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

COURSE : OPERATING SYSTEM

Assignment

Simple Operating System

Advisors: Nguyễn Lê Duy Lai
Trần Văn Hoài

Students: Lê Đức Cầm - 1952588
Cao Bá Huy - 1952713
Ngô Minh Đại - 1913008

HO CHI MINH CITY, MAY 2021

Contents

1 Scheduling

1.1 Question for Priority feedback queue

Question : What is the advantage of using priority feedback queue in comparison with other scheduling algorithms you have learned ?
Answer :
The priority feedback queue algorithm (PFQ) is the combination of other scheduling algorithms like Priority Scheduling , Multilevel Queue and Round - robin algorithm. To be specific, the Priority feedback queue algorithm uses 2 queues with the following functions :

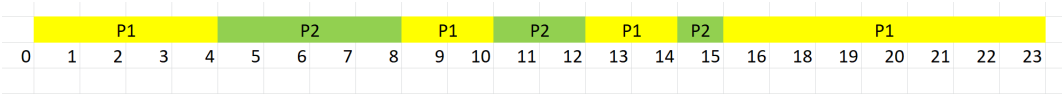
- *ready_queue* : contains all the processes along with the priority to be executed first compared to the one in *run_queue*. When the CPU moves to the next time slot, it will take of process form this *ready_queue* .

- *run_queue* : contains the processes which have used the time slot but not finished. All the processes from this queue get to be executed only if the *ready_queue* is empty.

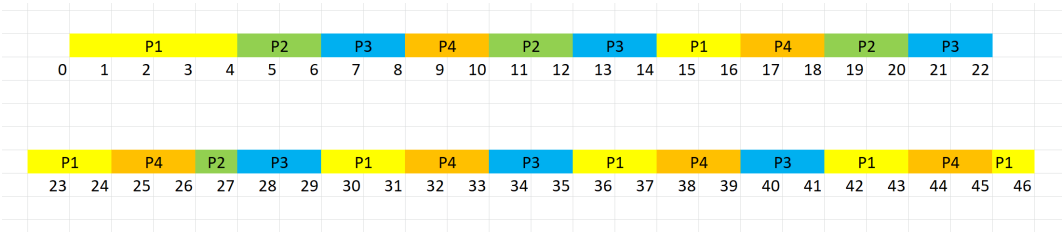
The advantages of Priority Feedback Queue
- Improve the performance as the process which has the highest priority will be executed first .
- We can prevent the starvation due to the time slot mechanism. If the high priority process takes too much time , it will be moved to the *run_queue* and lower priority process will be executed next.
- The algorithm uses time slot like the Round-Robin algorithm with a quantum time for each process. Therefore, we can improve the average response time and ensure that it can cycle through all ready tasks, giving each one a chance to run .
- Allows processes to move between queues so it is more flexible .

1.2 Gantt chart for the output

Requirement : Draw Gantt diagram describing how processes are executed by the CPU.
For test 0 :



For test 1 :



1.3 Implementation code

1.3.1 Priority queue

For the enqueue() function, since the maximum processes for a queue to handle is 10 so we will simply add a new process if the size of queue is less than 10 only.

```
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;
   q->size++ ;
}
```

For the dequeue() function:
+First step: look for the position of the process with highest priority, back up its index in the queue.
+Second step: return that process from the queue and update the 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; /* Index of the process whose priority is biggest*/
       uint32_t max = q->proc[0]->priority; /* max is the largest 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; i++){
           q->proc[i] = q->proc[i + 1];
        }
        q->proc[q->size - 1] = NULL;
        q->size--;
        return maxPCB;
    }
    return NULL;
}
```

1.3.2 Scheduler

In this assignment, the scheduler holds the function of updating the processes for the CPU to execute. If the *ready_queue* is empty then we will move all the processes from the *run_queue* to it. Otherwise, we will take the highest priority process from the *ready_queue* to the CPU. The function *get_proc*() will perform this task :

```
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.
     * */
    if(queue_empty()) return NULL;
    if (empty(&ready_queue)){ // If ready_queue is empty then push processed in
     run_queue back to it
       while (!empty(&run_queue)){
          pthread_mutex_lock(&queue_lock);
          struct pcb_t * cur = (struct pcb_t *)malloc(sizeof(struct pcb_t));
          cur = dequeue(&run_queue);
          enqueue(&ready_queue, cur);
          pthread_mutex_unlock(&queue_lock);
       }
    }
    pthread_mutex_lock(&queue_lock);
    proc = dequeue(&ready_queue);
    pthread_mutex_unlock(&queue_lock);

    return proc;
}
```

2 Memory

2.1 Question for segmentation with paging

Question : What is the advantage and disadvantage of segmentation with paging?
Answer :
Advantages:
- It reduces memory usage, which leads to the efficient use of memory.
- Segmentation allows for the sharing of procedures.
- Page table size is limited by the segment size.
- Segment page table has only one entry corresponding to one physical segment.
- External Fragmentation is not there.
- It simplifies memory allocation.

Disadvantages:
- It may lead to external fragmentation.
- The complexity level will be much higher as compare to paging.
- Segmentation is slower than paging method
- Un-equal size of segments is not good in the case of swapping.

2.2 The status of RAM

Show the status of RAM after each memory allocation and deallocation fuction call
Answer :

