implementation. - Answer Course's Questions	40%
2	Dang Nguyen Khanh	2053521	- Scheduler. - Answer Course's Questions	30%
3	Luc Gia Hung	2053109	- Memory Management - Answer Course's Questions	30%

2 Scheduler

2.1 Question

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:

- Firstly, it is more flexible than multilevel queue scheduling. It allows different processes to move between different queues.
- To optimize turnaround time algorithms like SJF are needed which require the running time of processes to schedule them. But the running time of the process is not known in advance. FBQ runs a process for a time quantum and then it can change its priority(if it is a long process). Thus it learns from past behavior of the process and then predicts its future behavior. This way it tries to run a shorter process first thus optimizing turnaround time.
- FBQ also reduces the response time compared with FIFO or RR and prevents starvation.

2.2 Result - Gantt Diagram

Start	s0	s1	s0	s1	s0	s1	s0	s1	s0	End
0	1	5	7	9	11	13	15	17	18	23

Figure 1: Gantt Chart Sched 0

Start	s0	s1	s2	s3	s1	s2	s0	s3	s1	s2	s0	s3	s1	s2	s0	s3	s2	s0	s3	s2	s0	s3	s0	End
0	1	5	7	9	11	13	15	17	19	21	23	25	27	28	30	32	34	36	38	40	42	44	45	46

Figure 2: Gantt Chart Sched 1

2.3 Code Implementation

2.3.1 enqueue() And dequeue Function

```
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;
}

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)) return NULL;
    int highestPriority = q->proc[0]->priority;
    int idx = 0;
    for (int i = 1; i < q->size; i++) {
        if (q->proc[i]->priority > highestPriority) {
            highestPriority = q->proc[i]->priority;
            idx = i;
        }
    }
    struct pcb_t *tmp = q->proc[idx];
    for (int i = idx; i < q->size - 1; i++) {
        q->proc[i] = q->proc[i + 1];
    }
    q->size--;
    return tmp;
}
```

2.3.2 get_proc Function

```
struct pcb_t *get_proc(void) {
    if (queue_empty()) return 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);
    struct pcb_t *proc = NULL;
    if (empty(&ready_queue)) {
        while (!empty(&run_queue)) {
            enqueue(&ready_queue, dequeue(&run_queue));
        }
    }
    proc = dequeue(&ready_queue);
    pthread_mutex_unlock(&queue_lock);
    return proc;
}
```

3 Memory Management

3.1 Question

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:

Segmentation has been included in 80 x 86 microprocessors to encourage programmers to split their applications into logically related entities, such as subroutines or global and local data areas All Linux processes running in User Mode use the same pair of segments to address instructions and data. These segments are called user code segment and user data segment , respectively. Similarly, all Linux processes running in Kernel Mode use the same pair of segments to address instructions and data: they are called kernel code segment and kernel data segment , respectively. Linux prefers paging to segmentation for the following reasons:

- Advantages
  1. The page table size is reduced as pages are present only for data of segments, hence reducing the memory requirements
  2. Reduces external fragmentation in comparison with segmentation.
  3. Since the entire segment need not be swapped out, the swapping out into virtual memory becomes easier .
- Disadvantages
  1. Internal fragmentation still exists in pages.
  2. Extra hardware is required
  3. Translation becomes more sequential increasing the memory access time.

3.2 Result - RAM Content

TEST 0

```
=========================Allocated memory region=========================
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)
=========================Allocated memory region=========================
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)
========================Free memory region========================
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
========================Allocated memory region========================
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)
========================Allocated memory region========================
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: 14
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: 64
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
```

TEST 1

```
===========================Allocated memory region===========================
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)
===========================Allocated memory region===========================
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)
===========================Free memory region===========================
```

```
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
==========================Allocated memory region==========================
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)
==========================Allocated memory region==========================
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)
==========================Free memory region==========================
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 memory region==========================
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
==========================Free memory region==========================
```

3.3 Code Implementation

3.3.1 get_page_table() Function

```
/* Search for page table from the 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
     *
     * */
    for (int i = 0; i < 1 << PAGE_LEN; i++) {
        if (seg_table->table[i].v_index == index) {
            if (seg_table->table[i].pages == NULL) {
                seg_table->table[i].pages = (struct page_table_t *) malloc(
                        sizeof(struct page_table_t) * (1 << PAGE_LEN));
            }
            return seg_table->table[i].pages;
        }
    }
    return NULL;
}
```

3.3.2 translate() Function

```
/* 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 < 1 << PAGE_LEN; i++) {
        if (page_table->table[i].v_index == second_lv) {
            /* TODO: Concatenate the offset of the virtual address
             * 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;
}
```