```
    ------ MEMORY MANAGEMENT TEST 0 -----------------------------------
./mem input/proc/m0
---------Allocated ----------
Before
------------------------------
after
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 ----------
Before
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)
-----------------------------
after
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)
-----------------------------
-------DeAllocated ----------
Before
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)
------------------------------
after
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
------------------------------
---------Allocated ----------
Before
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
------------------------------
after
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 ----------
Before
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)
------------------------------
after
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)
------------------------------
-----------Final dump--------------
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)
```

```
    ------ MEMORY MANAGEMENT TEST 1 ----------------------------------
./mem input/proc/m1
---------Allocated ----------
Before
-----------------------------
after
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 ----------
Before
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)
-----------------------------
```

```
after
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)
------------------------------
-------DeAllocated ----------
Before
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)
------------------------------
after
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
------------------------------
---------Allocated ----------
Before
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
------------------------------
after
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 ----------
Before
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)
------------------------------
after
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)
------------------------------
-------DeAllocated ----------
Before
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)
------------------------------
after
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)
------------------------------
-------DeAllocated ----------
Before
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)
------------------------------
after
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
------------------------------
-------DeAllocated ----------
Before
014: 03800-03bff - PID: 01 (idx 000, nxt: 015)
015: 03c00-03fff - PID: 01 (idx 001, nxt: -01)
------------------------------
after
------------------------------
-----------Final dump--------------
```

2.3 Implementation code

2.3.1 Find the page table in segment table

In the scope of the assignment, the size of virtual RAM is 1 MB so we must use 20 bit to represent the address of each of its byte. With the segmentation with paging mechanism, we use the first 5 bits for segment index, the next 5 bits for page index and the last 10 bits for offset.
In the C program mem.c, we have the function *get_page_table* (as shown below)

```c
/* Search for page 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
   if (seg_table == NULL) return NULL;
   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;
}
```

The mission of this function is to receive 5 segment index bits and the first level table *seg_table*. On the other hand, *seg_table* is an array of the elements each of which has the structure $(v\_index, page\_table\_t)$

```
struct seg_table_t {
   /* Translation table for the first layer */
   struct {
      addr_t v_index;   // Virtual index
      struct page_table_t * pages;
   } table[1 << PAGE_LEN];
   int size;   // Number of row in the first layer
};
```

So, in order to find the page table, we must sort the segment table *seg_table*, the element which has the virtual index *v_index* equal to the 5 bit index that function received *addr_t index*, we return the corresponding page table.

2.3.2 Translate a virtual address to physical address

In the C program mem.c, we have the function *translate* (as shown below)

```
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) {
        *physical_addr = (page_table->table[i].p_index << OFFSET_LEN) + offset;
         return 1;
      }
   }
   return 0;
}
```

Because each page table $page\_table\_t$ is an array of the elements each of which has the structure (v_index, p_index) (a virtual index mapped with its respective physical index).

```
struct page_table_t {
   /* A row in the page table of the second layer */
   struct  {
      addr_t v_index; // The index of virtual address
      addr_t p_index; // The index of physical address
   } table[1 << SEGMENT_LEN];
   int size;
};
```

Each address is the combination of 20 bits (as mentioned above) so in order to translate the physical address, we must shift left the first 10 bit to get the physical frame of the physical memory $[p\_index]$ and add with the $offset$ of the virtual address to produce the correct physical address and save it to $[*physical\_addr]$.

2.3.3 Allocate memory

2.3.3.a Check available memory

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 $[mem\_avail]$. Otherwise, set 0. We have the following source code.

```
    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?
    int free_page_index[num_pages]; // List of index of page we will use
    int i = 0; // Index in free_page_index[]
    int j = 0; // Index in _mem_stat[]
    //Loop through _mem_stat[NUM_PAGES] to find free page and store its index
    to free_page_index[]
    while (i < num_pages) {
        while (j < NUM_PAGES) {
            if (_mem_stat[j].proc == 0) {
                free_page_index[i++] = j++;
                break;
            }
            j++;
        }
        if (j == NUM_PAGES) break;
    }
```

```
/* Check if enough memory in physical address space is large enough*/
 if (i == num_pages  && (proc->bp + num_pages * PAGE_SIZE <= (1 <<
ADDRESS_SIZE))) {// Enough page required in both physical mem and virtual
mem
    mem_avail = 1;
}
```

2.3.3.b Allocate memory

Before allocating process to physical memory, we checked the case that empty segment table occurs (for example: at the beginning)

```
if(proc->seg_table->size == 0){ // empty segtable
   proc->seg_table->size +=1;
   proc->seg_table->table[0].pages =  malloc(sizeof(struct page_table_t));
   proc->seg_table->table[0].pages->size = 0;
}
```

If the memory is not enough, we must allocate it. The algorithm is first to sort the physical address, find the free page, it means that we find the *_mem_stat* which has *procID* equal zero. The following code updates the 5 bit *index* of this page and the *next* variable for the next page in the list then assigns the sign *used* by the process.

```
for (i = 0; i < num_pages; i++){
        /* Update [proc], [index] and [next] field */
        _mem_stat[free_page_index[i]].proc = proc->pid;
        _mem_stat[free_page_index[i]].index = i;
        if( i == (num_pages -1)){
            _mem_stat[free_page_index[i]].next = -1;
        }
        else{
            _mem_stat[free_page_index[i]].next = free_page_index[i+1];
        }
...
```

Furthermore, we also implement some code for the unexpected case: the page_table is full then add new table in seg_table

```
int seg_table_size = proc->seg_table->size;
    /* If the page_table is full then add new entry to seg_table */
```

```
    if (proc->seg_table->table[seg_table_size-1].pages->size == (1 << PAGE_LEN)
    ) {
          proc->seg_table->size ++;
          seg_table_size ++;
          proc->seg_table->table[seg_table_size-1].pages = malloc(sizeof(struct
     page_table_t));
          proc->seg_table->table[seg_table_size-1].pages->size = 0;
    }
```

Then we must add the entries of all below [*proc*] to segment table *seg_table* and page table *page_table*. We must allocate the dynamical array for the struct page table *page_table_t* and then add an entry to it. We implenment the algorith as the code shown below.

```
    ...
    struct page_table_t * page_table = proc->seg_table->table[seg_table_size
    -1].pages;

       // perform add

      proc->seg_table->table[seg_table_size].v_index = get_first_lv(old_bp);
      page_table->table[page_table->size].v_index = get_second_lv(old_bp);
      page_table->table[page_table->size].p_index = free_page_index[i];
      page_table->size ++;
      old_bp += PAGE_SIZE;
    ...
```

2.3.4 Deallocate memory

- **Step 1 :** Translate logical address into physical address

- **Step 2 :** Base on physical page (need to be deallocated) to check if there are some pages left (if physical_page == -1 then finish free-memory process and go to step 6).

- **Step 3 :** Set ID of the process[proc] used in the physical memory back to zero.

- **Step 4 :** Loop through the page_table inside the segment table to find and update the virtual address and physical address of the page which was deallocated to "-1".

- **Step 5 :** Update the virtual address and physical page, then go to the second step.

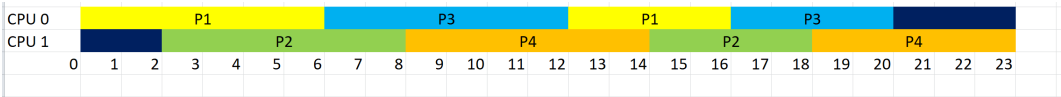
- **Step 6 :** : unlock and return

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

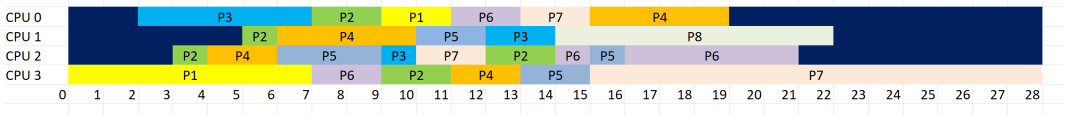
```
    pthread_mutex_lock(&mem_lock);
    addr_t physical_addr = -1;
    addr_t virtual_addr = address;
    if(translate(virtual_addr, &physical_addr, proc)) {
        addr_t physical_page = physical_addr >> OFFSET_LEN;
        while(physical_page != -1){
            _mem_stat[physical_page].proc = 0; // set[proc] of physical page
    back to 0

            struct page_table_t * page_table = NULL;
            page_table = get_page_table(get_first_lv(virtual_addr), proc->
    seg_table);
            if (page_table == NULL)   break;
            // find the page in segment-table to remove
            for (int i = 0; i < page_table->size; i++) {
              if( get_second_lv(virtual_addr) == page_table->table[i].v_index){
                    page_table->table[i].v_index=-1;
                    page_table->table[i].p_index=-1;
                    break;
                }
            }

            virtual_addr += PAGE_SIZE;
            physical_page = _mem_stat[physical_page].next;
        }
    }
    pthread_mutex_unlock(&mem_lock);
   return 0;
}
```

3 Overall

We do the "make all" and "make test_all", because of running multiple processes we got several result after doing many times, and the figure below will illustrate our ouput for this assignment:
For test 0



For test 1


CPU 0
CPU 1
CPU 2
CPU 3
P3
P2
P1
P6
P7
P4
P2
P4
P5
P3
P8
P2
P4
P5
P3
P7
P2
P6
P5
P6
P1
P6
P2
P4
P5
P7
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28