3.3.3 alloc_mem() Function

```
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).
 * */
//check amount of pages in physical address space
uint32_t check_num_pages = num_pages;
for (int i = 0; i < NUM_PAGES; i++) {
    if (_mem_stat[i].proc == 0) {
        check_num_pages--;
    }
    if (check_num_pages == 0 && proc->bp + num_pages * PAGE_SIZE <= RAM_SIZE) {
        mem_avail = 1;
        break;
    }
}

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 start_index = 0;
int prev_index = 0;
for (int i = 0; i < NUM_PAGES; i++) {
    if (_mem_stat[i].proc != 0) continue;
    _mem_stat[i].proc = proc->pid;
    if (start_index) {
        _mem_stat[prev_index].next = i;
    }
    _mem_stat[i].index = start_index;
    prev_index = i;

    /* Add entries to segment table page tables of [proc]
 * to ensure accesses to allocated memory slot is
 * valid. */
    struct seg_table_t *seg_table = proc->seg_table;
    if (seg_table->table[0].pages == NULL)
        seg_table->size = 0;
    addr_t curr_virt_addr = ret_mem + start_index * PAGE_SIZE;
    addr_t seg_idx = get_first_lv(curr_virt_addr);
    addr_t page_idx = get_second_lv(curr_virt_addr);
    int contain_page_table = 0;
    for (int sub_seg_idx = 0;
    sub_seg_idx < seg_table->size; sub_seg_idx++) {
        if (seg_table->table[sub_seg_idx].v_index == seg_idx) {
            struct page_table_t *curr_page_table =
            seg_table->table[sub_seg_idx].pages;
            curr_page_table->table[curr_page_table->size].v_index =
            page_idx;
            curr_page_table->table[curr_page_table->size++].p_index = i;
            contain_page_table = 1;
```

```
                break;
            }
        }
        if (!contain_page_table) {
            seg_table->table[seg_table->size].v_index = seg_idx;
            seg_table->table[seg_table->size].pages =
            (struct page_table_t *) malloc(sizeof(struct page_table_t));
            seg_table->table[seg_table->size].pages->size = 1;
            seg_table->table[seg_table->size].pages->table[0].v_index =
            page_idx;
            seg_table->table[seg_table->size++].pages->table[0].p_index = i;
        }
        start_index++;
        if (start_index == num_pages) {
            _mem_stat[i].next = -1;
            break;
        }
    }

  }
  pthread_mutex_unlock(&mem_lock);
  return ret_mem;
}
```

3.3.4 free_mem() Function

```
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);
int valid = 0;
struct page_table_t *page_table = get_page_table(get_first_lv(address),
proc->seg_table);
if (page_table) { // found page table
    int page_idx = get_second_lv(address);
    for (int i = 0; i < page_table->size; i++) {
        if (page_table->table[i].v_index == page_idx) { //found page
            int p_index = page_table->table[i].p_index;
            int num_free_pages = 0;
            addr_t cur_vir_addr = (num_free_pages << OFFSET_LEN) + address;
            do {
                _mem_stat[p_index].proc = 0;
                addr_t seg_idx = get_first_lv(cur_vir_addr);
                addr_t page_idx = get_second_lv(cur_vir_addr);
                for (int sub_table_idx = 0;
                sub_table_idx < proc->seg_table->size;
                sub_table_idx++) {
                    if (proc->seg_table->table[sub_table_idx].v_index
                    == seg_idx) {
                        for (int sub_page_idx =0;
                        sub_page_idx <
                        proc->seg_table->table[sub_table_idx].pages->size;
                        sub_page_idx++) {
                            if (proc->seg_table->table[sub_table_idx].pages
                            ->table[sub_page_idx].v_index == page_idx) {
                                for (int j = sub_page_idx;
                                j<
                                proc->seg_table->
                                table[sub_table_idx].pages->size - 1
                                ; j++) {
```

```
                        proc->seg_table->table[sub_table_idx].pages->table[j]
                        = proc->seg_table->table[sub_table_idx].pages->table[j + 1];
                                    } //remove unused page
                        proc->seg_table->table[sub_table_idx].pages->size--;
                                    if (proc->seg_table->table[sub_table_idx].pages
                                    ->size == 0) {
                                        for (int j = sub_table_idx; j <
                                        proc->seg_table->size - 1; j++) {
                                            proc->seg_table->table[j] =
                                            proc->seg_table->table[j + 1];
                                        }
                                        proc->seg_table->size--;
                                        //remove unused segment
                                    }
                                    break;
                                }
                            }
                        }
                    }
                    num_free_pages++;
                    p_index = _mem_stat[p_index].next;
                } while (p_index != -1);
                valid = 1;
            }
        }
    }
    pthread_mutex_unlock(&mem_lock);
    if(!valid)
        return 1;
    else
        return 0;
}
```

4 Synchronization

4.1 Question

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

Answer:

- Causes data loss, data errors during input and output, for example: there is an input program (producer), data transfer into a limited buffer and a program that retrieves and removes data in the buffer (consumer), if the producer inputs and transfers data to the buffer when the buffer is full, it will cause data loss, if the consumer gets data when the buffer is empty, it will make the output data different from the input data.

- Causing data collisions, resources are not properly distributed, for example: processes operate independently and have some data sources and resources used in common, if there is no synchronization process, it will cause problems. If two or more processes are active, modifying the same data at the same time can cause errors for other processes using that resource.

References

[1] James L Peterson and Abraham Silberschatz. Operating system concepts. Addison-Wesley Longman Publishing Co., Inc., 1985.